Snapdragon: A multipurpose library for *Mathematica*

User's manual

Stanislav Hledík
Silesian University in Opava
Institute of Physics

Mathematica 11.3+ compatible
Updated for *Mathematica* 12.0

This page intentionally left blank

Contents

Running this code creates two buttons. Clicking any button will open a separate notebook window with the table of contents (standard or condensed). The page numbers apply only if this notebook is fully evaluated, the hyperlinks are, however, correctly linked for any status of evaluation.

In[3]:=
```
Row[
 Button[#[[2]], NotebookOpen[StringDrop[NotebookFileName[], -3] <>
       "_TOC_" <> #[[1]] <> ".nb"]] & /@
  {{"Book", "Open TOC"}, {"BookCondensed", "Open TOC (condensed)"}},
 Spacer@8]
```

Out[3]= 

Proper functioning requires both table of contents notebooks to be stored in the same directory as this notebook. The table of contents can be regenerated as described in Appendix C: Creating table of contents.

Notice

The table of contents on this page is only useful when viewing this source notebook in *Mathematica*. The usual content useful for viewing the PDF export is available in Appendix C: Creating table of contents.

This page intentionally left blank

Preface

The history and future of Snapdragon

I had been using *Mathematica* occasionally in irregular intervals since early 1990's, usually for less serious tasks like graph plotting and image creating that did not require even an insignificant fraction of the full power of the language. I still remember when I first encountered version 2.x that needed plugging a hardware key (that had been supplied together with the installation media and a printed manual inside a box) somewhere into the rear bay port of the computer to bring *Mathematica* to life. Only much later, about eight years ago, during and after participation in five Wolfram Technology Conferences, I was coerced to delve deeper into study of the *Wolfram Language* (here I am due to make a comment that the language was officially named in 2013 despite it had been in use since *Mathematica*'s initial release in 1988 as the programming language of *Mathematica*) in the framework of a project which then was rather outside of my main field of activity (namely, biomedical sciences). Only then I have been able to fully unleash the power of the *Wolfram Language* (the fact that was well known to many people, contrary to me) and, maybe more importantly, that *Wolfram Language*, particularly when one strictly adheres to principles of functional pattern and rule-based programming, incredibly facilitates the transfer of mathematical (and others) ideas from one's head to a usable, concise and manageable code.

Since then, I have been involved in *Wolfram Language* programming; the knowledge I managed to absorb I then used in my fields of interest (astrophysics, signal processing), certain secondary biomedicine-related projects, and, last but not least, for my own amusement in the field of recreational math. Currently I use *Mathematica* regularly, and I frequently need the full strength it can provide me, in terms of speed, numerical and symbolic capabilities.

I have gradually developed several functions intended solely for my private use, aiming to simplify and facilitate my work. As it often happens in such cases, I copied them from one notebook to another (usually placing them in the initialization cells at the end of a notebook), making emendations here, there and everythere, until I got to a state of having several versions of the same code scattered throughout of disk, without any idea about location of the most recent or the most developed version. As the number of such function started growing, I realized that this is not a smart way how to manage my *Mathematica* "treasure." That finally led me to a decision to invest an extra effort into collecting the latest versions, carrying on testing and bulletproofing, and creating my own *Mathematica* application—mainly for myself, secondly, for my teaching activities, of course for clear pleasure that a perfectionist like me can draw from accumulating and cleaning the code I have written so far, and, last but not least, to make the intended manual accessible (and possibly useful) for other people as well.

If I should briefly describe the application described by this manual, I would put it in these words: Snapdragon is an eclectic multipurpose cross-disciplinary collection covering variety of features

in diverse fields of science, technology, engineering and mathematics that—from the point of view of my own needs and solely for my own work—either were found to be desirable but currently not available among *Mathematica* built-in system functions, or if so, were tailored for particular, frequently used purposes or dedicated actions (see also the Snapdragon usage statement in the Getting section below).

Finally: why "Snapdragon"? Of course, there is no connection with the Qualcomm's Snapdragon mobile processors [1]. This name was actually chosen because the author's surname coincides with the Czech folk term for Antirrhinum—a genus of plants commonly known as "dragon flowers" or "snapdragons" because of the flowers' fancied resemblance to the face of a dragon (Boca de Dragón in Spanish) that opens and closes its mouth when laterally squeezed [2].

The audience for this manual

I would like to point out at the very beginning that this manual is neither a textbook nor a substitute for general *Mathematica* manual or *Wolfram Language* & System Documentation Center. The opposite is true—using this manual expects the reader has got a essential familiarity with *Mathematica*, on the level of, e.g. [Wolfram, 2017; Wellin et al., 2005; Wellin, 2013; Ruskeepää, 2019; Shifrin, 2009]. There are also certified teaching materials that can be integrated into the *Mathematica* menu system [Wellin & Calkins, 2011]. Czech readers can use teaching manuals [Friedrich, 2013; Říha et al., 2011]. A considerable amount of inspiring trifles can also be found in [Mangano, 2010]. There are monographs that use *Mathematica* as a basic or complementary tool for expressing ideas of its main focus, for example [Baumann, 2005a; Baumann, 2005b; Gregory, 2005]. Prior programming experience would be quite advantageous (although not absolutely necessary) since discussion of the basic programming concepts is (intentionally) omitted. In particular, while I present examples of the code and discuss their usage (and sometimes quirks) along the way, I do not systematically introduce the syntax from the ground up—this is left to the reader. With a little exaggeration I can say that I fully rely on the proven saying that reading the source code is the best documentation. On the other way, many examples or discussions have been written with teaching purpose in sight.

Please don't hesitate to contact me with any comments, suggestions or (preferably constructive) criticism—I will make all my possible efforts to take them into account and use them for improving the value of the manual. You can contact me via e-mail address hlediks@gmail.com, or at my Tumblr account [Hledík, 2019a].

Disclaimer

Except when explicitly stated otherwise, all implementations of the Snapdragon functions are exclusively mine. All examples within this manual and other documentation were tested under *Mathematica* 11.3+. I am the only one responsible for any errors present in the code. Although I made an considerable effort to ensure that it is reasonably bug-free and bulletproofed, the code however almost certainly does contain some bugs. If you find any, I will be happy to learn about it from you to get rid of it in the future versions. If you decide to use Snapdragon for whatever

purposes, however, do it at your own risk.

All the text and code in this manual is provided *as is*, with no warranty of any kind, expressed or implied. Neither I nor Wolfram Research Inc. will be liable, under any circumstances, in any loss of any form, direct or indirect, related to or caused by the use of any of the materials contained in this manual.

Acknowledgements

This manual was created with the financial support of the project "Rozvoj vzdělávání na Slezské univerzitě v Opavě, reg. č. CZ.02.2.69/0.0/0.0/16_015/0002400". I am grateful to Josef Juráň for having once shown me the basics of *Mathematica*, and to my friends Zdeněk Buk, Jan Hurt, Filip Novotný, Antonín Slavík and Václav Žák for stimulating discussions.

I am also grateful to the Silesian University in Opava ("Slezská univerzita v Opavě" in Czech) for providing me with an access to the most recent versions of *Mathematica* and for equipping me with computers to run *Mathematica* on.

The contribution [Kubetta, 2012] proved to be an indispensable source of information in my attempts to convert *Mathematica* into a text processing tool that would at least slightly match the quality of the LaTeX typesetting system.

Finally, I am indebted to many contributors at [Mathematica Stack Exchange, 2019], a Q&A site for users of Mathematica and the Wolfram Language. I do not provide their full list here (with honorable exceptions of [Horvát, 2016; Horvát, 2017a; Horvát, 2017b; Horvát, 2017c; Jason, 2014]), but I must admit that the solutions—often very smart and ingenious—considerably contributed to this manual.

Preliminaries

Snapdragon application getting and installing

Distribution

Snapdragon is distributed as a paclet [Horvát, 2017a] which is package management native format of *Mathematica* (not yet officially documented at the time of writing this manual, more information can be found on the pages [Horvát, 2016; Horvát, 2017b]).

Download

Visit my repository [Hledík, 2019b], move to the Paclets folder, and click the file Snapdragon-M.m.R.paclet. If multiple such files are present, choose the latest one (first look for the greatest major version M, for multiple files sharing the same M choose the greatest release number R [3]). Then use *Download* ▷ *Direct download* and save [4]. In the same way download the file PacletInstaller.nb located along with the paclet.

Installation

After opening PacletInstaller.nb in *Mathematica* follow the instructions therein—running the code in the Selecting and Installing sections is enough. This will install Snapdragon in your $UserBasePacletsDirectory (which is the same as $UserBaseDirectory/Paclets [5]), and I strongly recommend that you stick to this convention. If you would prefer the system-wide installation, relevant instructions can be found in the Appendix A: System-wide installation. Although system-wide installation is possible, you should however avoid it: doing so is at your own risk.

Loading

Use standard system function Get or Needs. You should see something like "This is Snapdragon, Version M.m.R" with M, m, R replaced with numbers pertinent to the version just having been loaded [6].

```
In[4]:= << Snapdragon`

        This is Snapdragon, Version 31.3.191407
```

Observe the match between the major version reported on loading and the total number of symbols provided by the application:

In[5]:=
```
infoAbout@"Snapdragon"
Names["Snapdragon`s*"]
Length@%
```

> Snapdragon is an eclectic multipurpose cross-disciplinary collection covering variety of features in diverse fields of STEM that – from the point of view of the author's own needs and solely for his own work – either were found to be desirable but currently not available among *Mathematica* built-in system functions, or if so, were tailored for particular, frequently used purposes or dedicated actions.
> Upon loading 'This is Snapdragon, Version M.m.R' appears; the major version M must agree with Length@Names["Snapdragon`s*"] output.

```
Attributes[Snapdragon] = {Protected, ReadProtected}

SyntaxInformation[Snapdragon] = {ArgumentsPattern → {_, _, _}}
```

Out[6]=
```
{sBoole, sBulkExport, sCircularlySample, sCVectorDFTUnpack, sDiagonals,
 sDimensionsBoxed, sDirectoryDigest, sEmptyListsQ, sENextPow2, sENextPow2N,
 sExprOccupancy, sFactorExactNumber, sFileSelect, sGrandiSequence,
 sHeadsOptionCheck, sListDecimate, sListPush, sListTruncate, sNextPow2,
 sNextPow2N, sParseIdNumber, sPushKeyUp, sRealFourier, sReferTo,
 sRotateHalfLength, sRVectorCPack, sSamplingSpan, sSavGolBufferConvolve1D,
 sSavGolElasticKernels1D, sSavGolKernel1D, sUniqueName}
```

Out[7]=
```
31
```

As you may have noticed, all symbol names start with lowercase 's' (with the only exception of the symbol Snapdragon that does not represent any function but rather the application annotation as displayed above). This measure has been adopted to avoid namespace collisions.

Uninstallation

Open the installer file PacletInstaller.nb in *Mathematica* and follow instructions therein (using sections Selecting and Uninstalling will suffice for this task). *Never* try to uninstall Snapdragon by deleting its base directory or directories $UserBasePacletsDirectory/Snapdragon-M.m.R.

The organization of the manual

The manual is organized into five chapters, each corresponding to an area which the functions go best with. For the purpose of this manual, the following division sounded reasonable to me:

- Chapter 1: List and array manipulations—includes functions accepting a list, an array, or in particular cases a general expression as the first argument. Their main purpose are various manipulations with lists and expressions. Some of them may ambiguously fall into a different category, typically those included in Chapter 2.

- Chapter 2: Signal analysis—mostly devoted to functions facilitating sampling, spectral analysis, convolving and related topics that fall into the field of signal analysis and processing. Conversely, some of them may ambiguously fall into Chapter 1.
- Chapter 3: Files, import, export—introduces functions dealing with facilitating export, import and file handling. Hopefully this chapter could be found useful by those who frequently find files or directories interactively.
- Chapter 4: Miscellanea—introduces functions that do not fit into either of the above chapters.

Each chapter is partitioned into sections eponymous to the symbol described. Every section begins with a brief synopsis of the described symbol using dedicated function infoAbout based on the Information built-in, e.g. infoAbout@sSymb or infoAbout[sSymb, 0.6] for symbol sSymb (see explanation in Appendix D: Initialization), followed by

- subsection Details explaining more thoroughly syntax, usage and available options,
- subsection Examples demonstrating basic behavior,
- and subsection Application containing typical examples—or at least outlines—of usage.
- Some symbol sections may involve optional subsection Discussion in which more material of interest may be placed: typically it is devoted to performance comparison or discussing implemented methods.

Documentation for the Snapdragon package symbols is not integrated with *Mathematica*'s documentation center yet, but a symbol sSymb is extensively documented in the notebook

```
Snapdragon/Documentation/English/ReferencePages/Symbols/sSymb.nb
```

which complies with the *Mathematica* standard application directory tree structure.

The text in Bibliography, Built-in symbols, guides and tutorials and Notes is, where appropriate, provided with hyperlinks referring both to web (for those not equipped with *Mathematica*) and to *Wolfram Language* & System Documentation Center (the former being marked by square brackets). In the PDF export of this notebook, only external hyperlinks work; sadly, the internal ones are defunct.

It is strongly recommended to begin evaluating the code in this manual with a fresh session of *Mathematica*, to avoid confusions caused by variables, functions and settings surviving from the previous sessions. Alternatively, one may restart the *Mathematica* kernel(s).

Error handling

All symbols provided by Snapdragon are equipped with syntax coloring and other advisories when sSymb[…] is entered as input. This is achieved using the SyntaxInformation system function. For example, sGrandiSequence accepts exactly one argument, therefore this feature ensures red marking in case of lack of arguments or too many arguments:

```
In[8]:= sGrandiSequence[ ]
        Snapdragon: sGrandiSequence[ ] does not match the rule base.
Out[8]= $Failed

In[9]:= sGrandiSequence[20, 2]
        Snapdragon: sGrandiSequence[20, 2] does not match the rule base.
Out[9]= $Failed
```

After evaluating the above examples, two features can be observed:

1. An application general error message "Snapdragon: sSymb[...] does not match the rule base." is issued upon an ill-formed function call.
2. Execution of the function code is terminated, and
3. a special symbol $Failed is returned.

The first feature is ubiquitous in Snapdragon. Actually, each function defined in Snapdragon is bulletproofed by having at least two definitions (in the sense of pattern matching in *Wolfram Language*). All definitions except the last one are devoted to particular actions performed by the function. As opposed, the last definition is programmed as a general catchall rule to catch all wrong arguments: if the user's call is not matched by any of the previous definitions, the last one comes into action since its arguments pattern is *a*___ (triple underscore, BlankNullSequence), therefore, it matches any argument sequence. This kind of error handler is very effective and elegant to program, but provides little information about the detailed cause of the error—it just tells the user that what he/she wrote does not match the rule base saved in DownValues of the function called.

There are, therefore, exceptions from this rule; some functions are equipped with one or more specific error handler(s) showing the origin of what went wrong. All messages of this kind described are prefixed with the corresponding symbol name. This behavior is typical for function options, and generally for older functions—programming this usually requires polluting the code with If's, Which's or Switch's. For example

```
In[10]:= sUniqueName[37]
         sUniqueName: base 37 is not an integer between 2 and 36, or 64.
Out[10]= $Failed

In[11]:= sParseIdNumber["690620/5273", "Country" → "PL"]
         sParseIdNumber: country "PL" is not implemented.
Out[11]= $Failed
```

There is another kind of messages that are prefixed—in addition to the symbol name—with uppercase "NOTE." They however do not terminate execution of the code (the second feature) and instead of stopping they permit the execution to be continued and the output normally returned. The purpose of these messages is informative only; they do not announce an error, but

potential problem that could, under specific circumstances, arise from using such function calls. For example,

```
In[12]:= sUniqueName[64]

         sUniqueName: NOTE: Windows–x86–64 is a case–insensitive system.

Out[12]= BGLD6GR
```

By the way, this example demonstrates a very unlikely but realistic possibility of generating two strings differing only in letter case, which on case-insensitive systems can potentially lead to problems.

The last feature also has exceptions: certain functions return Null or False when incorrectly called (for good reasons). In such cases the return value is mentioned within the corresponding "Details" subsection:

```
In[13]:= sBoole[^] // InputForm

Out[13]//InputForm=
         Null


In[14]:= sEmptyListsQ[{}, 1]

Out[14]= False
```

Some functions have the "Debug" option (with accepted values True and False), it can even be the only option a function has. This option was meant for testing purposes only, but if present, it is displayed at the beginning of the section containing function description, as an output of Options[sSymb]. Except for very rare occurrence, the "Debug" is not explained nor used during the function description. For example:

```
In[15]:= sListDecimate[{1, 2, 3, 4}, {}, "Debug" → True]

         DEBUG: list = {1, 2, 3, 4}
         nSave = 1, nKill = 1, start = 1
         opts = {{}, Debug → True}

         Snapdragon: sListDecimate[{1, 2, 3, 4}, 1, 1, 1, {}, Debug → True] does not match the rule base.

Out[15]= $Failed


In[16]:= sReferTo[List, "Debug" → True]

         DEBUG: total options: {Debug → True}

         DEBUG: Hyperlink options: {}

Out[16]= List
```

In some cases, Snapdragon symbols allow passing options pertinent to system functions called within them. For example, the function sReferTo has three own options, "Website", "Active", and "Context" (apart from the "Debug" option):

```
In[17]:= Options[sReferTo] // InputForm

Out[17]//InputForm=
{"Website" -> False, "Active" -> True, "Context" -> "System", "Debug" -> False}
```

It can, however, accept the ActiveStyle option to be passed down to the system function Hyperlink. In cases like this, syntax coloring will mark the option in red regardless of is perfect functionality:

```
In[18]:= sReferTo[List, ActiveStyle → Magenta]

Out[18]= List
```

Another example of this behavior is

```
In[19]:= sFactorExactNumber[17, GaussianIntegers → True]

Out[19]= {-ⅈ, 1 + 4 ⅈ, 4 + ⅈ}
```

In the function descriptions, no examples of error handling are presented; there are, of course, some isolated exceptions, mainly to elucidate some properties with educational aims in mind.

Batch evaluation

There are symbols that, when having been evaluated, invoke user interaction, or create new files and subdirectories in this notebook directory. This applies for sFileSelect (Section 3.1), sDirectoryDigest (Section 3.2), and sBulkExport (Section 3.3). This feature, no matter how otherwise useful, is clumsy and unconvenient for batch evaluation of this notebook using the Evaluation ▷ Evaluate Notebook menu item since the user would be disturbed with interactive windows popped out by sFileSelect, in addition, some files and directories would be created by the remaining two symbols. The following short piece of code allows decision whether to run the interactive examples in relevant sections or not; evaluating it will assign default value False to the global symbol runX, thus making the examples codes of the abovementioned symbols inactive. If you wish to bring the examples back to life and play with them, just check the checkbox in the output (then runX will be set to True); the examples then will work as if they were not wrapped in the If conditions.

```
In[20]:= Clear[runX];
         Panel[Row[{"Run examples?", Checkbox[Dynamic[runX]], Dynamic[runX]},
           Spacer[10]], ImageSize → {208, 40}]
```

Out[21]=

Having the variable being set to True, you can return to sFileSelect (Section 3.1), sDirectoryDigest (Section 3.2), or sBulkExport (Section 3.3) for playing with examples.

1

List and array manipulations

Lists are the universal container for *Mathematica* expressions of all kinds. Data of arbitrary complexity can be represented as a certain (perhaps complex and nested) list. For example, vectors, audio signals, matrices, images, spatial matrices, 3D images, generally arrays of any depth, also trees, graphs, tensors, and many more. Although *Wolfram Language* provides a plethora of list manipulation tools, here is a modest collection of a few new list manipulation functions which proved themselves to be useful in daily work. Included are functions accepting a general list, an array, or in particular cases a general expression as the first argument. Some of them may ambiguously fall into a different category, typically those included in Chapter 2.

1.1 sListDecimate

In[22]:=
```
infoAbout@sListDecimate
```

sListDecimate[*list*, *nsave*, *nkill*, *start*] returns *list*
in "decimated" form controlled by parameters *nsave*, *nkill*, *start*.

```
Attributes[sListDecimate] = {Protected, ReadProtected}

Options[sListDecimate] = {Method → 1, Debug → False}

SyntaxInformation[sListDecimate] =
 {ArgumentsPattern → {_, _., _., _., OptionsPattern[]}}
```

Details

sListDecimate[*list*, *nsave*, *nkill*, *start*, *opts*] returns *list* in "decimated" form: contiguous sequence of the first *nsave* (default 1) elements is retained, contiguous sequence of the next *nkill* (default 1) elements is removed, then the following contiguous sequences of *nsave* and *nkill* elements are

retained and removed, respectively, and so on, with the period of *nsave* + *nkill*. The positive integer *start* (default 1) indicates the element to start decimation from.

- Violating any of the constraints imposed on the parameters,

  $n_{\text{save}} + n_{\text{kill}} \le \text{Length}[list]$

  $1 \le n_{\text{save}} \le \text{Length}[list]$

  $0 \le n_{\text{kill}} \le \text{Length}[list] - 1$

  $1 \le start \le n_{\text{kill}} + 1$

  causes $Failed to be returned.
- *list* must be a full array of any depth, but only the top (outmost) level is decimated; deeper levels can be decimated using the Map or MapIndexed system function.
- Six methods are currently available via option Method: Method → 1 (based on the Pick built-in symbol, default), 2 (based on the Partition built-in symbol), 3 (based on the Downsample built-in symbol), 4 (based on the MapIndexed built-in symbol), 5 (based on the ReplacePart built-in symbol), 6 (based on the FixedPoint built-in symbol). See Discussion below for their respective performance comparison; the default method 1 appears to be the golden rule, being reasonably fast in most circumstances.

sListDecimate may be thought of as a generalization of the Downsample built-in symbol.

Examples

Clearing global symbols:

```
In[23]:= Clear[vec, mat, m3, a, aud, adat, apar, img, ilg, idat, idesc, res, im3,
           idt3, t, nm, v, tm, st, nt];
         ClearAll[coefOfVariance, comparePerformance];
```

Decimating 1D arrays (vectors)

First we generate a test vector:

```
In[25]:= vec = Array["v"<>FromDigits[{##}] &, {18}];
```

The default behavior is killing every second element, which correspnds to all n_{save}, n_{kill} and *start* equal to 1:

```
In[26]:= {#, sListDecimate[#]} &@vec // Column
```

```
Out[26]= {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18}
         {v1, v3, v5, v7, v9, v11, v13, v15, v17}
```

In the following 1D array decimation examples, both original and decimated vectors will be displayed one above other. Starting from the first element (default), retain 4 elements, remove next 6 elements, and so on:

```
In[27]:= {#, sListDecimate[#, 4, 6]} &@vec // Column
```

```
Out[27]= {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18}
         {v1, v2, v3, v4, v11, v12, v13, v14}
```

The same as above, but starting from the seventh element:

```
In[28]:= {#, sListDecimate[#, 4, 6, 7]} &@vec // Column
```

```
Out[28]= {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18}
         {v7, v8, v9, v10, v17, v18}
```

Giving any allowed n_{Save}, zero n_{Kill} and default *start* leads to identity:

```
In[29]:= {#, sListDecimate[#, 4, 0]} &@vec // Column
```

```
Out[29]= {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18}
         {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18}
```

```
In[30]:= {#, sListDecimate[#, Length@#, 0]} &@vec // Column
```

```
Out[30]= {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18}
         {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18}
```

Decimating 2D arrays (matrices)

We generate a test matrix:

```
In[31]:= mat = Array["a"FromDigits[{##}] &, {8, 5}];
```

In the following 2D array decimation examples, both original and decimated matrices will be displayed side by side. The first bunch of examples: decimate matrix rows only, columns only, and both, respectively:

```
In[32]:= Row[MatrixForm /@ {#, sListDecimate[#, 1, 2]}] &@mat
```

```
Out[32]=
⎛a11 a12 a13 a14 a15⎞ ⎛a11 a12 a13 a14 a15⎞
⎜a21 a22 a23 a24 a25⎟ ⎜a41 a42 a43 a44 a45⎟
⎜a31 a32 a33 a34 a35⎟ ⎝a71 a72 a73 a74 a75⎠
⎜a41 a42 a43 a44 a45⎟
⎜a51 a52 a53 a54 a55⎟
⎜a61 a62 a63 a64 a65⎟
⎜a71 a72 a73 a74 a75⎟
⎝a81 a82 a83 a84 a85⎠
```

```
In[33]:= Row[MatrixForm /@ {#, sListDecimate[#, 1, 2] & /@ #}] &@mat
```

```
Out[33]=
⎛a11 a12 a13 a14 a15⎞ ⎛a11 a14⎞
⎜a21 a22 a23 a24 a25⎟ ⎜a21 a24⎟
⎜a31 a32 a33 a34 a35⎟ ⎜a31 a34⎟
⎜a41 a42 a43 a44 a45⎟ ⎜a41 a44⎟
⎜a51 a52 a53 a54 a55⎟ ⎜a51 a54⎟
⎜a61 a62 a63 a64 a65⎟ ⎜a61 a64⎟
⎜a71 a72 a73 a74 a75⎟ ⎜a71 a74⎟
⎝a81 a82 a83 a84 a85⎠ ⎝a81 a84⎠
```

In[34]:=
```
Row[MatrixForm /@ {#, sListDecimate[#, 1, 2] & /@ sListDecimate[#, 1, 2]}] &@
 mat
```

Out[34]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} a_{11} & a_{14} \\ a_{41} & a_{44} \\ a_{71} & a_{74} \end{pmatrix}$$

The last example above which has the same decimating specification {1, 2} in both levels can be accomplished using the "Fold idiom":

In[35]:=
```
Row[
   MatrixForm /@
    {#, Fold[Map[sListDecimate[#, 1, 2] &, #1, {#2}] &, #,
      Range[0, ArrayDepth@# - 1]]}] &@mat
```

Out[35]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} a_{11} & a_{14} \\ a_{41} & a_{44} \\ a_{71} & a_{74} \end{pmatrix}$$

With a little more amount of coding, the "Fold idiom" can be used even if the decimation control is required for each level separately (see the very end of the code for the decimation specification):

In[36]:=
```
Row[MatrixForm /@ {#,
      Fold[Map[With[{a = #2[[2]]}, sListDecimate[#, Sequence @@ a] &], #1,
        {#2[[1]]}] &, #,
       {Range[0, ArrayDepth@# - 1], {{1, 2}(*rows*), {2, 1, 2}
         (*columns*)}}^T]}] &@mat
```

Out[36]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} a_{12} & a_{13} & a_{15} \\ a_{42} & a_{43} & a_{45} \\ a_{72} & a_{73} & a_{75} \end{pmatrix}$$

Decimating 3D arrays (spatial matrices)

We generate a test 3D (spatial) matrix:

In[37]:=
```
m3 = Array["a"<>FromDigits[{##}] &, {4, 4, 3}];
```

In the following 3D array decimation examples, both original and decimated spatial matrices will

be displayed side by side. The first bunch of examples: decimate 3D matrix rows only, columns only, depth only:

```
In[38]:= MatrixForm /@ {#, sListDecimate[#]} &@m3 // Row
```

Out[38]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \end{pmatrix} \\ \begin{pmatrix} a_{411} \\ a_{412} \\ a_{413} \end{pmatrix} & \begin{pmatrix} a_{421} \\ a_{422} \\ a_{423} \end{pmatrix} & \begin{pmatrix} a_{431} \\ a_{432} \\ a_{433} \end{pmatrix} & \begin{pmatrix} a_{441} \\ a_{442} \\ a_{443} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \end{pmatrix} \end{pmatrix}$$

```
In[39]:= MatrixForm /@ {#, sListDecimate[#] & /@ #} &@m3 // Row
```

Out[39]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \end{pmatrix} \\ \begin{pmatrix} a_{411} \\ a_{412} \\ a_{413} \end{pmatrix} & \begin{pmatrix} a_{421} \\ a_{422} \\ a_{423} \end{pmatrix} & \begin{pmatrix} a_{431} \\ a_{432} \\ a_{433} \end{pmatrix} & \begin{pmatrix} a_{441} \\ a_{442} \\ a_{443} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \end{pmatrix} \\ \begin{pmatrix} a_{411} \\ a_{412} \\ a_{413} \end{pmatrix} & \begin{pmatrix} a_{431} \\ a_{432} \\ a_{433} \end{pmatrix} \end{pmatrix}$$

```
In[40]:= MatrixForm /@ {#, Map[sListDecimate[#] &, #, {2}]} &@m3 // Row
```

Out[40]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \end{pmatrix} \\ \begin{pmatrix} a_{411} \\ a_{412} \\ a_{413} \end{pmatrix} & \begin{pmatrix} a_{421} \\ a_{422} \\ a_{423} \end{pmatrix} & \begin{pmatrix} a_{431} \\ a_{432} \\ a_{433} \end{pmatrix} & \begin{pmatrix} a_{441} \\ a_{442} \\ a_{443} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{113} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{123} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{133} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{143} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{213} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{223} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{233} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{243} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{313} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{323} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{333} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{343} \end{pmatrix} \\ \begin{pmatrix} a_{411} \\ a_{413} \end{pmatrix} & \begin{pmatrix} a_{421} \\ a_{423} \end{pmatrix} & \begin{pmatrix} a_{431} \\ a_{433} \end{pmatrix} & \begin{pmatrix} a_{441} \\ a_{443} \end{pmatrix} \end{pmatrix}$$

The "Fold idiom" can be used even if the decimation control is required for each level separately (see the very end of the code):

In[41]:=
```
MatrixForm /@
    {#, Fold[Map[With[{a = #2[[2]]}, sListDecimate[#, Sequence @@ a] &],
        #1, {#2[[1]]}] &, #,
      {Range[0, ArrayDepth@# - 1], {{1, 1, 2}(*rows*), {1, 1, 1}
        (*columns*), {2, 1, 1}(*depth*)}}^T]} &@m3 // Row
```

Out[41]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \end{pmatrix} \\ \begin{pmatrix} a_{411} \\ a_{412} \\ a_{413} \end{pmatrix} & \begin{pmatrix} a_{421} \\ a_{422} \\ a_{423} \end{pmatrix} & \begin{pmatrix} a_{431} \\ a_{432} \\ a_{433} \end{pmatrix} & \begin{pmatrix} a_{441} \\ a_{442} \\ a_{443} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{211} \\ a_{212} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \end{pmatrix} \\ \begin{pmatrix} a_{411} \\ a_{412} \end{pmatrix} & \begin{pmatrix} a_{431} \\ a_{432} \end{pmatrix} \end{pmatrix}$$

Applications

Audio aliasing demonstration

First we generate a test audio (the data of which is actually a 1D array):

In[42]:=
```
aud = AudioGenerator[{"Sin", 200 π # &}, 8, SampleRate → 16000]
adat = AudioData[aud];
apar = Through[{AudioSampleRate, AudioType, AudioChannels, Duration}[aud]]
```

Out[42]=

Out[44]= $\{16000\,\text{Hz}, \text{Real32}, 1, 8.\,\text{s}\}$

Now take every 8th sample, effectively downsampling the audio signal by factor of 8 (as if the original continuous audiosignal had been sampled at 2000 Hz instead of 16000 Hz). This is an example of aliasing (listen to the resulting audio and compare with the original one):

In[45]:=
```
With[{nSave = 1, nKill = 7},
 Audio[sListDecimate[#, nSave, nKill] & /@ adat,
  SampleRate → Round[apar[[1, 1]]/(nSave + nKill)]]]
Through[{AudioSampleRate, AudioType, AudioChannels, Duration}[%]]
```

Out[45]=

Out[46]=
```
{2000 Hz, Real32, 1, 8. s}
```

Image aliasing demonstration

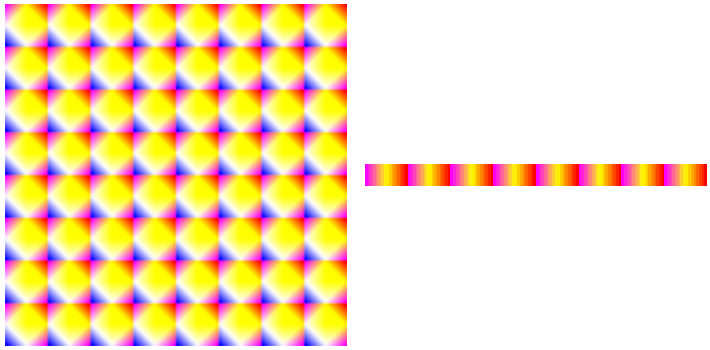
We generate a test image (which is actually a 2D array):

In[47]:=
```
img =
  ImageAdd[
   ilg = ImageAssemble@
     ConstantArray[LinearGradientImage[{Blue, Yellow, Red}, {16, 128}], 8],
   ImageRotate@ilg];
idat = ImageData@img;
idesc = {(*ImageType,*)ImageDimensions(*,ImageChannels*)};
```

In the following image decimation examples, both original and decimated images will be displayed side by side. Decimate image rows only, columns only, and both, respectively. First we take every 16th row, starting from the first one:

In[49]:=
```
res = {img, Image[sListDecimate[idat, 1, 15]]};
Grid@{res, Through[idesc[#]] & /@ res}
```

Out[50]=

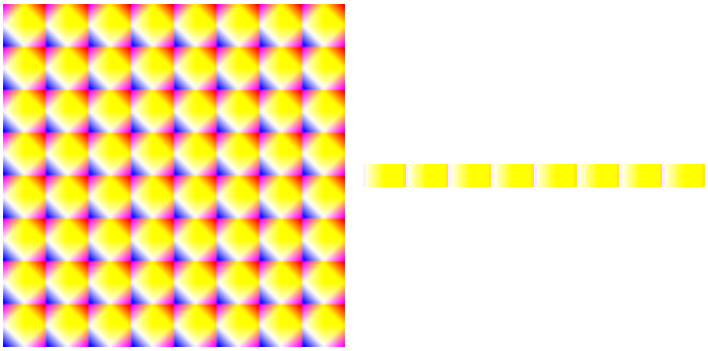


{{128, 128}} {{128, 8}}

Now the same, but starting from the 8th one:

```
In[51]:= res = {img, Image[sListDecimate[idat, 1, 15, 8]]};
         Grid@{res, Through[idesc[#]] & /@ res}
```

Out[52]=

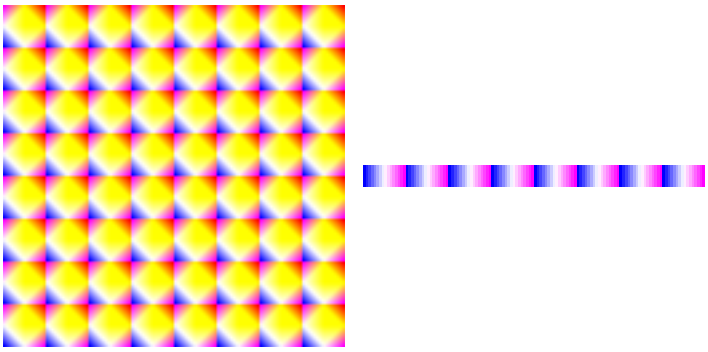


{{128, 128}} {{128, 8}}

Starting decimation from the 16th row leads to even more different aliasing:

```
In[53]:= res = {img, Image[sListDecimate[idat, 1, 15, 16]]};
         Grid@{res, Through[idesc[#]] & /@ res}
```

Out[54]=

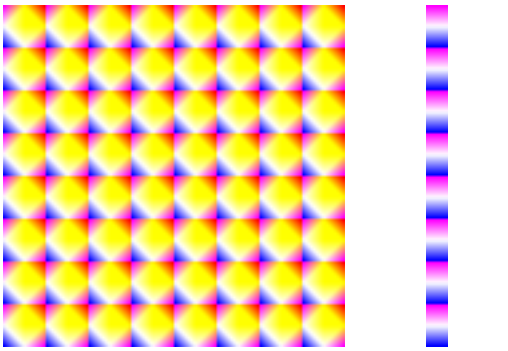


{{128, 128}} {{128, 8}}

Here we take every 16th column, starting from the first one:

```
In[55]:= res = {img, Image[sListDecimate[#, 1, 15] & /@ idat]};
         Grid@{res, Through[idesc[#]] & /@ res}
```

Out[56]=

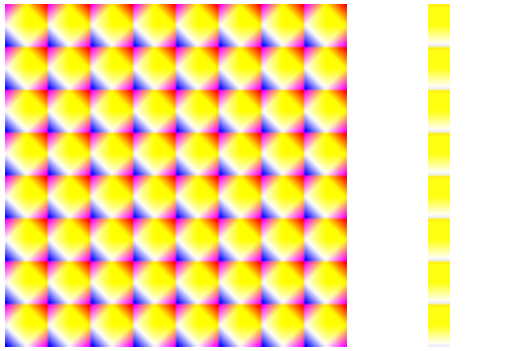


{{128, 128}} {{8, 128}}

Now the same as in the previous example, but starting from the 8th column:

In[57]:=
```
res = {img, Image[sListDecimate[#, 1, 15, 8] & /@ idat]};
Grid@{res, Through[idesc[#]] & /@ res}
```

Out[58]=

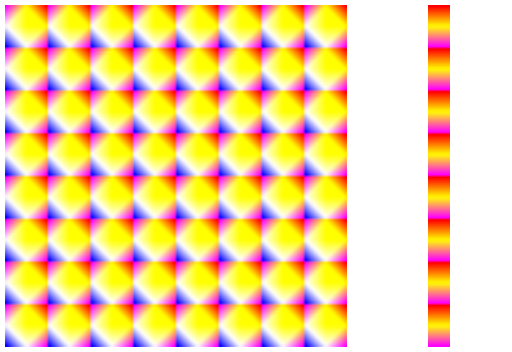


{{128, 128}} {{8, 128}}

Starting decimation from the 16th column leads to even more different aliasing:

In[59]:=
```
res = {img, Image[sListDecimate[#, 1, 15, 16] & /@ idat]};
Grid@{res, Through[idesc[#]] & /@ res}
```

Out[60]=

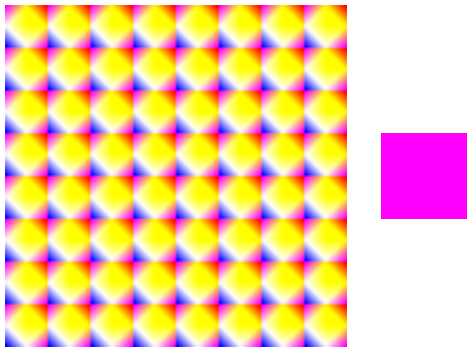


{{128, 128}} {{8, 128}}

Decimating by taking every 16th row and every 16th column (starting from the first ones) gives completely aliased image:

In[61]:=
```
res =
  {img, With[{p = Sequence[1, 15]},
    Image[sListDecimate[#, p] & /@ sListDecimate[idat, p]]]};
Grid@{res, Through[idesc[#]] & /@ res}
```

Out[62]=

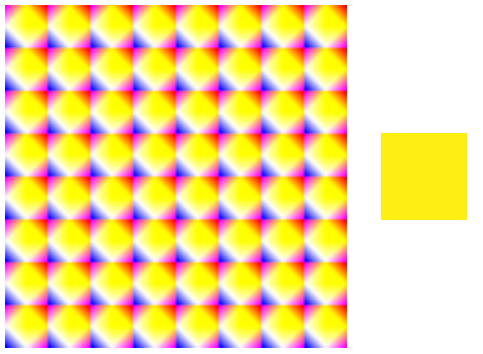


{{128, 128}} {{8, 8}}

Now the same as in the previous example, but starting from the 1st row and the 8th column:

```
In[63]:= res =
           {img, With[{p = Sequence[1, 15]},
             Image[sListDecimate[#, p, 8] & /@ sListDecimate[idat, p]]]};
         Grid@{res, Through[idesc[#]] & /@ res}
```

Out[64]=

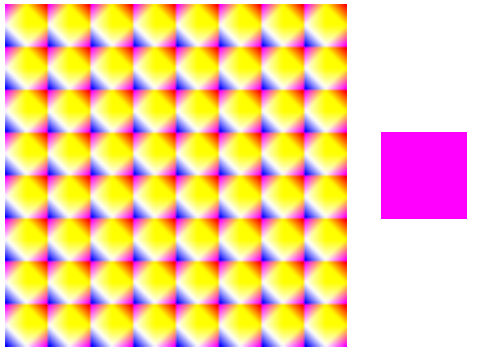


{{128, 128}} {{8, 8}}

The last example which has the same decimating specification {1, 15} in both levels can be accomplished using the "Fold idiom":

```
In[65]:= res = {img,
             Image[Fold[Map[sListDecimate[#, 1, 15] &, #1, {#2}] &, #,
               Range[0, ArrayDepth@# - 2]]] &@idat};
         Grid@{res, Through[idesc[#]] & /@ res}
```

Out[66]=

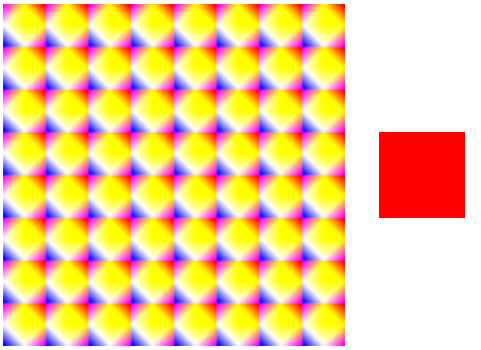


{{128, 128}} {{8, 8}}

With a little more amount of coding, the "Fold idiom" can be used even if the decimation control is required for each level separately (see the very end of the code):

In[67]:=
```
res =
  {img,
   Image[Fold[Map[With[{a = #2[[2]]}, sListDecimate[#, Sequence @@ a] &],
         #1, {#2[[1]]}] &, #,
       {Range[0, ArrayDepth@# - 2], {{1, 15}(*rows*), {1, 15, 16}
         (*columns*)}}^T]] &@idat};
Grid@{res, Through[idesc[#]] & /@ res}
```

Out[68]=



{{128, 128}} {{8, 8}}

3D image aliasing demonstration

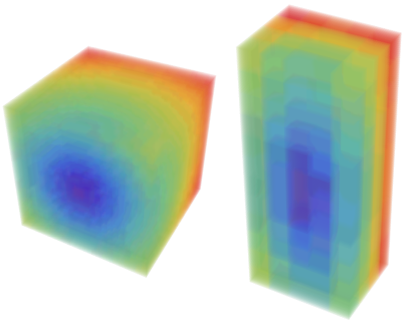
We generate a test image (which is actually a 2D array):

In[69]:=
```
im3 = RadialGradientImage[{Front, {Right, Back, Top}} → ColorData["Rainbow"],
    {24, 24, 24}];
idt3 = ImageData[im3];
idesc = {(*ImageType,*)ImageDimensions(*,ImageChannels*)};
```

In the following 3D image decimation example, both original and decimated spatial image will be displayed side by side. Decimate 3D image in all dimensions using the "Fold idiom":

```
In[71]:= res =
           {im3,
            Image3D[
              Fold[Map[With[{a = #2[[2]]}, sListDecimate[#, Sequence @@ a] &],
                  #1, {#2[[1]]}] &, #,
                {Range[0, ArrayDepth@# - 2], {{1, 1, 1}, {1, 6, 1}, {1, 4, 1}}}^T]] &@
             idt3};
         Grid[{res, Through[idesc[#]] & /@ res}]
```

Out[72]=



```
{{24, 24, 24}}   {{5, 4, 12}}
```

Discussion

Methods output comparison

First we generate a test vector. A total of six methods are available:

```
In[73]:= Clear[vec];
         vec = Array["v"_FromDigits[{##}] &, {16}];
         nm = 6 (* # of methods *);
```

Let us decimate the test vector using all six methods, and test the results for sameness (True is the desired result):

```
In[74]:= Column[Column /@ (Map[MatrixForm[#, TableDirections → Row] &, t = Table[{#,
                 sListDecimate[#, 1, 2, 1, Method → m]}, {m, nm}] &@vec, {2}])]
         SameQ @@ t[[All, 2]]
```

```
Out[74]=
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10  v11  v12  v13  v14  v15  v16 )
( v1  v4  v7  v10  v13  v16 )
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10  v11  v12  v13  v14  v15  v16 )
( v1  v4  v7  v10  v13  v16 )
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10  v11  v12  v13  v14  v15  v16 )
( v1  v4  v7  v10  v13  v16 )
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10  v11  v12  v13  v14  v15  v16 )
( v1  v4  v7  v10  v13  v16 )
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10  v11  v12  v13  v14  v15  v16 )
( v1  v4  v7  v10  v13  v16 )
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10  v11  v12  v13  v14  v15  v16 )
( v1  v4  v7  v10  v13  v16 )

Out[75]= True
```

Now we try a test matrix:

```
In[76]:= Clear[mat]; mat = Array["a"<>FromDigits[{##}] &, {8, 5}];
```

Again, we decimate the test vector using all six methods, and test the results for sameness with various combinations of decimation parameters (with True being the desired result, and the output suppressed to save space):

```
In[77]:= Column@(Map[MatrixForm, t = Table[{#,
           sListDecimate[#, 2, Method → m]}, {m, nm}] &@@mat, {2}]);
         SameQ@@t[[All, 2]]
Out[78]= True

In[79]:= Column@(Map[MatrixForm, t = Table[{#,
           sListDecimate[#, 1, 1, 2, Method → m] & /@ #}, {m, nm}] &@@mat, {2}]);
         SameQ@@t[[All, 2]]
Out[80]= True

In[81]:= Column@(Map[MatrixForm, t = Table[{#,
           sListDecimate[#, 2, 2, 3, Method → m] & /@ sListDecimate[#, 2, 2]}, {m, r
       mat, {2}]);
         SameQ@@t[[All, 2]]
Out[82]= True
```

We got True in all cases. Finally, we generate a test 3D (spatial) matrix, and test the output of all methods for sameness:

```
In[83]:= Clear[m3]; m3 = Array["a"<>FromDigits[{##}] &, {5, 6, 4}];

In[84]:= Column@(Map[MatrixForm, t = Table[{#,
           Fold[Map[With[{a = #2[[2]]}, sListDecimate[#, Sequence@@a, Method → m]
              #1, {#2[[1]]}] &, #,
            {Range[0, ArrayDepth@# - 1], {{1, 1, 2}, {2, 1, 1}, {1, 3, 1}}}^T]
          }, {m, nm}] &@@m3, {2}]);
         SameQ@@t[[All, 2]]
Out[85]= True
```

And again, application of all methods leads to same results.

Performance comparison of methods

Although all six methods are equivalent in terms of outcome, they may differ in execution times, and indeed they do. We generate a long vector and define a set of basic statistics (mean, median and the coefficient of variance defined as the proportion of standard deviation to the mean, here given in percents) to compare performances of individual methods:

In[86]:=
```
v = RandomReal[{-1, 1}, 10^5];
nm = 6(* # of methods *);
nt = 10(* nt samples for stats *);
coefOfVariance[list_] :=
  Module[{m = Mean[list]}, If[m == 0, ∞, 100.0 StandardDeviation[list] / m]];
st = {Mean, Median, coefOfVariance}; tm = Timing;
comparePerformance[nSave_, nKill_, start_] :=
 (TableForm[
   SortBy[#[[2]] &][
    Table[
     Join[{m},
      Through[
       st[Table[tm[sListDecimate[v, nSave, nKill, start, Method → m];][[1]],
         nt]]]], {m, nm}]],
   TableHeadings → {Range[nm], Join[{Method}, st]}])
```

In[90]:=
```
comparePerformance[1, 1, 1]
```

Out[90]//TableForm=

	Method	Mean	Median	coefOfVariance
1	1	0.0124801	0.0156001	52.7046
2	2	0.0202801	0.0156001	37.1574
3	3	0.129481	0.124801	5.81983
4	4	0.198121	0.202801	7.46995
5	5	0.238682	0.234002	3.15716
6	6	1.99681	1.99681	0.520833

In[91]:=
```
comparePerformance[3, 2, 1]
```

Out[91]//TableForm=

	Method	Mean	Median	coefOfVariance
1	1	0.0124801	0.0156001	52.7046
2	2	0.0124801	0.0156001	52.7046
3	3	0.151321	0.156001	4.97985
4	4	0.209041	0.210601	5.21795
5	5	0.230881	0.234002	2.8489
6	6	0.864246	0.858006	0.932126

In[92]:=
```
comparePerformance[9, 1, 1]
```

Out[92]//TableForm=

	Method	Mean	Median	coefOfVariance
1	1	0.0109201	0.0156001	69.0066
2	2	0.0109201	0.0156001	69.0066
3	4	0.209041	0.202801	3.85371
4	5	0.226201	0.226201	3.6348
5	3	0.232441	0.234002	2.12233
6	6	0.489843	0.483603	1.64458

In[93]:= **comparePerformance[1, 9, 1]**

Out[93]//TableForm=

	Method	Mean	Median	coefOfVariance
1	2	0.00936006	0.0156001	86.0663
2	1	0.0109201	0.0156001	69.0066
3	3	0.0280802	0.0312002	23.4243
4	4	0.199681	0.202801	3.29404
5	5	0.234002	0.234002	0.
6	6	0.301082	0.296402	2.50283

In[94]:= **comparePerformance[5, 5, 1]**

Out[94]//TableForm=

	Method	Mean	Median	coefOfVariance
1	2	0.00936006	0.0156001	86.0663
2	1	0.0124801	0.0156001	52.7046
3	3	0.126361	0.124801	3.90405
4	4	0.205921	0.202801	4.79133
5	5	0.229321	0.234002	3.28603
6	6	0.394683	0.390003	1.90927

In[95]:= **comparePerformance[98, 2, 1]**

Out[95]//TableForm=

	Method	Mean	Median	coefOfVariance
1	2	0.00780005	0.00780005	105.409
2	1	0.0109201	0.0156001	69.0066
3	6	0.0514803	0.0468003	14.6378
4	4	0.212161	0.218401	5.14122
5	5	0.212161	0.218401	3.79704
6	3	0.266762	0.265202	3.31957

In[96]:= **comparePerformance[2, 98, 1]**

Out[96]//TableForm=

	Method	Mean	Median	coefOfVariance
1	3	0.00468003	0.	161.015
2	2	0.00780005	0.00780005	105.409
3	1	0.00936006	0.0156001	86.0663
4	6	0.0280802	0.0312002	23.4243
5	4	0.196561	0.195001	5.54925
6	5	0.230881	0.234002	2.8489

In[97]:= **comparePerformance[50, 50, 1]**

Out[97]//TableForm=

	Method	Mean	Median	coefOfVariance
1	2	0.00780005	0.00780005	105.409
2	1	0.0109201	0.0156001	69.0066
3	6	0.0405603	0.0468003	19.8615
4	3	0.129481	0.124801	5.81983
5	4	0.204361	0.202801	2.41395
6	5	0.223081	0.218401	3.37794

In[98]:=
```
comparePerformance[997, 3, 1]
```

Out[98]//TableForm=

	Method	Mean	Median	coefOfVariance
1	6	0.00624004	0.	129.099
2	2	0.00936006	0.0156001	86.0663
3	1	0.0109201	0.0156001	69.0066
4	4	0.209041	0.202801	3.85371
5	5	0.213721	0.218401	3.52588
6	3	0.397803	0.397803	2.06685

In[99]:=
```
comparePerformance[3, 997, 1]
```

Out[99]//TableForm=

	Method	Mean	Median	coefOfVariance
1	3	0.00156001	0.	316.228
2	6	0.00312002	0.	210.819
3	2	0.00780005	0.00780005	105.409
4	1	0.00936006	0.0156001	86.0663
5	4	0.195001	0.195001	4.21637
6	5	0.229321	0.234002	3.28603

In[100]:=
```
comparePerformance[500, 500, 1]
```

Out[100]//TableForm=

	Method	Mean	Median	coefOfVariance
1	6	0.00468003	0.	161.015
2	2	0.00780005	0.00780005	105.409
3	1	0.0109201	0.0156001	69.0066
4	3	0.196561	0.202801	4.0984
5	4	0.201241	0.202801	4.40036
6	5	0.227761	0.234002	3.53697

We thus may conclude that individual methods can significantly differ for various decimation specifications. The default method 1 however really appears to be the golden rule, being reasonably fast in most circumstances.

1.2 sListPush

In[101]:=
```
infoAbout@sListPush
```

sListPush[*u*, *n*] shifts elements in list *u* by |*n*| places,
padding vacant places with 0's, discarding opposite protruding ones.

```
Attributes[sListPush] = {Protected, ReadProtected}
```

```
SyntaxInformation[sListPush] = {ArgumentsPattern → {_, _., _.}}
```

Details

sListPush[*u*, *n*, *pad*] shifts the elements in the list *u* to the right (left) by | *n* | places (1 by default) if *n* is a nonnegative (negative) integer, padding vacant places with *pad*'s (default 0's) at one end

and discarding opposite protruding elements.

- List *u* must be a full array of any depth, but only the top (outmost) level is affected; use Map or MapIndexed for deeper levels.
- Padding specifications *pad* coincides with those of PadLeft or PadRight.

Examples

Clearing global symbols:

```
In[102]:= Clear[vec, mat, m3, r, c, d, aud, adat, apar, img, idat, idesc, res, im3, idt3];
```

Pushing 1D arrays (vectors)

We generate a test vector:

```
In[103]:= vec = Array["v"<>FromDigits[{##}] &, {18}];
```

Shift elements in the list to the right by one place (default), padding with 0:

```
In[104]:= MatrixForm[{#, sListPush[#]} &@vec]
```

```
Out[104]//MatrixForm=
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10 v11 v12 v13 v14 v15 v16 v17 v18 )
(  0  v1  v2  v3  v4  v5  v6  v7  v8   v9  v10 v11 v12 v13 v14 v15 v16 v17 )
```

Specifying zero shift leads to identity:

```
In[105]:= MatrixForm[{#, sListPush[#, 0]} &@vec]
```

```
Out[105]//MatrixForm=
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10 v11 v12 v13 v14 v15 v16 v17 v18 )
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10 v11 v12 v13 v14 v15 v16 v17 v18 )
```

Give an explicite element to pad vacancy with:

```
In[106]:= MatrixForm[{#, sListPush[#, 1, "x"]} &@vec]
```

```
Out[106]//MatrixForm=
( v1  v2  v3  v4  v5  v6  v7  v8  v9  v10 v11 v12 v13 v14 v15 v16 v17 v18 )
(  x  v1  v2  v3  v4  v5  v6  v7  v8   v9  v10 v11 v12 v13 v14 v15 v16 v17 )
```

Shift the elements in the list to the left by one place:

```
In[107]:= MatrixForm[{#, sListPush[#, -1]} &@vec]
```

```
Out[107]//MatrixForm=
( v1  v2  v3  v4  v5  v6  v7  v8   v9  v10 v11 v12 v13 v14 v15 v16 v17 v18 )
( v2  v3  v4  v5  v6  v7  v8  v9  v10 v11 v12 v13 v14 v15 v16 v17 v18  0  )
```

Give an explicite element to pad vacancy with:

```
In[108]:= MatrixForm[{#, sListPush[#, -1, "x"]} &@vec]
```

Out[108]//MatrixForm=

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} \\ v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} & x \end{pmatrix}$$

Specify your custom shifts:

```
In[109]:= MatrixForm[{#, sListPush[#, 4]} &@vec]
```

Out[109]//MatrixForm=

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} \\ 0 & 0 & 0 & 0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} \end{pmatrix}$$

```
In[110]:= MatrixForm[{#, sListPush[#, -2, "x"]} &@vec]
```

Out[110]//MatrixForm=

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} \\ v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} & x & x \end{pmatrix}$$

If the absolute value of the shift equals or exceeds the length of the list, only pads will fill the output:

```
In[111]:= MatrixForm[{#, sListPush[#, 20]} &@vec]
```

Out[111]//MatrixForm=

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

```
In[112]:= MatrixForm[{#, sListPush[#, -25, "x"]} &@vec]
```

Out[112]//MatrixForm=

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} \\ x & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x & x \end{pmatrix}$$

Empty list in place of optional pad leads to discarding elements:

```
In[113]:= MatrixForm[{#, sListPush[#, 4, {}]} &@vec]
```

Out[113]//MatrixForm=

$$\begin{pmatrix} \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}\} \\ \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}\} \end{pmatrix}$$

This behavior is due to padding specification is inherited from built-in symbols PadRight, PadLeft:

```
In[114]:= {#, sListPush[#, 4, {"x"}]} &@vec === {#, sListPush[#, 4, "x"]} &@
          vec

Out[114]= True
```

Finally, observe carefully how the padding is specified in PadRight, PadLeft:

In[115]:=
```
MatrixForm[{#, sListPush[#, 5, {"x", "y"}]} &@vec]
```

Out[115]//MatrixForm=

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} \\ x & y & x & y & x & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} \end{pmatrix}$$

In[116]:=
```
MatrixForm[{#, sListPush[#, -5, {"x", "y"}]} &@vec]
```

Out[116]//MatrixForm=

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} \\ v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} & y & x & y & x & y \end{pmatrix}$$

In[117]:=
```
MatrixForm[{#, sListPush[#, 7, {"w", "x", "y", "z"}]} &@vec]
```

Out[117]//MatrixForm=

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} \\ y & z & w & x & y & z & w & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} \end{pmatrix}$$

In[118]:=
```
MatrixForm[{#, sListPush[#, -7, {"w", "x", "y", "z"}]} &@vec]
```

Out[118]//MatrixForm=

$$\begin{pmatrix} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} \\ v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} & v_{15} & v_{16} & v_{17} & v_{18} & z & w & x & y & z & w & x \end{pmatrix}$$

Pushing 2D arrays (matrices)

We generate a test matrix:

In[119]:=
```
mat = Array["a"FromDigits[{##}] &, {8, 5}];
```

In the following 2D array push examples, both original and pushed matrices will be displayed side by side. Here we push matrix rows only:

In[120]:=
```
Row@
 Map[MatrixForm,
  {#, sListPush[#, 4, List@ConstantArray[0, Last@Dimensions@mat]]} &@mat]
```

Out[120]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{pmatrix}$$

In[121]:=
```
Row@
 Map[MatrixForm,
  {#, sListPush[#, -7, List@ConstantArray[0, Last@Dimensions@mat]]} &@@mat]
```

Out[121]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

In[122]:=
```
Row@Map[MatrixForm, {#, sListPush[#, 1, {Range[Last@Dimensions@mat]}]} &@
   mat]
```

Out[122]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \end{pmatrix}$$

In[123]:=
```
Row@Map[MatrixForm, {#, sListPush[#, 1, Range[Last@Dimensions@mat]]} &@@mat]
```

Out[123]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} 3 \\ \{a_{11}, a_{12}, a_{13}, a_{14}, a_{15}\} \\ \{a_{21}, a_{22}, a_{23}, a_{24}, a_{25}\} \\ \{a_{31}, a_{32}, a_{33}, a_{34}, a_{35}\} \\ \{a_{41}, a_{42}, a_{43}, a_{44}, a_{45}\} \\ \{a_{51}, a_{52}, a_{53}, a_{54}, a_{55}\} \\ \{a_{61}, a_{62}, a_{63}, a_{64}, a_{65}\} \\ \{a_{71}, a_{72}, a_{73}, a_{74}, a_{75}\} \end{pmatrix}$$

In[124]:=
```
With[{w = Last@Dimensions@mat},
 Row@Map[MatrixForm, {#, sListPush[#, 2, {Range[w], Range[w] + w}]} &@@mat]]
```

Out[124]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \end{pmatrix}$$

In[125]:=
```
Row@
 Map[MatrixForm,
  {#, sListPush[#, -2, Table[i^k, {i, 2}, {k, Last@Dimensions@mat}]]} &@mat]
```

Out[125]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 4 & 8 & 16 & 32 \end{pmatrix}$$

Here we push matrix columns only:

In[126]:=
```
Row@Map[MatrixForm, {#, sListPush[#] & /@ #} &@mat]
```

Out[126]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} 0 & a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{21} & a_{22} & a_{23} & a_{24} \\ 0 & a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & a_{41} & a_{42} & a_{43} & a_{44} \\ 0 & a_{51} & a_{52} & a_{53} & a_{54} \\ 0 & a_{61} & a_{62} & a_{63} & a_{64} \\ 0 & a_{71} & a_{72} & a_{73} & a_{74} \\ 0 & a_{81} & a_{82} & a_{83} & a_{84} \end{pmatrix}$$

In[127]:=
```
Row@Map[MatrixForm, {#, sListPush[#, -2, 0] & /@ #} &@mat]
```

Out[127]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} a_{13} & a_{14} & a_{15} & 0 & 0 \\ a_{23} & a_{24} & a_{25} & 0 & 0 \\ a_{33} & a_{34} & a_{35} & 0 & 0 \\ a_{43} & a_{44} & a_{45} & 0 & 0 \\ a_{53} & a_{54} & a_{55} & 0 & 0 \\ a_{63} & a_{64} & a_{65} & 0 & 0 \\ a_{73} & a_{74} & a_{75} & 0 & 0 \\ a_{83} & a_{84} & a_{85} & 0 & 0 \end{pmatrix}$$

In[128]:=
```
Row@Map[MatrixForm, {#, sListPush[#, 2, {0}] & /@ #} &@mat]
```

Out[128]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} 0 & 0 & a_{11} & a_{12} & a_{13} \\ 0 & 0 & a_{21} & a_{22} & a_{23} \\ 0 & 0 & a_{31} & a_{32} & a_{33} \\ 0 & 0 & a_{41} & a_{42} & a_{43} \\ 0 & 0 & a_{51} & a_{52} & a_{53} \\ 0 & 0 & a_{61} & a_{62} & a_{63} \\ 0 & 0 & a_{71} & a_{72} & a_{73} \\ 0 & 0 & a_{81} & a_{82} & a_{83} \end{pmatrix}$$

In[129]:=
```
Row@Map[MatrixForm, {#, sListPush[#, 2, {0, 1}] & /@ #} &@mat]
```

Out[129]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} 1 & 0 & a_{11} & a_{12} & a_{13} \\ 1 & 0 & a_{21} & a_{22} & a_{23} \\ 1 & 0 & a_{31} & a_{32} & a_{33} \\ 1 & 0 & a_{41} & a_{42} & a_{43} \\ 1 & 0 & a_{51} & a_{52} & a_{53} \\ 1 & 0 & a_{61} & a_{62} & a_{63} \\ 1 & 0 & a_{71} & a_{72} & a_{73} \\ 1 & 0 & a_{81} & a_{82} & a_{83} \end{pmatrix}$$

In[130]:=
```
Row@Map[MatrixForm, {#, MapIndexed[sListPush[#1, 2, #2 - 5] &, #]} &@mat]
```

Out[130]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} -4 & -4 & a_{11} & a_{12} & a_{13} \\ -3 & -3 & a_{21} & a_{22} & a_{23} \\ -2 & -2 & a_{31} & a_{32} & a_{33} \\ -1 & -1 & a_{41} & a_{42} & a_{43} \\ 0 & 0 & a_{51} & a_{52} & a_{53} \\ 1 & 1 & a_{61} & a_{62} & a_{63} \\ 2 & 2 & a_{71} & a_{72} & a_{73} \\ 3 & 3 & a_{81} & a_{82} & a_{83} \end{pmatrix}$$

Observe carefully how the padding is specified in PadRight, PadLeft:

In[131]:=
```
Row@
 Map[MatrixForm,
  {#, sListPush[#, 4, Table[i^k, {i, 4}, {k, Last@Dimensions@mat}]]} &@mat]
```

Out[131]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 4 & 8 & 16 & 32 \\ 3 & 9 & 27 & 81 & 243 \\ 4 & 16 & 64 & 256 & 1024 \\ a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{pmatrix}$$

In[132]:=
```
Row@
 Map[MatrixForm,
  {#, sListPush[#, -6, Table[i^k, {i, 6}, {k, Last@Dimensions@mat}]]} &@mat]
```

Out[132]=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \end{pmatrix} \begin{pmatrix} a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} \\ 3 & 9 & 27 & 81 & 243 \\ 4 & 16 & 64 & 256 & 1024 \\ 5 & 25 & 125 & 625 & 3125 \\ 6 & 36 & 216 & 1296 & 7776 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 4 & 8 & 16 & 32 \end{pmatrix}$$

Pushing 3D arrays (spatial matrices)

We generate a test 3D matrix:

In[133]:=
```
m3 = Array["a"<>FromDigits[{##}] &, {3, 4, 5}];
{r, c, d} = Dimensions[m3];
```

Push 3D matrix rows only:

In[134]:=
```
Row@Map[MatrixForm, {m3, sListPush[m3, 1, {ConstantArray[0, {c, d}]}]}]
```

Out[134]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\ \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \end{pmatrix} \end{pmatrix}$$

In[135]:=
```
Row@
 Map[MatrixForm,
  {m3, sListPush[m3, -2, {ConstantArray[0, {c, d}],
     ConstantArray[1, {c, d}]}]}]
```

Out[135]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \\ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \end{pmatrix} \end{pmatrix}$$

In[136]:=
```
Row@Map[MatrixForm, {m3, sListPush[m3, -2, {Table[i^k, {i, c}, {k, d}]}]}]
```

Out[136]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \Bigg| \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \Bigg| \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 2 \\ 4 \\ 8 \\ 16 \\ 32 \end{pmatrix} & \begin{pmatrix} 3 \\ 9 \\ 27 \\ 81 \\ 243 \end{pmatrix} & \begin{pmatrix} 4 \\ 16 \\ 64 \\ 256 \\ 1024 \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \Bigg| \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 2 \\ 4 \\ 8 \\ 16 \\ 32 \end{pmatrix} & \begin{pmatrix} 3 \\ 9 \\ 27 \\ 81 \\ 243 \end{pmatrix} & \begin{pmatrix} 4 \\ 16 \\ 64 \\ 256 \\ 1024 \end{pmatrix} \end{pmatrix}$$

Push 3D matrix columns only:

In[137]:=
```
Row@Map[MatrixForm, {m3, sListPush[#, 1, ConstantArray[0, {c, d}]] & /@ m3}]
```

Out[137]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \Bigg| \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \Bigg| \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \Bigg| \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} \end{pmatrix}$$

In[138]:=
```
Row@
 Map[MatrixForm,
  {m3, sListPush[#, -1, ConstantArray[Range[d, 1, -1], d]] & /@ m3}]
```

Out[138]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} & \begin{pmatrix} 5 \\ 4 \\ 3 \\ 2 \\ 1 \end{pmatrix} \end{pmatrix}$$
$$\begin{pmatrix} \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} & \begin{pmatrix} 5 \\ 4 \\ 3 \\ 2 \\ 1 \end{pmatrix} \end{pmatrix}$$
$$\begin{pmatrix} \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} & \begin{pmatrix} 5 \\ 4 \\ 3 \\ 2 \\ 1 \end{pmatrix} \end{pmatrix}$$

Push 3D matrix depth only:

In[139]:=
```
Row@Map[MatrixForm, {m3, Map[sListPush[#, -2, 0] &, m3, {2}]}]
```

Out[139]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{113} \\ a_{114} \\ a_{115} \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{123} \\ a_{124} \\ a_{125} \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{133} \\ a_{134} \\ a_{135} \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{143} \\ a_{144} \\ a_{145} \\ 0 \\ 0 \end{pmatrix} \end{pmatrix}$$
$$\begin{pmatrix} \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{213} \\ a_{214} \\ a_{215} \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{223} \\ a_{224} \\ a_{225} \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{233} \\ a_{234} \\ a_{235} \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{243} \\ a_{244} \\ a_{245} \\ 0 \\ 0 \end{pmatrix} \end{pmatrix}$$
$$\begin{pmatrix} \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} a_{313} \\ a_{314} \\ a_{315} \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{323} \\ a_{324} \\ a_{325} \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{333} \\ a_{334} \\ a_{335} \\ 0 \\ 0 \end{pmatrix} & \begin{pmatrix} a_{343} \\ a_{344} \\ a_{345} \\ 0 \\ 0 \end{pmatrix} \end{pmatrix}$$

Combine push in both rows and columns:

In[140]:=
```
Row@
 Map[MatrixForm,
  {m3, sListPush[#, 2, ConstantArray[CharacterRange[123 - d, 122], d]] & /@
    sListPush[m3, 1, {ConstantArray[CharacterRange[97, 96 + d], c]}]}]
```

Out[140]=

$$\left(\begin{array}{cccc|cccc}
\begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} & \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} \\
\begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} \\
\begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix}
\end{array}\right)$$

In[141]:=
```
Row@
 Map[MatrixForm,
  {m3,
   sListPush[
    sListPush[#, 2, ConstantArray[CharacterRange[123 - d, 122], d]] & /@ m3,
    1, {ConstantArray[CharacterRange[97, 96 + d], c]}]}]
```

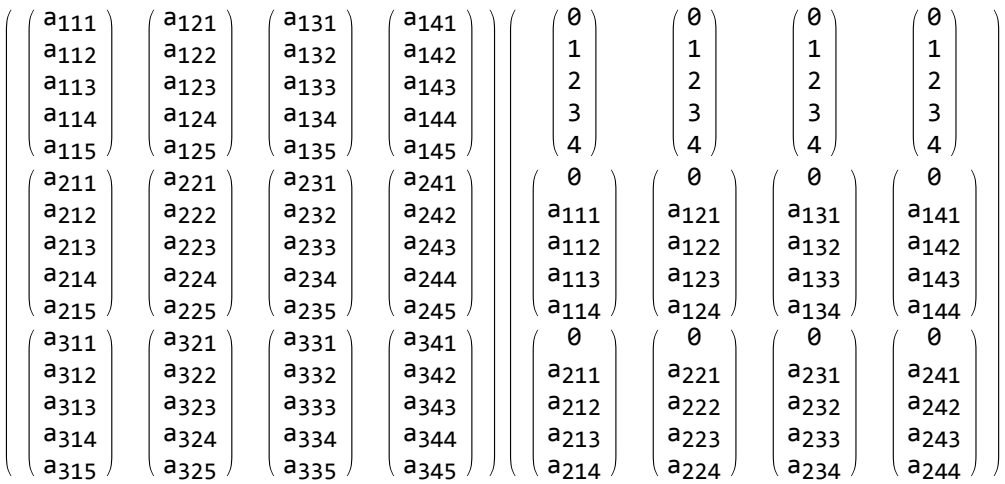
Out[141]=

$$\left(\begin{array}{cccc|cccc}
\begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} & \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} & \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} & \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} & \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} \\
\begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} \\
\begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix}
\end{array}\right)$$

Combine push in both rows and depth:

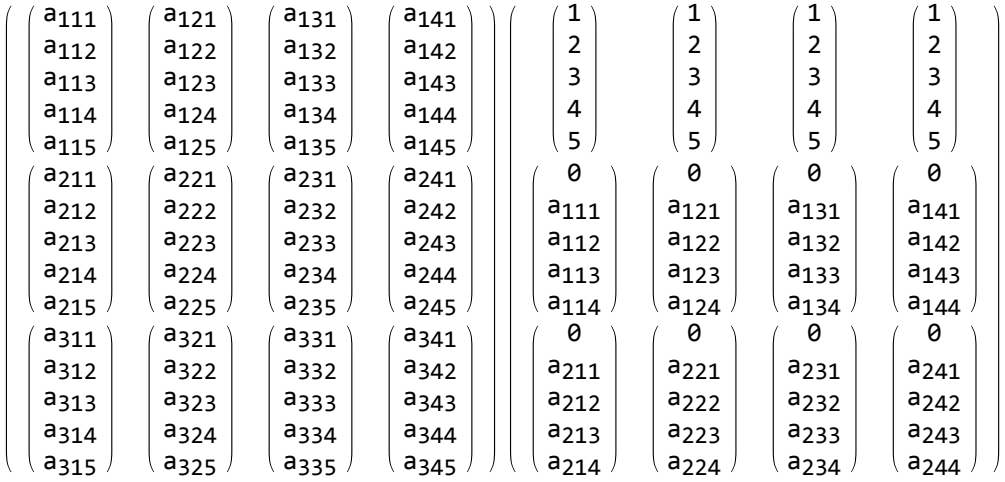
In[142]:=
```
Row@
 Map[MatrixForm,
  {m3, Map[sListPush[#, 1, 0] &,
    sListPush[m3, 1, {ConstantArray[Range[d], c]}], {2}]}]
```

Out[142]=



In[143]:=
```
Row@
 Map[MatrixForm,
  {m3, sListPush[Map[sListPush[#, 1, 0] &, m3, {2}], 1,
    {ConstantArray[Range[d], c]}]}]
```

Out[143]=



Combine push in both columns and depth:

In[144]:=
```
Row@
 Map[MatrixForm,
  {m3, Map[sListPush[#, 2, 0] &,
    sListPush[#, 1, ConstantArray[Range[d], c]] & /@ m3, {2}]}]
```

Out[144]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{311} \\ a_{312} \\ a_{313} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{321} \\ a_{322} \\ a_{323} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{331} \\ a_{332} \\ a_{333} \end{pmatrix} \end{pmatrix}$$

In[145]:=
```
Row@
 Map[MatrixForm,
  {m3, sListPush[#, 1, ConstantArray[Range[d], c]] & /@
    Map[sListPush[#, 2, 0] &, m3, {2}]}]
```

Out[145]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} \\ \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} \\ \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{311} \\ a_{312} \\ a_{313} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{321} \\ a_{322} \\ a_{323} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{331} \\ a_{332} \\ a_{333} \end{pmatrix} \end{pmatrix}$$

Combine push in all three dimensions:

In[146]:=
```
Row@
 Map[MatrixForm,
  {m3, Map[sListPush[#, 2, 0] &,
    sListPush[#, 1, ConstantArray[CharacterRange[123 - d, 122], d]] & /@
     sListPush[m3, 1, {ConstantArray[Range[d], c]}], {2}]}]
```

Out[146]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 0 \\ 0 \\ v \\ w \\ x \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ v \\ w \\ x \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ v \\ w \\ x \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} \end{pmatrix}$$

In[147]:=
```
Row@
 Map[MatrixForm,
  {m3, Map[sListPush[#, 2, 0] &,
    sListPush[
     sListPush[#, 1, ConstantArray[CharacterRange[123 - d, 122], d]] & /@ m3,
     1, {ConstantArray[Range[d], c]}], {2}]}]
```

Out[147]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ v \\ w \\ x \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ v \\ w \\ x \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} \end{pmatrix}$$

In[148]:=
```
Row@
 Map[MatrixForm,
  {m3, sListPush[#, 1, ConstantArray[CharacterRange[123 - d, 122], d]] & /@
    sListPush[Map[sListPush[#, 2, 0] &, m3, {2}], 1,
     {ConstantArray[Range[d], c]}]}]
```

Out[148]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \left(\begin{array}{c|ccc} \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \\ \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} \\ \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} \end{array} \right)$$

In[149]:=
```
Row@
 Map[MatrixForm,
  {m3, sListPush[#, 1, ConstantArray[CharacterRange[123 - d, 122], d]] & /@
    Map[sListPush[#, 2, 0] &, sListPush[m3, 1, {ConstantArray[Range[d], c]}],
     {2}]}]
```

Out[149]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \left(\begin{array}{c|ccc} \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} \\ \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} \\ \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} \end{array} \right)$$

In[150]:=
```
Row@
 Map[MatrixForm,
  {m3, sListPush[#, 1, ConstantArray[CharacterRange[123 - d, 122], d]] & /@
    sListPush[Map[sListPush[#, 2, 0] &, m3, {2}], 1,
     {ConstantArray[Range[d], c]}]}]
```

Out[150]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \\ \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} \\ \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} \end{pmatrix}$$

In[151]:=
```
Row@
 Map[MatrixForm,
  {m3, sListPush[Map[sListPush[#, 2, 0] &,
     sListPush[m3, 1, {ConstantArray[Range[d], c]}], {2}], 1,
    {ConstantArray[Range[d], c]}]}]
```

Out[151]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 3 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 0 \\ a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{141} \\ a_{142} \\ a_{143} \end{pmatrix} \end{pmatrix}$$

In[152]:=
```
Row@
 Map[MatrixForm,
  {m3,
   sListPush[
    sListPush[#, 1, ConstantArray[CharacterRange[123 - d, 122], d]] & /@
     Map[sListPush[#, 2, 0] &, m3, {2}], 1, {ConstantArray[Range[d], c]}]}]
```

Out[152]=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \\ a_{113} \\ a_{114} \\ a_{115} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \\ a_{123} \\ a_{124} \\ a_{125} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \\ a_{133} \\ a_{134} \\ a_{135} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \\ a_{143} \\ a_{144} \\ a_{145} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \\ a_{213} \\ a_{214} \\ a_{215} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \\ a_{223} \\ a_{224} \\ a_{225} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \\ a_{233} \\ a_{234} \\ a_{235} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \\ a_{243} \\ a_{244} \\ a_{245} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \\ a_{313} \\ a_{314} \\ a_{315} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \\ a_{323} \\ a_{324} \\ a_{325} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \\ a_{333} \\ a_{334} \\ a_{335} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \\ a_{343} \\ a_{344} \\ a_{345} \end{pmatrix} \end{pmatrix} \left| \begin{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \\ \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{111} \\ a_{112} \\ a_{113} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{121} \\ a_{122} \\ a_{123} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{131} \\ a_{132} \\ a_{133} \end{pmatrix} \\ \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{211} \\ a_{212} \\ a_{213} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{221} \\ a_{222} \\ a_{223} \end{pmatrix} & \begin{pmatrix} 0 \\ 0 \\ a_{231} \\ a_{232} \\ a_{233} \end{pmatrix} \end{pmatrix} \right.$$

Applications

Audio push demonstration

We generate a test audio—a log sweep (the data of which is actually a 1D array):

In[153]:=
```
aud = AudioGenerator[{"Sin", 200 π # &}, 8, SampleRate → 16 000]
adat = AudioData[aud];
apar = Through[{AudioSampleRate, AudioType, AudioChannels, Duration}[aud]]
```

Out[153]=

Out[155]= $\{16\,000\ \text{Hz}, \text{Real32}, 1, 8.\ \text{s}\}$

Now we perform the audio push:

In[156]:=
```
Audio[sListPush[#, 10^5, 0.0] & /@ adat, SampleRate → apar[[1, 1]]]
Through[{AudioSampleRate, AudioType, AudioChannels, Duration}[%]]
```

Out[156]=

Out[157]=
```
{16 000 Hz , Real32, 1, 8. s}
```

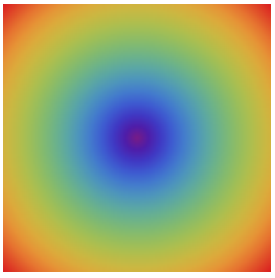
This can be useful, e.g., for stereo channels mutual time shift, or for synchronization of audio and video tracks.

Image push demonstration

We generate a test image (which is actually a 2D array):

In[158]:=
```
img = RadialGradientImage["Rainbow", {128, 128}, ImageSize → Tiny];
idat = ImageData@img;
idesc = {(*ImageType,*)ImageDimensions(*,ImageChannels*)};
Row[{img, Through[idesc[#]] &[img]}, Spacer[10]]
```

Out[160]=

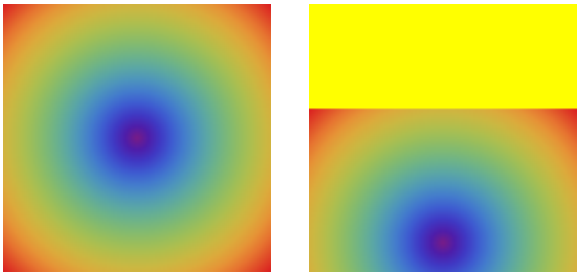


```
{{128, 128}}
```

Here we present a few examples of image row pushing. The reader is encouraged to browse all the code thoroughly:

In[161]:=
```
res = Image[sListPush[idat, 50, {ConstantArray[{1., 1., 0.}, Length@idat]}],
   ImageSize → Tiny];
Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

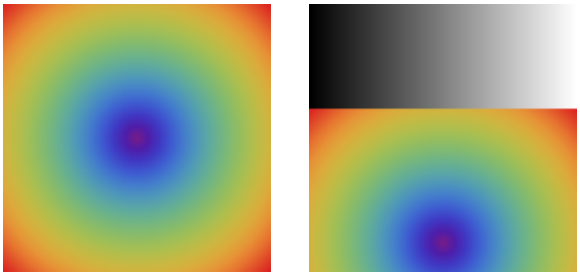
Out[162]=



```
{{128, 128}}
```

In[163]:=
```
res = With[{l = Length@idat},
   Image[sListPush[idat, 50, {Table[{(k - 1)/(l - 1), (k - 1)/(l - 1), (k - 1)/(l - 1)}, {k, l}]}],
    ImageSize → Tiny]];
Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

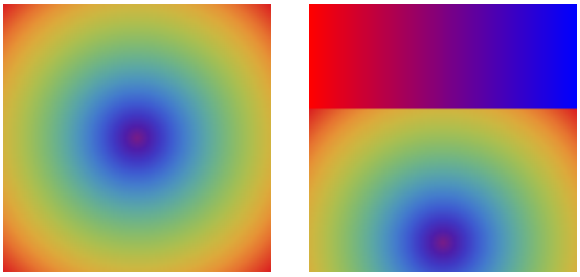
Out[164]=



{{128, 128}}

In[165]:=
```
res = With[{l = Length@idat},
   Image[sListPush[idat, 50, {Table[{(l - k)/(l - 1), 0, (k - 1)/(l - 1)}, {k, l}]}],
    ImageSize → Tiny]];
Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

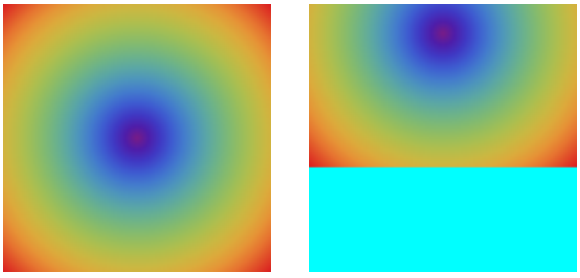
Out[166]=



{{128, 128}}

In[167]:=
```
res = Image[sListPush[idat, -50, {ConstantArray[{0., 1., 1.}, Length@idat]}],
   ImageSize → Tiny];
Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

Out[168]=

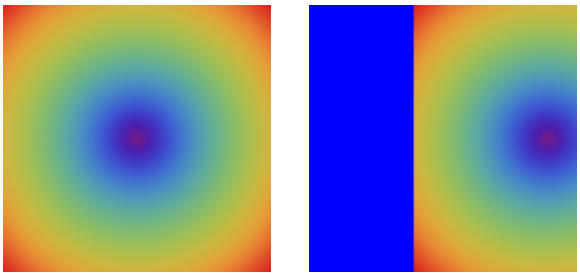


{{128, 128}}

Here you can see and browse the code of a few examples of image column pushing:

In[169]:=
```
res = Image[sListPush[#, 50, ConstantArray[{0., 0., 1.}, Length@idat]] & /@
    idat, ImageSize → Tiny];
Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

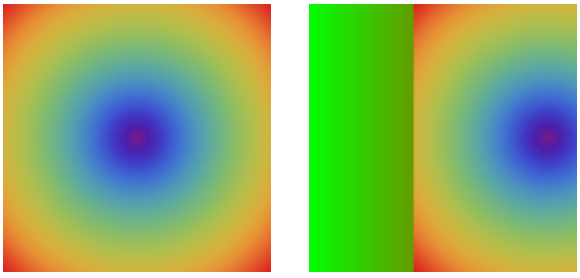
Out[170]=



{{128, 128}}

In[171]:=
```
res = With[{l = Length@idat},
   Image[sListPush[#, 50, Table[{k/(l - 1), (l - k)/(l - 1), 0}, {k, l}]] & /@ idat,
    ImageSize → Tiny]];
Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

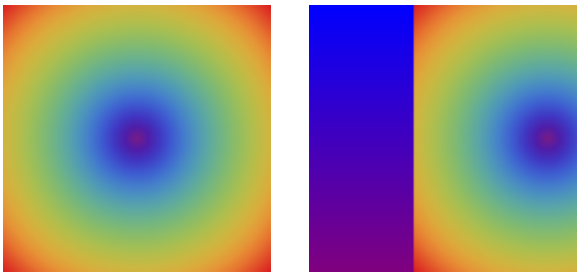
Out[172]=



{{128, 128}}

In[173]:=
```
res =
  Image[
   MapIndexed[
    sListPush[#1, 50, ConstantArray[{First@#2/255, 0., 1 - First@#2/255},
       Length@idat]] &, idat], ImageSize → Tiny];
Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

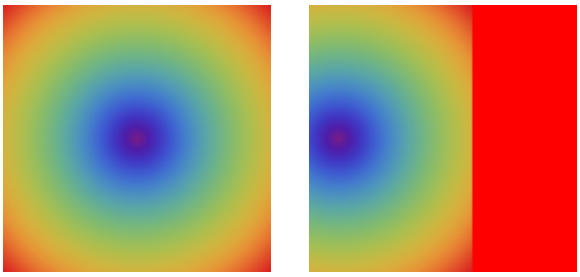
Out[174]=



{{128, 128}}

```
In[175]:= res = Image[sListPush[#, -50, ConstantArray[{1., 0., 0.}, Length@idat]] & /@
            idat, ImageSize → Tiny];
          Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

Out[176]=

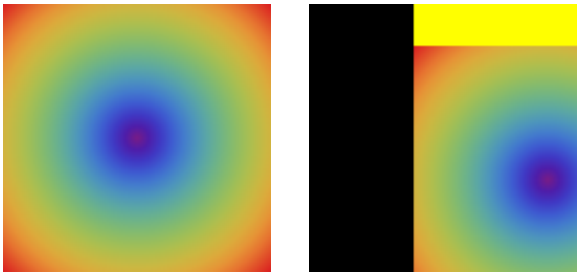


{{128, 128}}

Finally, we can push both image rows and columns in many variations:

```
In[177]:= res = Image[sListPush[#, 50, ConstantArray[{0., 0., 0.}, Length@idat]] & /@
             sListPush[idat, 20, {ConstantArray[{1., 1., 0.}, Length@idat]}],
           ImageSize → Tiny];
          Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

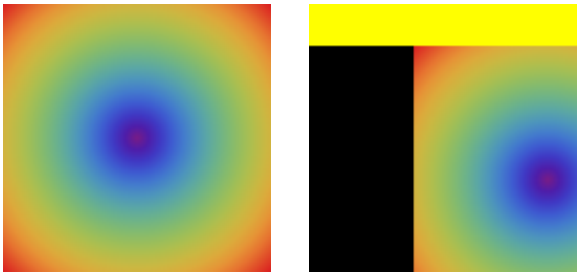
Out[178]=



{{128, 128}}

```
In[179]:= res =
            Image[
             sListPush[sListPush[#, 50, ConstantArray[{0., 0., 0.}, Length@idat]] & /@
               idat, 20, {ConstantArray[{1., 1., 0.}, Length@idat]}],
             ImageSize → Tiny];
          Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

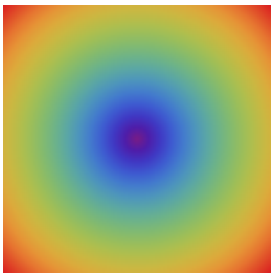
Out[180]=

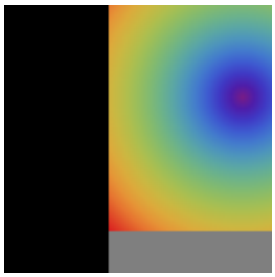


{{128, 128}}

In[181]:=
```
res = Image[sListPush[#, 50, ConstantArray[{0., 0., 0.}, Length@idat]] & /@
    sListPush[idat, -20, {ConstantArray[{.5, .5, .5}, Length@idat]}],
   ImageSize → Tiny];
Row[{img, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

Out[182]=





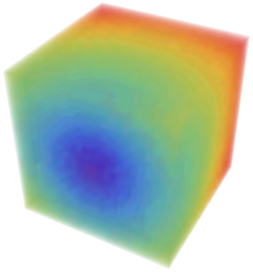
```
{{128, 128}}
```

3D image push demonstration

We generate a test image (which is actually a 3D array):

In[183]:=
```
im3 = RadialGradientImage[{Front, {Right, Back, Top}} → ColorData["Rainbow"],
   {24, 24, 24}, ImageSize → Tiny];
idt3 = ImageData[im3];
idesc = {(*ImageType,*)ImageDimensions(*,ImageChannels*)};
Row[{im3, Through[idesc[#]] &[im3]}, Spacer[10]]
```

Out[185]=

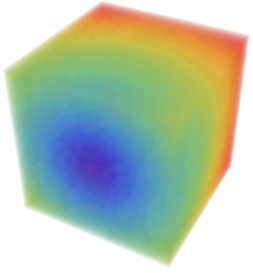


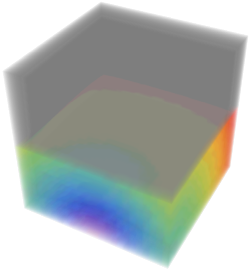
```
{{24, 24, 24}}
```

Here we present a few examples of 3D image row pushing:

In[186]:=
```
res =
  Image3D[sListPush[idt3, 12,
    Table[{.5, .5, .5}, Length@idt3, Length@idt3, Length@idt3]],
   ImageSize → Tiny];
Row[{im3, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

Out[187]=



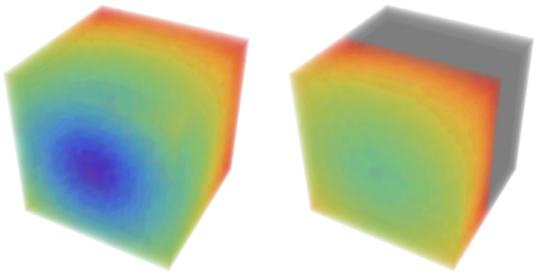


```
{{24, 24, 24}}
```

Here you can see a few examples of 3D image column pushing:

```
In[188]:= res = Image3D[sListPush[#, 12, {ConstantArray[{.5, .5, .5}, Length@idt3]}] & /@
             idt3, ImageSize → Tiny];
          Row[{im3, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

Out[189]=

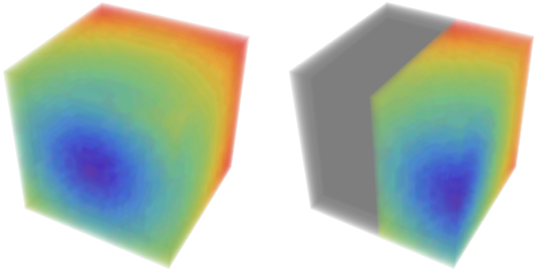


```
{{24, 24, 24}}
```

And here you can see a few examples of image depth pushing:

```
In[190]:= res = Image3D[Map[sListPush[#, 12, Table[{.5, .5, .5}, Length@idt3]] &,
             idt3, {2}], ImageSize → Tiny];
          Row[{im3, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

Out[191]=

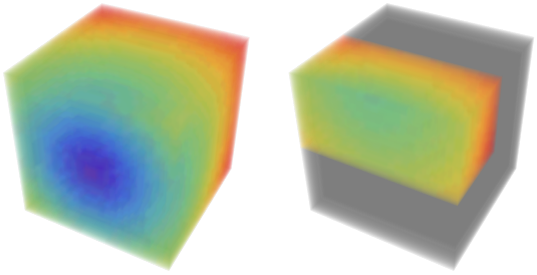


```
{{24, 24, 24}}
```

Here we demonstrate a few examples in which 3D image rows and columns are pushed:

```
In[192]:= res = Image3D[sListPush[#, 12, {ConstantArray[{.5, .5, .5}, Length@idt3]}] & /@
             sListPush[idt3, -12, Table[{.5, .5, .5}, Length@idt3, Length@idt3,
               Length@idt3]], ImageSize → Tiny];
          Row[{im3, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

Out[193]=

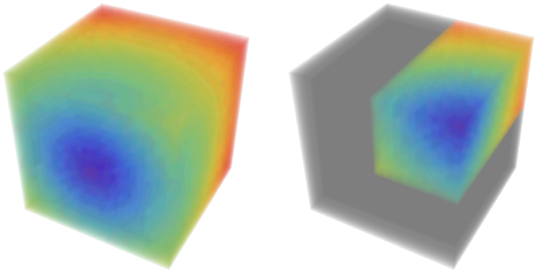


```
{{24, 24, 24}}
```

And here we demonstrate two examples in which 3D image rows (columns) and depth are pushed:

In[194]:=
```
res = Image3D[Map[sListPush[#, 12, Table[{.5, .5, .5}, Length@idt3]] &,
    sListPush[idt3, -12, Table[{.5, .5, .5}, Length@idt3, Length@idt3,
      Length@idt3]], {2}], ImageSize → Tiny];
Row[{im3, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

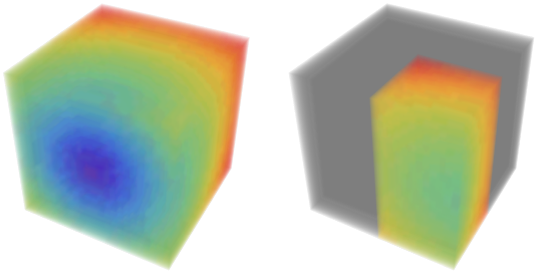
Out[195]=



{{24, 24, 24}}

In[196]:=
```
res = Image3D[Map[sListPush[#, 12, Table[{.5, .5, .5}, Length@idt3]] &,
    sListPush[#, 12, {ConstantArray[{.5, .5, .5}, Length@idt3]}] & /@ idt3,
    {2}], ImageSize → Tiny];
Row[{im3, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

Out[197]=

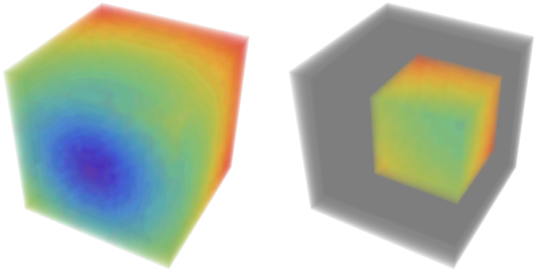


{{24, 24, 24}}

Finally, here is a demonstration in which 3D image all three dimensions are pushed:

In[198]:=
```
res = Image3D[sListPush[#, 12, {ConstantArray[{.5, .5, .5}, Length@idt3]}] & /@
    sListPush[Map[sListPush[#, 12, Table[{.5, .5, .5}, Length@idt3]] &,
      idt3, {2}], -12, Table[{.5, .5, .5}, Length@idt3, Length@idt3,
      Length@idt3]], ImageSize → Tiny];
Row[{im3, res, Through[idesc[#]] &[res]}, Spacer[10]]
```

Out[199]=



{{24, 24, 24}}

1.3 sListTruncate

In[200]:=
```
infoAbout@sListTruncate
```

sListTruncate[*list*, *test*] deletes the longest contiguous sequence
of trailing elements in *list* on which the application of *test* gives True.

```
Attributes[sListTruncate] = {Protected, ReadProtected}
```

```
SyntaxInformation[sListTruncate] = {ArgumentsPattern → {_, _, _.}}
```

Details

sListTruncate[*list*, *test*, *end*] deletes the longest contiguous sequence of leading (trailing) elements in *list* on which the application of *test* gives True if *end* is 1 (−1, default).

- *list* need not be a full array, only the top (outmost) level is affected; use Map or MapIndexed for deeper levels.
- A typical *test* is in the form of a pure function Abs[#] < δ & or RealAbs[#] < δ &, where δ is an appropriate positive real number; this cuts off longest contiguous sequence of trailing elements in *list* whose magnitude is less than δ. However, any sensible function that returns True or False can be used in place of *test*.

Examples

Clearing global symbols:

In[201]:=
```
Clear[u1, u2, u3, u4, u5, u6, u7, u8, c, x];
```

First, we define a test list:

In[202]:=
```
u1 = {4, -3, 2, -1, 10, 11, 13, 14, 2, 17, 0, -18, 1, -1, 19, -20, 4, 21,
    -4, 3, -2, 1, 0};
```

Truncate trailing elements with absolute value less than 5. There are elements satisfying the test in the result, but they are cut off by the element with value of 21:

In[203]:=
```
Column@{u1, sListTruncate[u1, RealAbs[#] < 5 &]}
```

Out[203]=
```
{4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
 0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
{4, -3, 2, -1, 10, 11, 13, 14, 2, 17, 0, -18, 1, -1, 19, -20, 4, 21}
```

Truncate leading elements with magnitude less than 5:

In[204]:=
```
Column[{u1, sListTruncate[u1, RealAbs[#] < 5 &, 1]}, Alignment → Right]
```

Out[204]=
```
                          {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
                           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
{10, 11, 13, 14, 2, 17, 0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
```

Truncate leading elements with absolute value less than 20:

```
In[205]:= Column[{u1, sListTruncate[u1, RealAbs[#] < 20 &, 1]}, Alignment → Right]

Out[205]= {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
                          {-20, 4, 21, -4, 3, -2, 1, 0}
```

No elements are truncated on either end if the test is changed as follows:

```
In[206]:= Column@{u1, sListTruncate[u1, RealAbs[#] ≥ 5 &]}

Out[206]= {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
          {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}

In[207]:= Column@{u1, sListTruncate[u1, RealAbs[#] ≥ 5 &, 1]}

Out[207]= {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
          {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
```

No elements are truncated when *test* is explicitly designed to return `False` for any argument:

```
In[208]:= Column@{u1, sListTruncate[u1, False &]}
          Column[{u1, sListTruncate[u1, False &, 1]}, Alignment → Right]

Out[208]= {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
          {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}

Out[209]= {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
          {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
```

Contrary, all elements are truncated if *test* returns `True` for all arguments:

```
In[210]:= Column@{u1, sListTruncate[u1, True &]}
          Column[{u1, sListTruncate[u1, True &, 1]}, Alignment → Right]

Out[210]= {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
          {}

Out[211]= {4, -3, 2, -1, 10, 11, 13, 14, 2, 17,
           0, -18, 1, -1, 19, -20, 4, 21, -4, 3, -2, 1, 0}
                                                    {}
```

Truncate all trailing elements with magnitude less than 0.1:

```
In[212]:= u2 = {10^-4, -10^-2, -10^-1, -0.45, 10^-2, -0.37, -0.92, -10^-4, -9.3, 6.4,
              -9.2, 10^-1, -10^-2, 10^-3, -10^-4};
          Column@{u2, sListTruncate[u2, RealAbs[#] < 10^-1 &]}
```

```
Out[213]= {1/10000, -1/100, -1/10, -0.45, 1/100, -0.37, -0.92,
           -1/10000, -9.3, 6.4, -9.2, 1/10, -1/100, 1/1000, -1/10000}
          {1/10000, -1/100, -1/10, -0.45, 1/100, -0.37, -0.92, -1/10000, -9.3, 6.4, -9.2, 1/10}
```

The same but for leading elements:

```
In[214]:= Column[{u2, sListTruncate[u2, RealAbs[#] < 10^-1 &, 1]}, Alignment → Right]
```

```
Out[214]= {1/10000, -1/100, -1/10, -0.45, 1/100, -0.37, -0.92,
           -1/10000, -9.3, 6.4, -9.2, 1/10, -1/100, 1/1000, -1/10000}
          {-1/10, -0.45, 1/100, -0.37, -0.92,
           -1/10000, -9.3, 6.4, -9.2, 1/10, -1/100, 1/1000, -1/10000}
```

Truncate all trailing (leading) even elements:

```
In[215]:= u3 = {0, -6, 2, 3, 4, 5, 6, 8, 10, 12};
          Column@{u3, sListTruncate[u3, EvenQ]}
          Column[{u3, sListTruncate[u3, EvenQ, 1]}, Alignment → Right]
```

```
Out[216]= {0, -6, 2, 3, 4, 5, 6, 8, 10, 12}
          {0, -6, 2, 3, 4, 5}
```

```
Out[217]= {0, -6, 2, 3, 4, 5, 6, 8, 10, 12}
                {3, 4, 5, 6, 8, 10, 12}
```

Any function can be used as test, here we try compositions:

```
In[218]:= Column@{u3, sListTruncate[u3, EvenQ@*Identity]}
```

```
Out[218]= {0, -6, 2, 3, 4, 5, 6, 8, 10, 12}
          {0, -6, 2, 3, 4, 5}
```

```
In[219]:= Column@{u3, sListTruncate[u3, Identity /* EvenQ]}
```

```
Out[219]= {0, -6, 2, 3, 4, 5, 6, 8, 10, 12}
          {0, -6, 2, 3, 4, 5}
```

Truncate all trailing (leading) expressions with length less or equal 3:

In[220]:=
```
u4 = {{{"a"}, "b", {"c", "d"}}, {}, "", {1, 2, 3, 4}, {4, 5}, {"x", "y"},
   {""}, {1, 2, 3, -5}, {1, 2, 5}, {""}, ""};
Column@{u4, sListTruncate[u4, Length[#] ≤ 3 &]}
Column[{u4, sListTruncate[u4, Length[#] ≤ 3 &, 1]}, Alignment → Right]
```

Out[221]=
```
{{{a}, b, {c, d}}, {}, , {1, 2, 3, 4},
 {4, 5}, {x, y}, {}, {1, 2, 3, -5}, {1, 2, 5}, {}, }
{{{a}, b, {c, d}}, {}, , {1, 2, 3, 4}, {4, 5}, {x, y}, {}, {1, 2, 3, -5}}
```

Out[222]=
```
                {{{a}, b, {c, d}}, {}, , {1, 2, 3, 4},
                 {4, 5}, {x, y}, {}, {1, 2, 3, -5}, {1, 2, 5}, {}, }
{{1, 2, 3, 4}, {4, 5}, {x, y}, {}, {1, 2, 3, -5}, {1, 2, 5}, {}, }
```

The following test always returns False, resulting to identity:

In[223]:=
```
u5 = {-55, -5, -1, 0, 2, 4, 6, 1, 5, 55};
Column@{u5, sListTruncate[u5, And[OddQ[#], EvenQ[#]] &]}
```

Out[224]=
```
{-55, -5, -1, 0, 2, 4, 6, 1, 5, 55}
{-55, -5, -1, 0, 2, 4, 6, 1, 5, 55}
```

Truncating a complex vector:

In[225]:=
```
u6 = {0.67 + 0.78 i, 0.81 + 0.99 i, 0.88 + 0.25 i, -0.01 + 0.44 i,
   1.2 × 10^-7 + 5.1 × 10^-7 i, 1.0 × 10^-11 + 2.8 × 10^-12 i};
Column@{u6, sListTruncate[u6, Abs[#] < 10^-6 &]}
```

Out[226]=
```
{0.67 + 0.78 i, 0.81 + 0.99 i, 0.88 + 0.25 i,
 -0.01 + 0.44 i, 1.2 × 10^-7 + 5.1 × 10^-7 i, 1. × 10^-11 + 2.8 × 10^-12 i}
{0.67 + 0.78 i, 0.81 + 0.99 i, 0.88 + 0.25 i, -0.01 + 0.44 i}
```

Truncate trailing (leading) string elements:

In[227]:=
```
u7 = {"a", "b", 1, 1.2, 7 / 8, "c", 22 / 7, "d", "e"};
Column@{u7, sListTruncate[u7, StringQ]}
Column[{u7, sListTruncate[u7, StringQ, 1]}, Alignment → Right]
```

Out[228]=
$$\left\{a, b, 1, 1.2, \frac{7}{8}, c, \frac{22}{7}, d, e\right\}$$
$$\left\{a, b, 1, 1.2, \frac{7}{8}, c, \frac{22}{7}\right\}$$

Out[229]=
$$\left\{a, b, 1, 1.2, \frac{7}{8}, c, \frac{22}{7}, d, e\right\}$$
$$\left\{1, 1.2, \frac{7}{8}, c, \frac{22}{7}, d, e\right\}$$

And finally a little bit less trivial usage:

In[230]:=
```
u8 = Range /@ {3, 2, 4, 5, 2, 3};
Column[{u8, Function[l, sListTruncate[l, # ≤ 3 &]] /@ u8},
 Alignment → Center]
```

Out[231]=
```
{{1, 2, 3}, {1, 2}, {1, 2, 3, 4}, {1, 2, 3, 4, 5}, {1, 2}, {1, 2, 3}}
          {{}, {}, {1, 2, 3, 4}, {1, 2, 3, 4, 5}, {}, {}}
```

In[232]:=
```
Column[{u8, Function[l, sListTruncate[l, # ≤ 3 &, 1]] /@ u8},
 Alignment → Center]
```

Out[232]=
```
{{1, 2, 3}, {1, 2}, {1, 2, 3, 4}, {1, 2, 3, 4, 5}, {1, 2}, {1, 2, 3}}
                  {{}, {}, {4}, {4, 5}, {}, {}}
```

Applications

Let us calculate the ChebyshevT approximation of a function (in this case we take natural exponential), the maximum absolute error being required below 10^{-6}:

In[233]:=
```
(c = 2/π Table[NIntegrate[e^x ChebyshevT[n, x]/Sqrt[1 - x^2], {x, -1, 1},
      PrecisionGoal → 10, WorkingPrecision → 30], {n, 0, 15}]) // Style[#, 7] &

Length@c
```

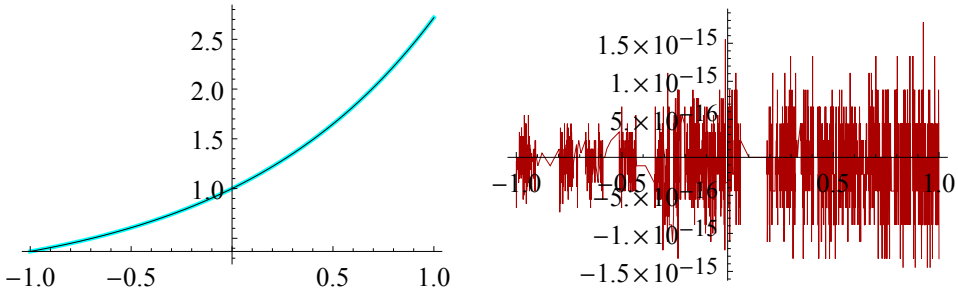
Out[233]=
```
{2.532131755504016671196489251 02, 1.130318207984970054415392055 69, 0.271495339534076562365705140676,
 0.044336849848663804952571496 1194, 0.005474240442093732650276168973 29, 0.000542926311913943750362148360922,
 0.000044977322954295146654691 1125004, 3.198436462401990505864127819 40×10^-6,
 1.992124806672795725961572709 86×10^-7, 1.103677172551734432616996093 11×10^-8,
 5.505896079673747250471420668 79×10^-10, 2.497956616984982522712010927 64×10^-11,
 1.039152230678570050499634673 34×10^-12, 3.991263356414401512887720397 48×10^-14,
 1.423758010825657148827367994 41×10^-15, 4.740926102561496171089928550 77×10^-17}
```

Out[234]=
```
16
```

It is obvious that 16 coefficients that have been calculated are unnecessary for the prescribed precision (the left plot shows the function and its approximation, the right plot shows the error):

In[235]:=
```
GraphicsRow[{Plot[{e^x,
    -First[c] / 2 + Evaluate[c . Table[ChebyshevT[n - 1, x], {n, Length[c]}]]},
   {x, -1, 1},
   PlotStyle → {Directive[Thickness[0.01], Cyan], Directive[Thin, Black]}],
 Plot[ e^x - (-First[c] / 2 +
      Evaluate[c . Table[ChebyshevT[n - 1, x], {n, Length[c]}]]),
   {x, -1, 1}, PlotStyle → Directive[Thin, Darker@Red]]},
 ImageSize → Medium]
```

Out[235]=



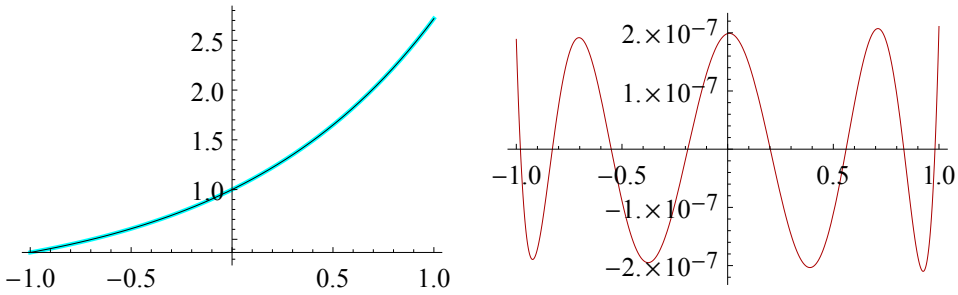
Truncating with the required precision preserves only 8 coefficients, wiping out the rest contiguous sequence of coefficients on which the application of the test gives True:

In[236]:=
```
(c = sListTruncate[c, RealAbs[#] < 10^-6 &]) // Style[#, 7] &
Length@c
GraphicsRow[{Plot[ {e^x,
    -First[c] / 2 + Evaluate[c. Table[ChebyshevT[n - 1, x], {n, Length[c]}]]},
   {x, -1, 1},
   PlotStyle → {Directive[Thickness[0.01], Cyan], Directive[Thin, Black]}],
 Plot[ e^x - (-First[c] / 2 +
      Evaluate[c . Table[ChebyshevT[n - 1, x], {n, Length[c]}]]),
   {x, -1, 1}, PlotStyle → Directive[Thin, Darker@Red]]},
 ImageSize → Medium]
```

Out[236]=
```
{2.532131755504016671196489251 02, 1.130318207984970054415392055 69, 0.271495339534076562365705140676,
 0.044336849848663804952571496 1194, 0.005474240442093732650276168973 29, 0.000542926311913943750362148360922,
 0.000044977322954295146654691 1125004, 3.198436462401990505864127819 40×10^-6}
```

Out[237]= 8

Out[238]=



1.4 sRotateHalfLength

In[239]:=
```
infoAbout@sRotateHalfLength
```

sRotateHalfLength[*u*] cycles all *n* elements in a list *u* ⌊*n*/2⌋ positions to the left; for odd *n*, the odd last element is deleted before that.
sRotateHalfLength[*u*, 1] in addition, the rotated list first element is copied on the right end.

```
Attributes[sRotateHalfLength] = {Protected, ReadProtected}

SyntaxInformation[sRotateHalfLength] = {ArgumentsPattern → {_, _., _.}}
```

Details

sRotateHalfLength[*u*, *als*, *test*] cycles all *n* elements in a list *u* by ⌊*n*/2⌋ positions to the left if *als* == 0 (default) or False; for odd *n*, the *u*'s odd last element is deleted before that, giving always even length.

- If *als* == 1 or True, the first element of the rotated list is duplicated on its right end in addition to the previous case, giving always odd length.
- Before deleting the odd last element, *test* is performed whether it is aliased with the first element (SameQ by default, use True & to completely eliminate this feature). If *test* fails, an info message is issued without stopping execution.
- *u* must be a full array of any depth, only the top (outmost) level is affected; use Map or MapIndexed for deeper levels.
- All elements must pass NumericQ.
- Typical usage comprises preprocessing a vector to pass it to the DFT represented by function Fourier or an akin symbol:

  Fourier[sRotateHalfLength[*u*]]

  or unwrapping the DFT output:

  sRotateHalfLength[Fourier[*u*], 1]

Examples

Clearing global symbols:

In[240]:=
```
Clear[optsListPlot, n, u, v, aud, adat, apar, spectrum, idesc, i2, i2e,
  i2o, res, i3, i3e, i3o, i3d];
ClearAll[rothlEx1, rothlEx2, rothlEx3, rothlEx4, rothlApp];
```

1D arrays (vectors)

Even length, no duplicating

First we will examine a vector of even length, calling sRotateHalfLength[*u*] (or

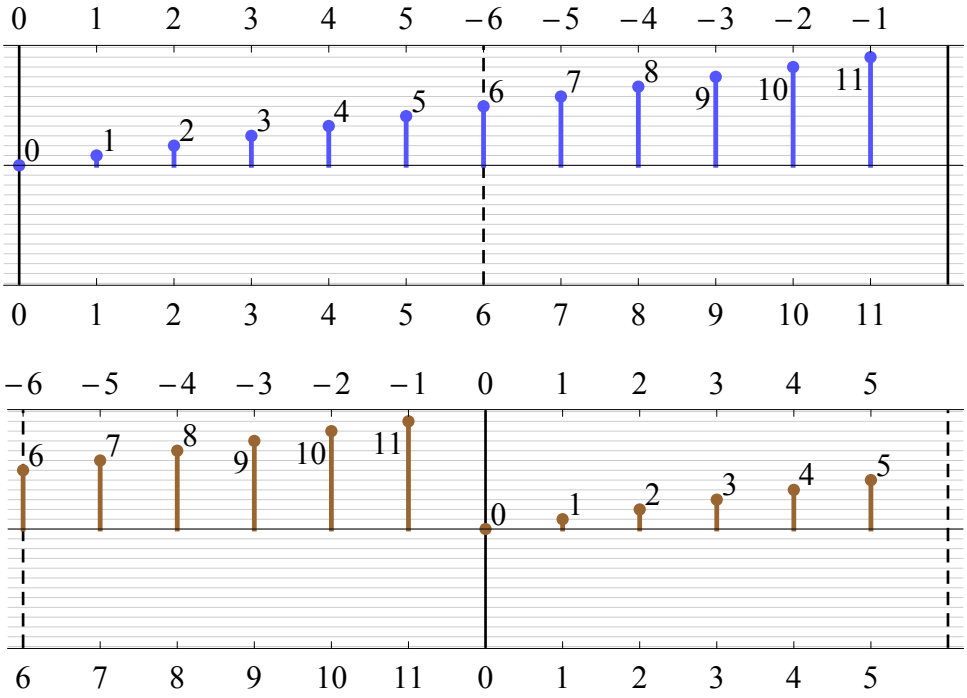
sRotateHalfLength[u, 0|False]), hence no duplication of the first element will happen. This allows restoring the options of the built-in symbol ListPlot to the original state:

In[242]:=
```
optsListPlot = Options[ListPlot];
```

Let us define some settings and a helper function, and visit Section 2.1 to make acquaintance with the symbol sSamplingSpan behavior:

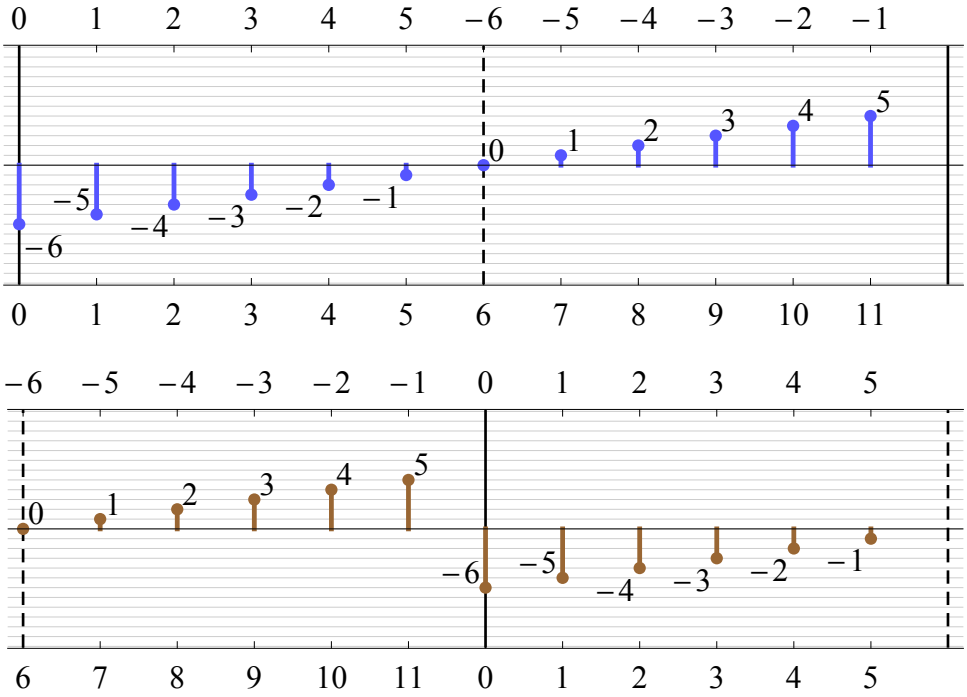
In[243]:=
```
n = 12;
SetOptions[ListPlot, PlotRange → {{-0.2, n + 0.2}, {-n - 0.2, n + 0.2}},
  Axes → True, Frame → {{False, False}, {True, True}}, DataRange → {0, n - 1},
  PlotStyle → Directive[PointSize[Large]], Filling → Axis,
  FillingStyle → Directive[Thickness[0.005]],
  GridLines → {None, Range[-n, n]}, LabelStyle → 12, AspectRatio → 1 / 4,
  ImageSize → Medium];
rothlEx1[u_] := Module[{v = sRotateHalfLength[u], n = Length@u, s},
  s = sSamplingSpan[n];
  If[v =!= $Failed, Column[{
     ListPlot[Labeled[#, #] & /@ u,
      FrameTicks → {{None, None},
        {s, {s, sRotateHalfLength[sSamplingSpan[-n]]}^T}},
      PlotStyle → Directive[Lighter[Blue]],
      Prolog → {Black, Line[{{0, -n}, {0, n}}], Line[{{n, -n}, {n, n}}],
        Dashed, Line[{{Floor[n / 2], -n}, {Floor[n / 2], n}}]}],
     ListPlot[Labeled[#, #] & /@ v,
      FrameTicks → {{None, None}, {{s, sRotateHalfLength[s]}^T,
         {s, sSamplingSpan[-n]}^T}}, PlotStyle → Directive[Brown],
      Prolog → {Black, Line[{{Floor[n / 2], -n}, {Floor[n / 2], n}}],
        Dashed, Line[{{0, -n}, {0, n}}], Line[{{n, -n}, {n, n}}]}]
    }, Alignment → Left, Spacings → {Scaled[0.025], Scaled[0.025]}]]]
```

In[246]:=
```
rothlEx1[sSamplingSpan[n]]
```



Out[246]=

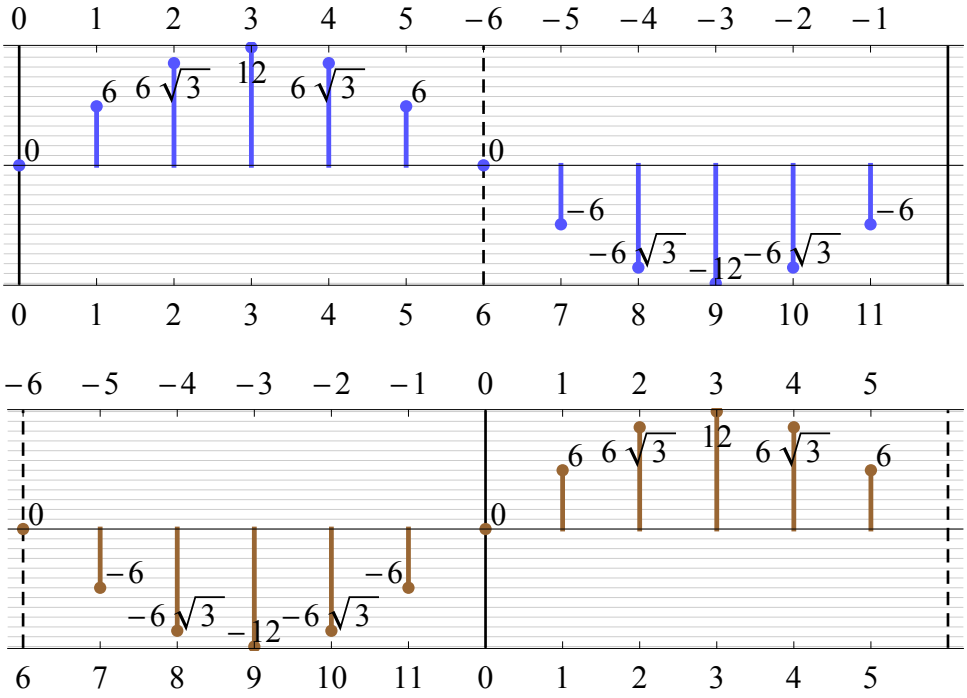
In[247]:=
```
rothlEx1[sSamplingSpan[-n]]
```



Out[247]=

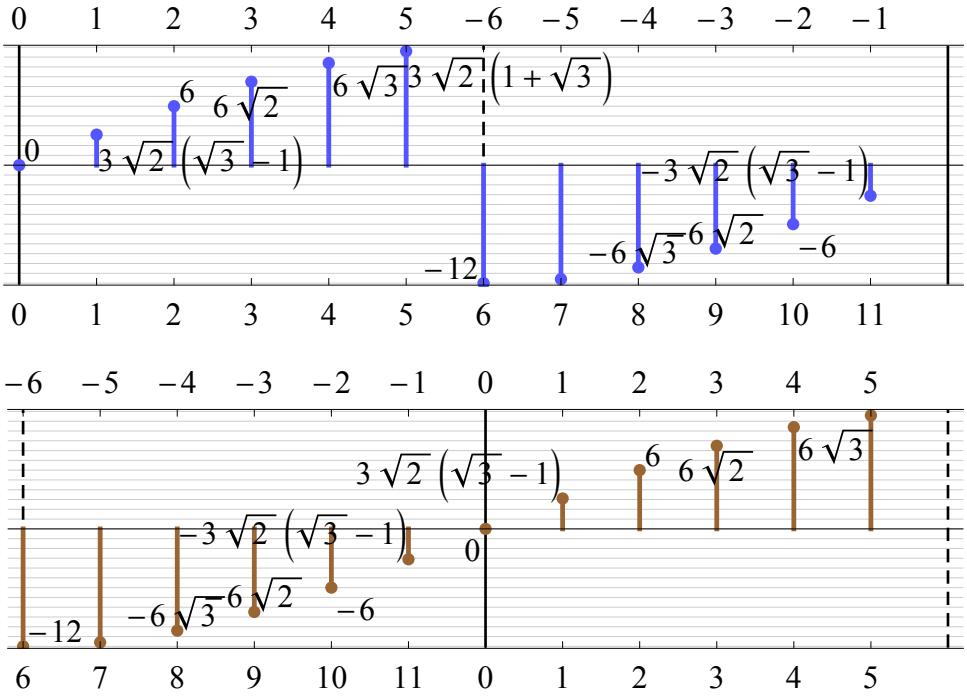
In[248]:=
```
rothlEx1[n Sin[sCircularlySample[{-π, π}, n]]]
```

Out[248]=



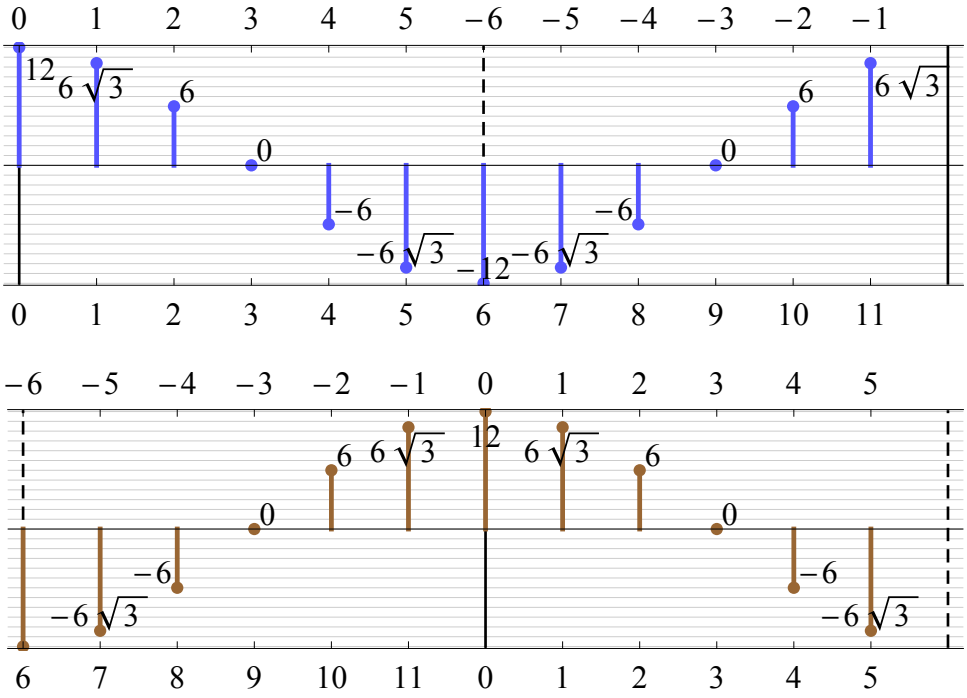
In[249]:=
```
rothlEx1[n Sin[sCircularlySample[{-π / 2, π / 2}, n]]]
```

Out[249]=



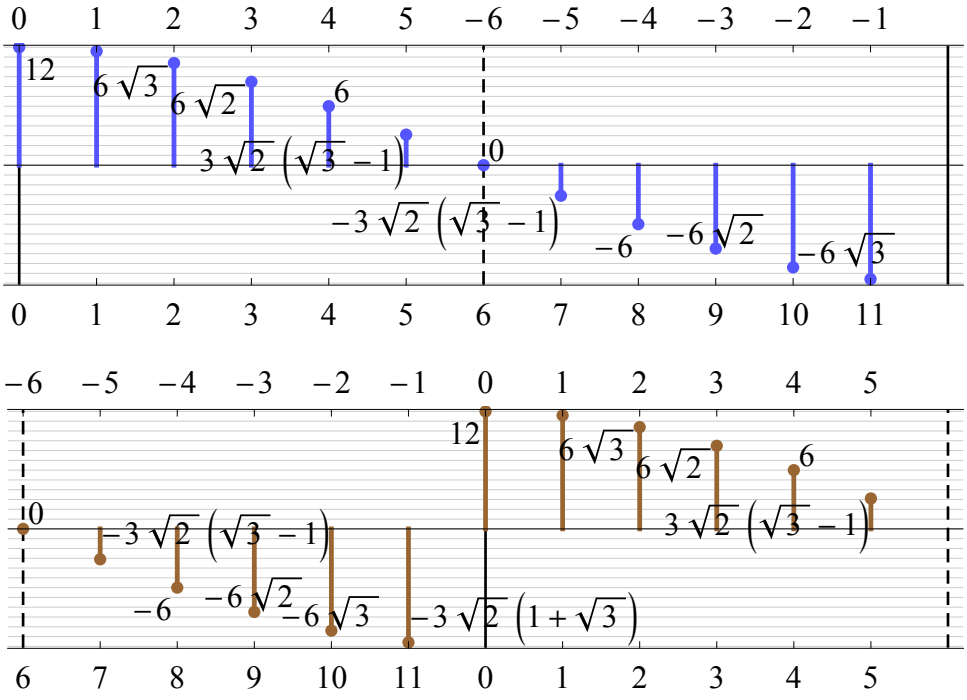
In[250]:=
```
rothlEx1[n Cos[sCircularlySample[{0, 2 π}, n]]]
```

Out[250]=



In[251]:=
```
rothlEx1[n Cos[sCircularlySample[{0, π}, n]]]
```

Out[251]=



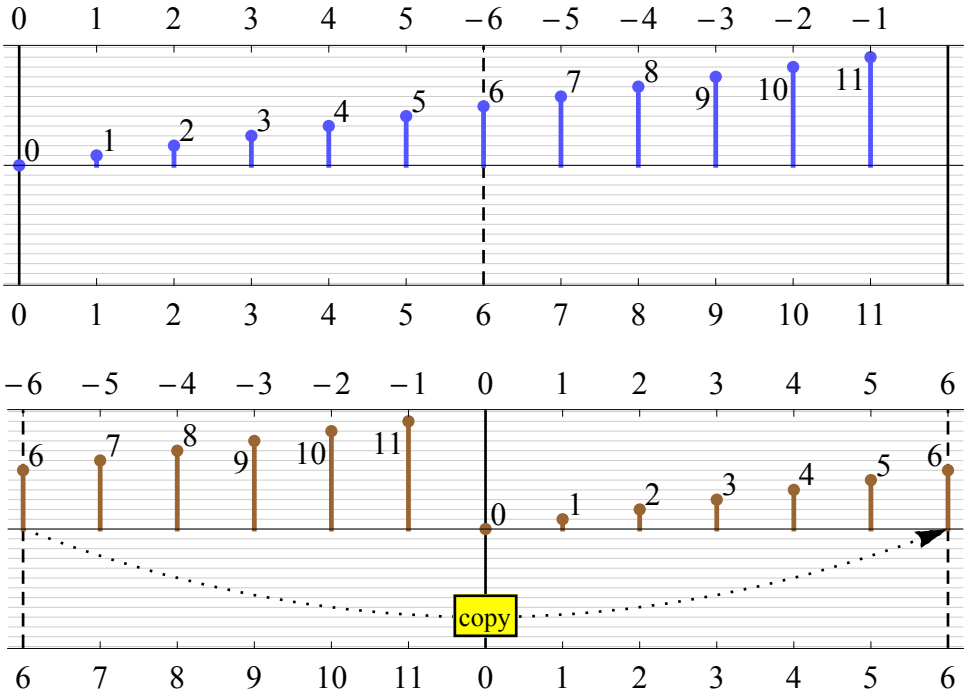
Even length, duplicating

Now we will examine a vector of even length, calling sRotateHalfLength[u, 1 | True], hence the first element of the rotated list is duplicated on its right end. Let us define a helper function, and visit Section 2.1 to make acquaintance with the symbol sSamplingSpan behavior:

In[252]:=
```
n = 12;
rothlEx2[u_] := Module[{v = sRotateHalfLength[u, 1], n = Length@u, s, s1},
  s = sSamplingSpan[n];
  s1 = sSamplingSpan[n + 1];
  If[v =!= $Failed, Column[{
     ListPlot[Labeled[#, #] & /@ u,
      FrameTicks → {{None, None},
        {s, {s, sRotateHalfLength[sSamplingSpan[-n]]}^T}},
      PlotStyle → Directive[Lighter[Blue]],
      Prolog → {Black, Line[{{0, -n}, {0, n}}], Line[{{n, -n}, {n, n}}],
        Dashed, Line[{{Floor[n / 2], -n}, {Floor[n / 2], n}}]}],
     ListPlot[Labeled[#, #] & /@ v, DataRange → {0, n},
      FrameTicks → {{None, None}, {{s1, sRotateHalfLength[s1, 1, True &]}^T,
         {s1, sSamplingSpan[-n - 1]}^T}}, PlotStyle → Directive[Brown],
      Prolog → {Black, Line[{{Floor[n / 2], -n}, {Floor[n / 2], n}}],
        Dashed, Line[{{0, -n}, {0, n}}], Line[{{n, -n}, {n, n}}],
        Dotted, Arrowheads[Medium],
        Arrow@BezierCurve[{{0, 0}, {n / 2, -1.5 n}, {n, 0}}]},
      Epilog →
       {Inset[Framed[Style["copy", 10], Background → Yellow,
          FrameMargins → Tiny], {n / 2, -3 n / 4}]}]
    }, Alignment → Left, Spacings → {Scaled[0.025], Scaled[0.025]}]]]
```

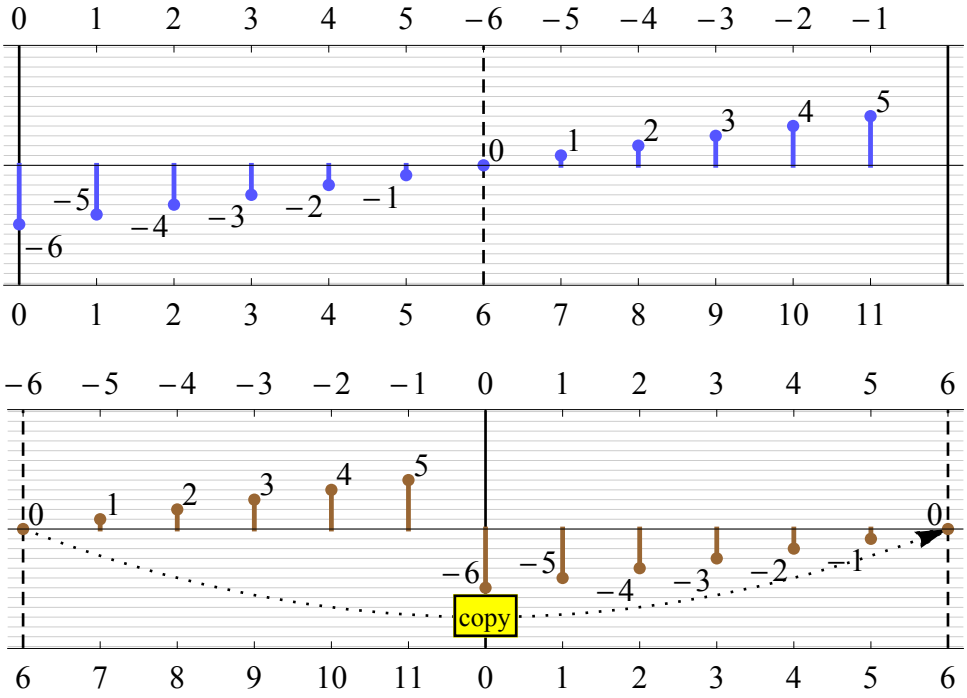
In[253]:=
```
rothlEx2[sSamplingSpan[n]]
```

Out[253]=



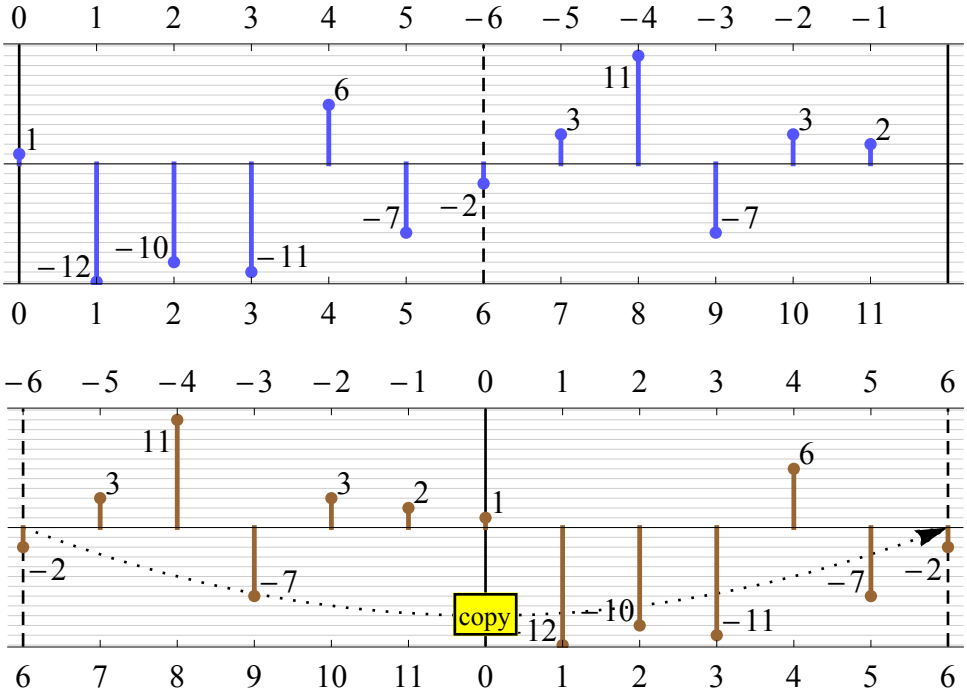
In[254]:=
```
rothlEx2[sSamplingSpan[-n]]
```

Out[254]=



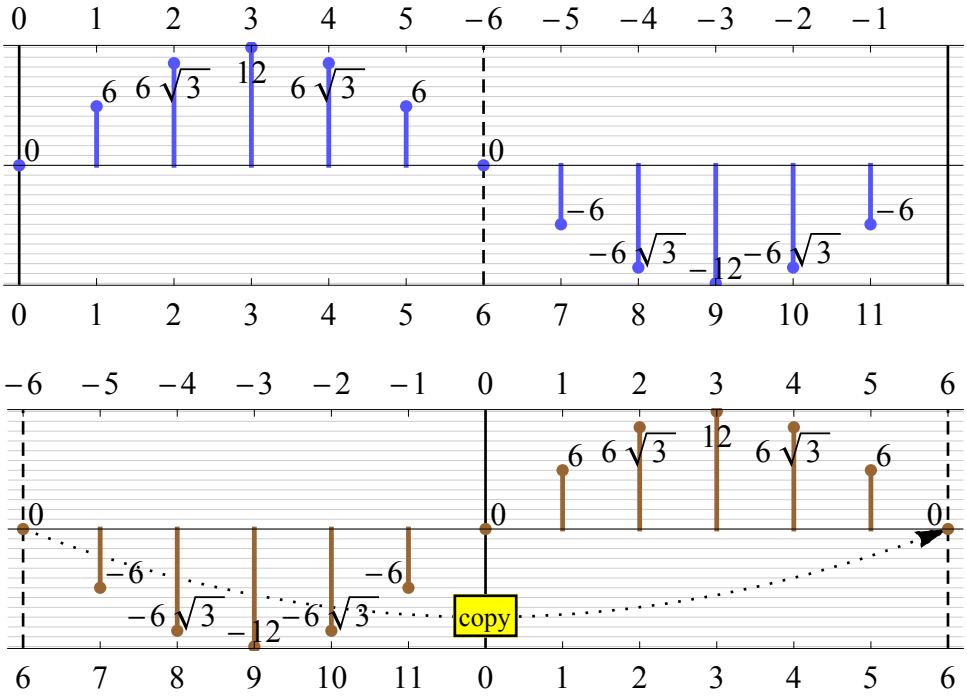
In[255]:=
```
rothlEx2[RandomInteger[{-n, n}, n]]
```

Out[255]=



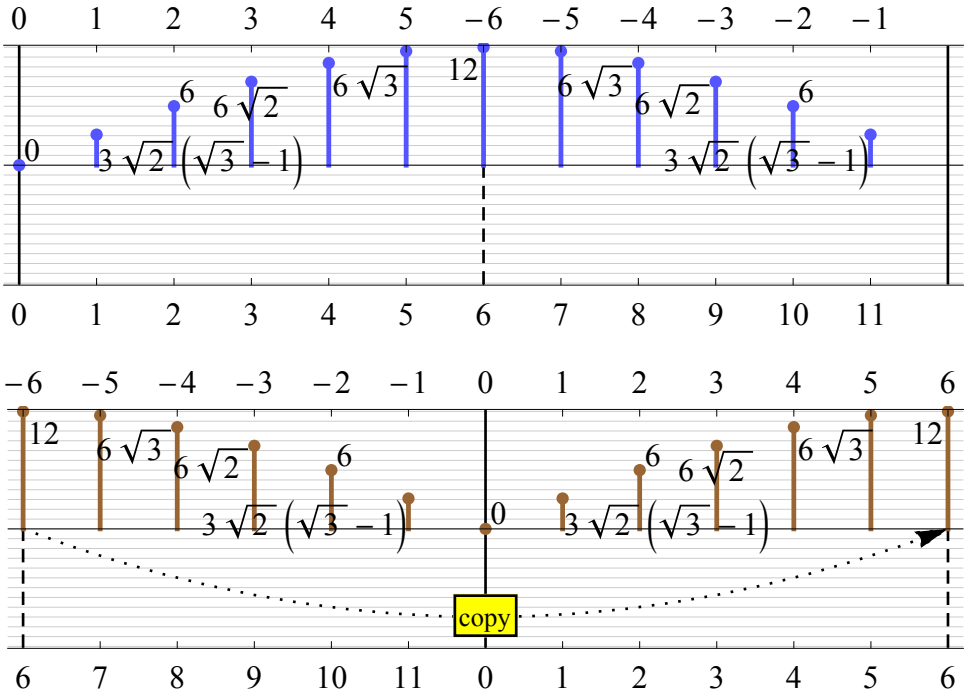
In[256]:=
```
rothlEx2[n Sin[sCircularlySample[{0, 2 π}, n]]]
```

Out[256]=



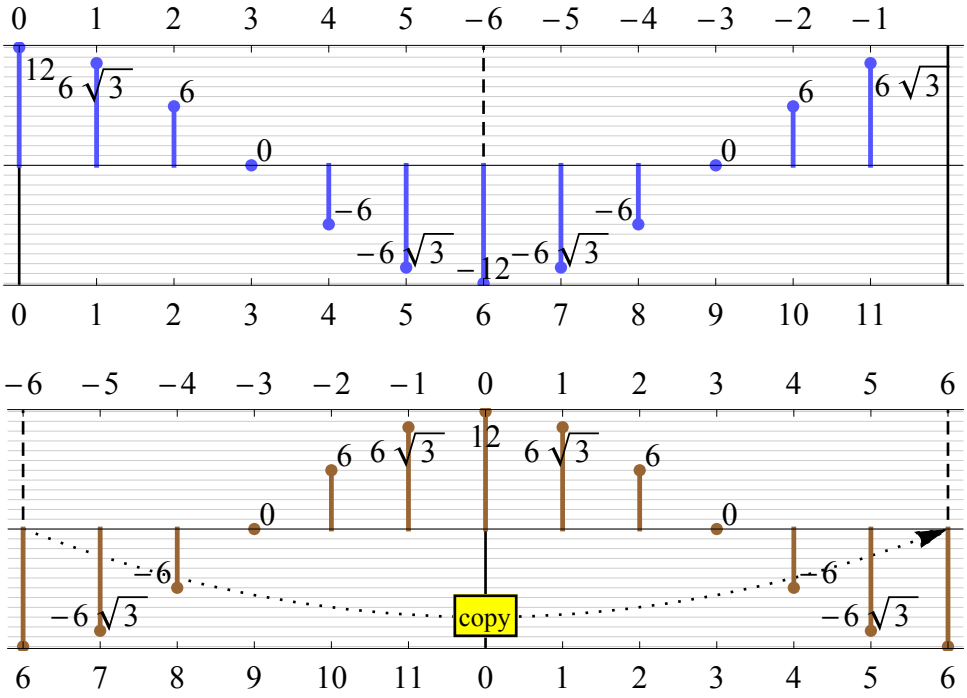
In[257]:=
```
rothlEx2[n Sin[sCircularlySample[{0, π}, n]]]
```

Out[257]=



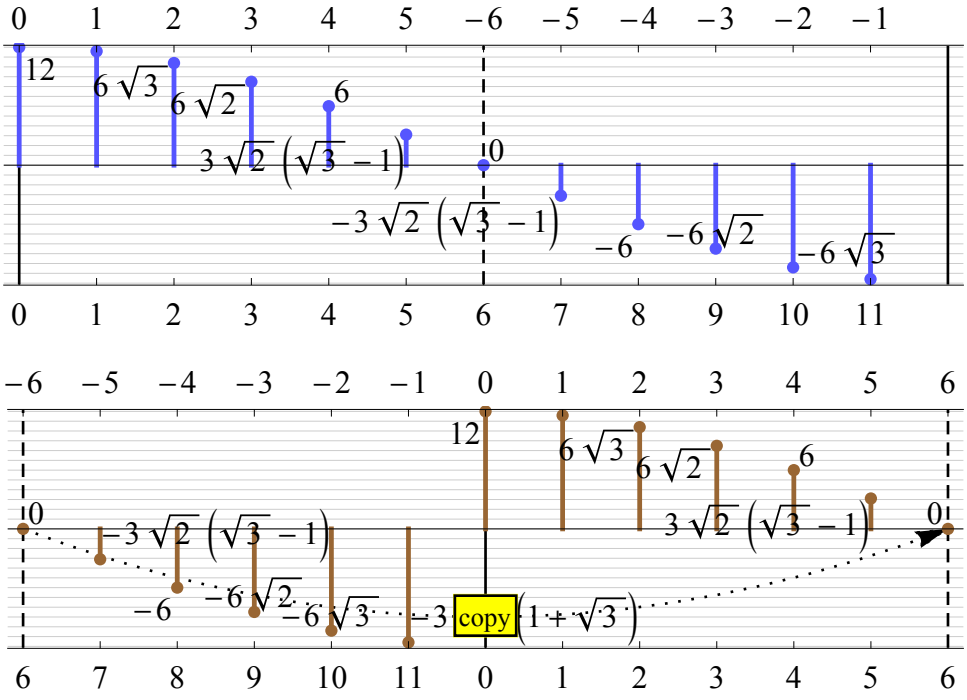
In[258]:=
```
rothlEx2[n Cos[sCircularlySample[{0, 2 π}, n]]]
```

Out[258]=



In[259]:=
```
rothlEx2[n Cos[sCircularlySample[{0, π}, n]]]
```

Out[259]=



Odd length, no duplicating

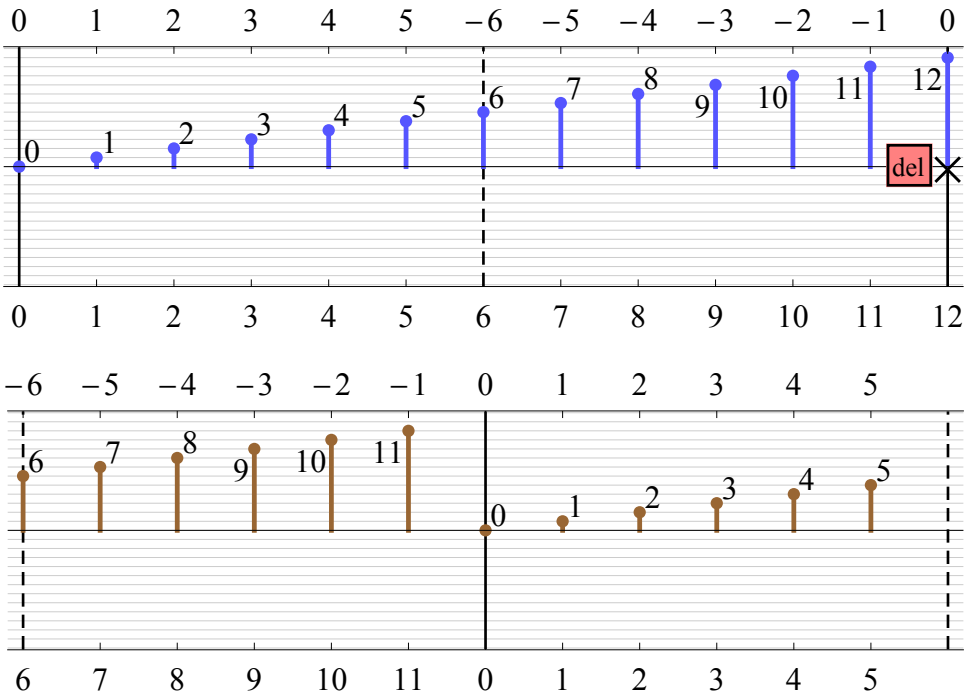
In the following we will demonstrate the behavior with a vector of odd length, calling sRotateHalfLength[*u*] (or sRotateHalfLength[*u*, 0|False]). Let us define some (restorable) settings and a helper function, and visit Section 2.1 to make acquaintance with the symbol sSamplingSpan behavior:

In[260]:=
```
n = 13;
SetOptions[ListPlot, PlotRange → {{-0.2, n - 0.8}, {-n - 0.2, n + 0.2}},
 Axes → True, Frame → {{False, False}, {True, True}}, DataRange → {0, n - 1},
 PlotStyle → Directive[PointSize[Large]], Filling → Axis,
 FillingStyle → Directive[Thickness[0.005]], GridLines → {None, Range[-n, n]},
 LabelStyle → 12, AspectRatio → 1 / 4, ImageSize → Medium];
rothlEx3[u_] := Module[{v = sRotateHalfLength[u], n = Length@u, s},
  s = sSamplingSpan[n];
  If[v =!= $Failed, Column[{
     ListPlot[Labeled[#, #] & /@ u,
      FrameTicks → {{None, None},
        {s, {s, sRotateHalfLength[sSamplingSpan[-n], 1, True &]}^T}},
      PlotStyle → Directive[Lighter[Blue]],
      Prolog → {Black, Line[{{0, -n}, {0, n}}], Line[{{n - 1, -n}, {n - 1, n}}],
        Dashed, Line[{{Floor[n / 2], -n}, {Floor[n / 2], n}}]},
      Epilog → {Black, Text[Style["×", 20], {n - 1, 0}],
        Inset[Framed[Style["del", 10], Background → Pink,
          FrameMargins → Tiny], {0.885 n, 0}]}],
     ListPlot[Labeled[#, #] & /@ v, DataRange → {0, n - 2},
      FrameTicks → {{None, None},
        {{Most@s, sRotateHalfLength[Most@s, True &]}^T,
         {Most@s, sSamplingSpan[-n + 1]}^T}}, PlotStyle → Directive[Brown],
      Prolog → {Black, Line[{{Floor[n / 2], -n}, {Floor[n / 2], n}}],
        Dashed, Line[{{0, -n}, {0, n}}], Line[{{n - 1, -n}, {n - 1, n}}]}]
    }, Alignment → Left, Spacings → {Scaled[0.025], Scaled[0.025]}]]]
```

In[262]:=
```
rothlEx3[sSamplingSpan[n]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

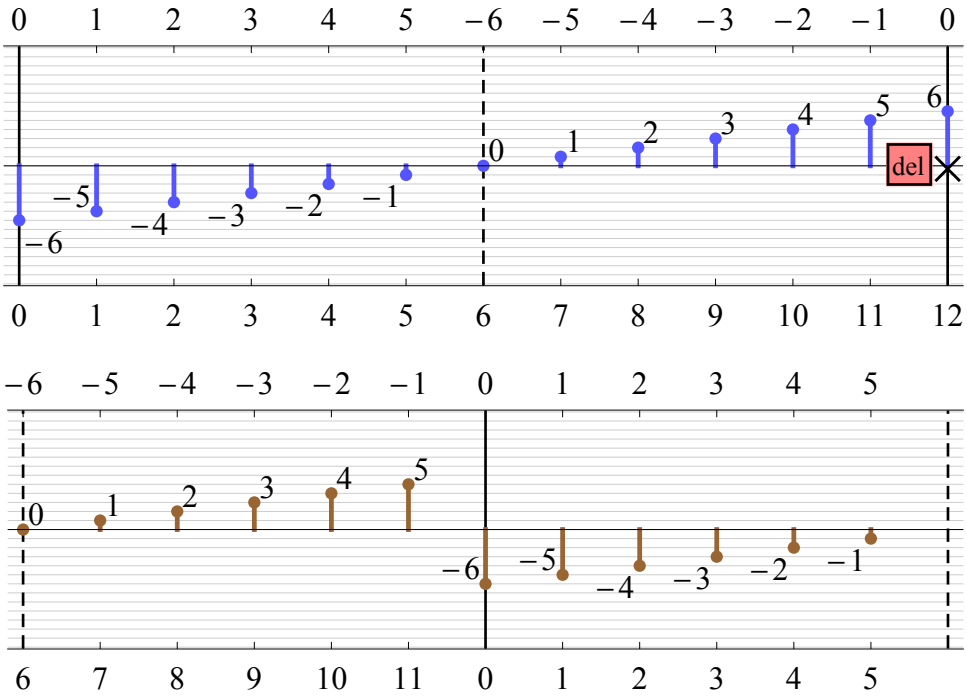
Out[262]=



In[263]:=
```
rothlEx3[sSamplingSpan[-n]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

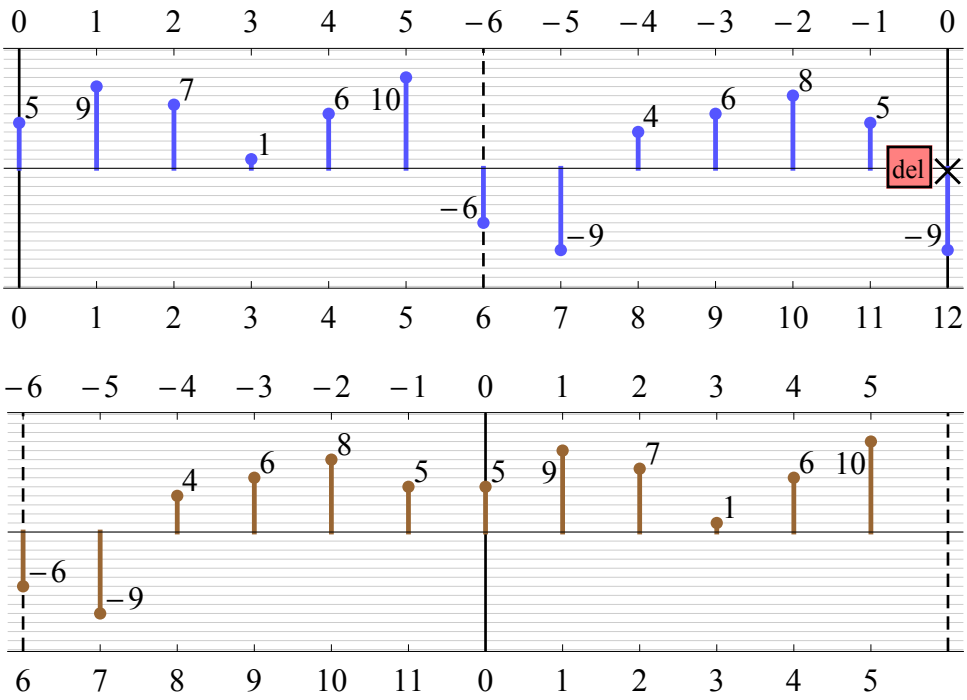
Out[263]=



In[264]:=
```
rothlEx3[RandomInteger[{-n, n}, n]]
```

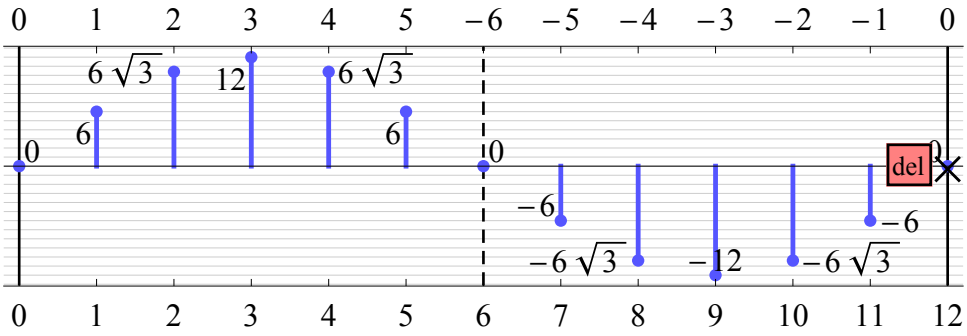
sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

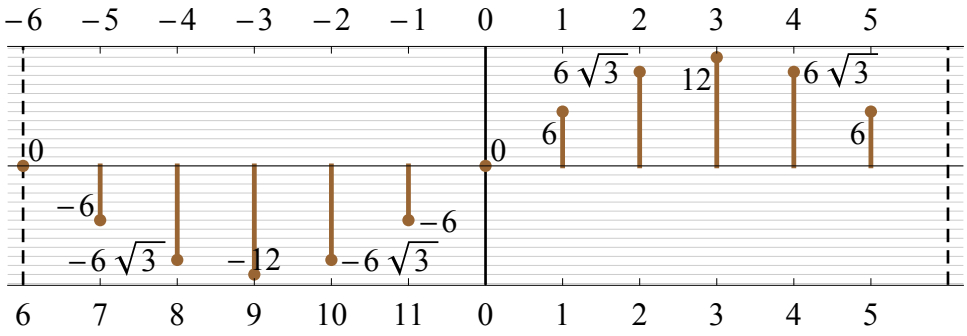
Out[264]=



In[265]:=
```
rothlEx3[(n - 1) Join[Sin[sCircularlySample[{0, 2 π}, n - 1]], {0}]]
```

Out[265]=

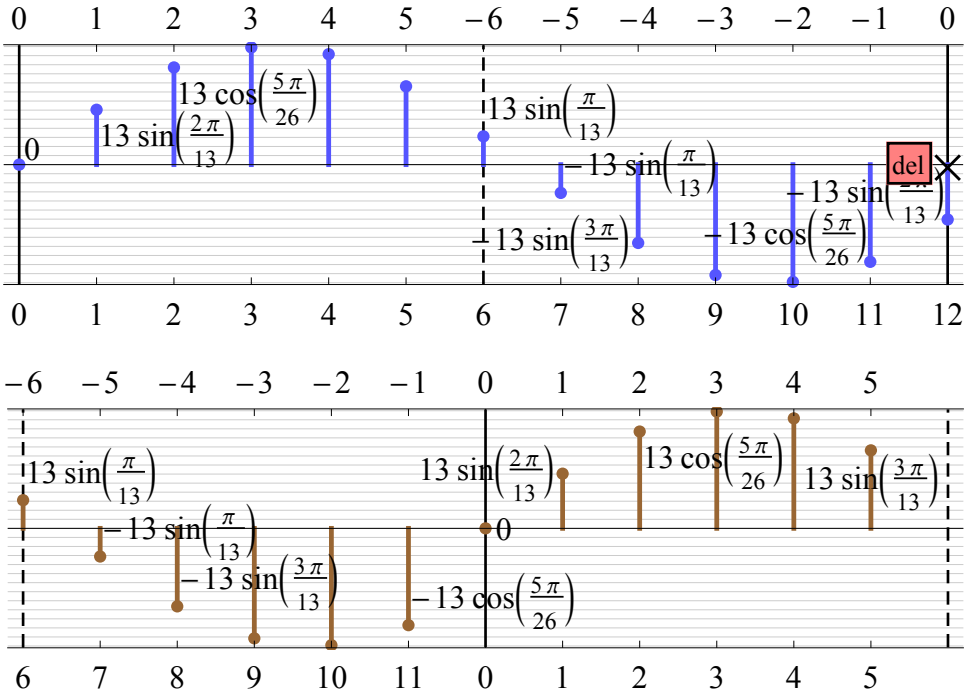




In[266]:=
```
rothlEx3[n Sin[sCircularlySample[{0, 2 π}, n]]]
```

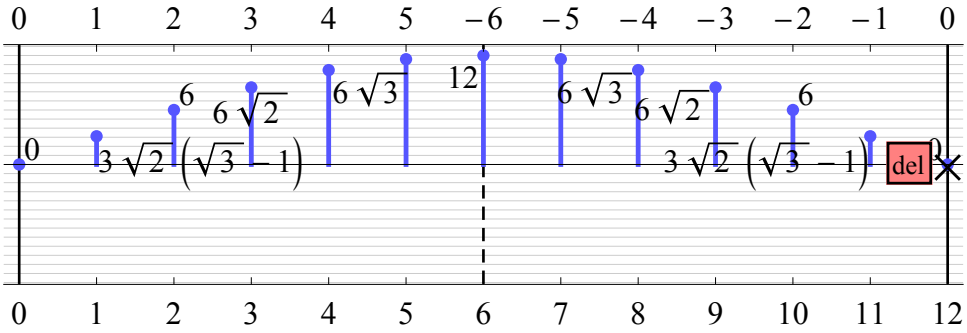
sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

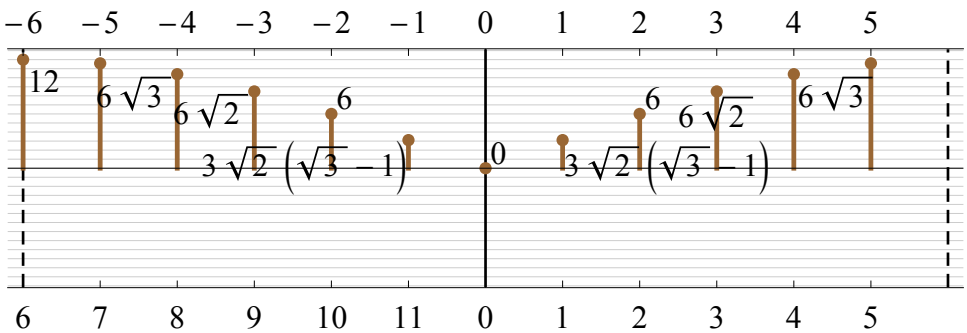
Out[266]=



In[267]:=
```
rothlEx3[Join[(n - 1) Sin[sCircularlySample[{0, π}, n - 1]], {0}]]
```

Out[267]=

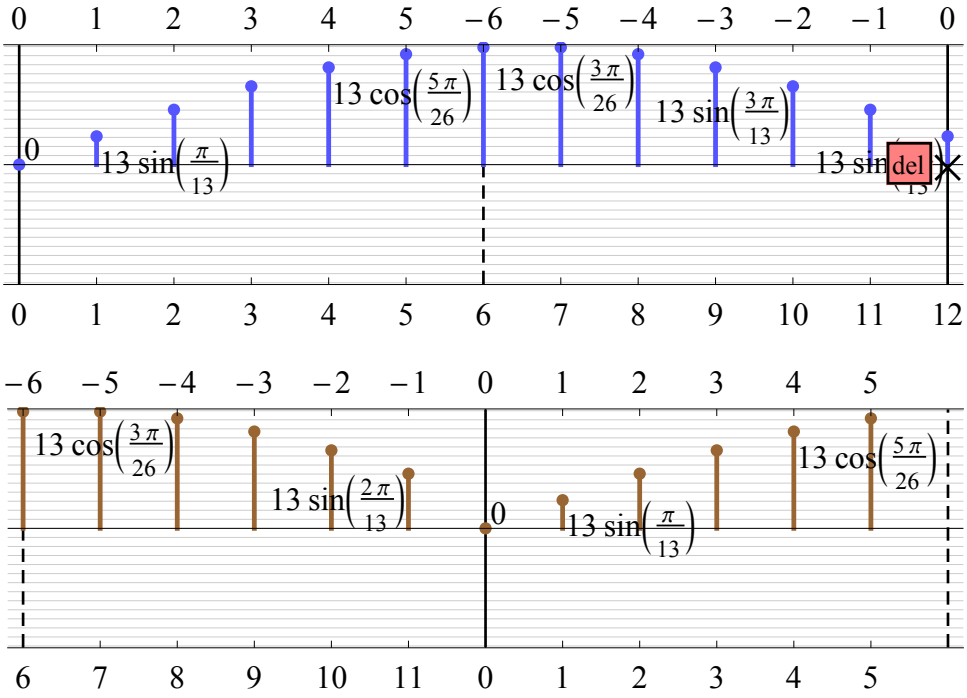




In[268]:=
```
rothlEx3[n Sin[sCircularlySample[{0, π}, n]]]
```

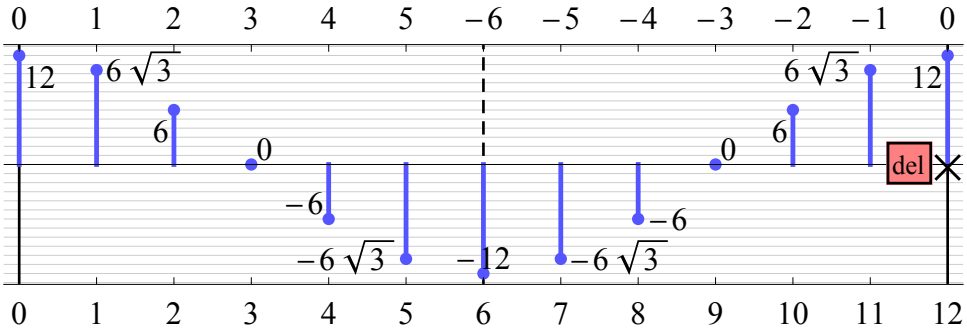
sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

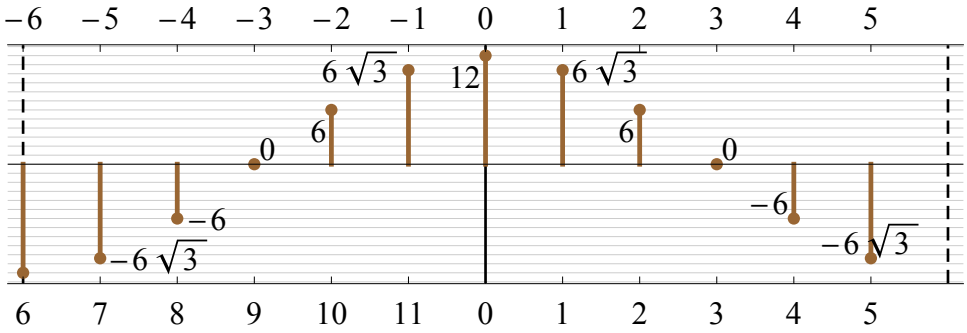
Out[268]=



In[269]:=
```
rothlEx3[Join[(n - 1) Cos[sCircularlySample[{0, 2 π}, n - 1]], {n - 1}]]
```

Out[269]=

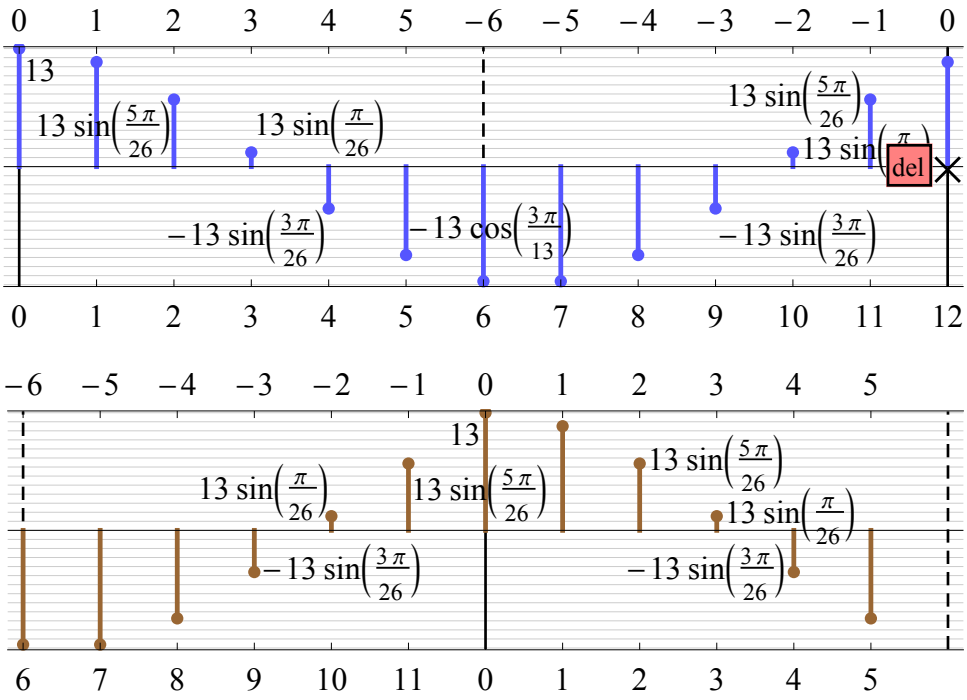




In[270]:=
```
rothlEx3[n Cos[sCircularlySample[{0, 2 π}, n]]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

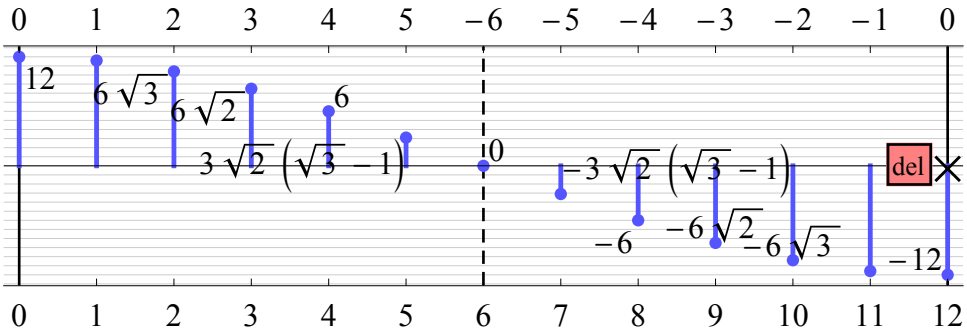
Out[270]=

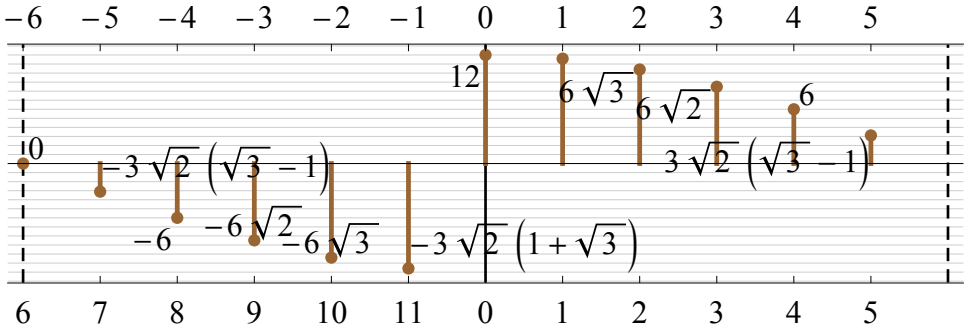


In[271]:=
```
rothlEx3[Join[(n - 1) Cos[sCircularlySample[{0, π}, n - 1]], {-n + 1}]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

Out[271]=

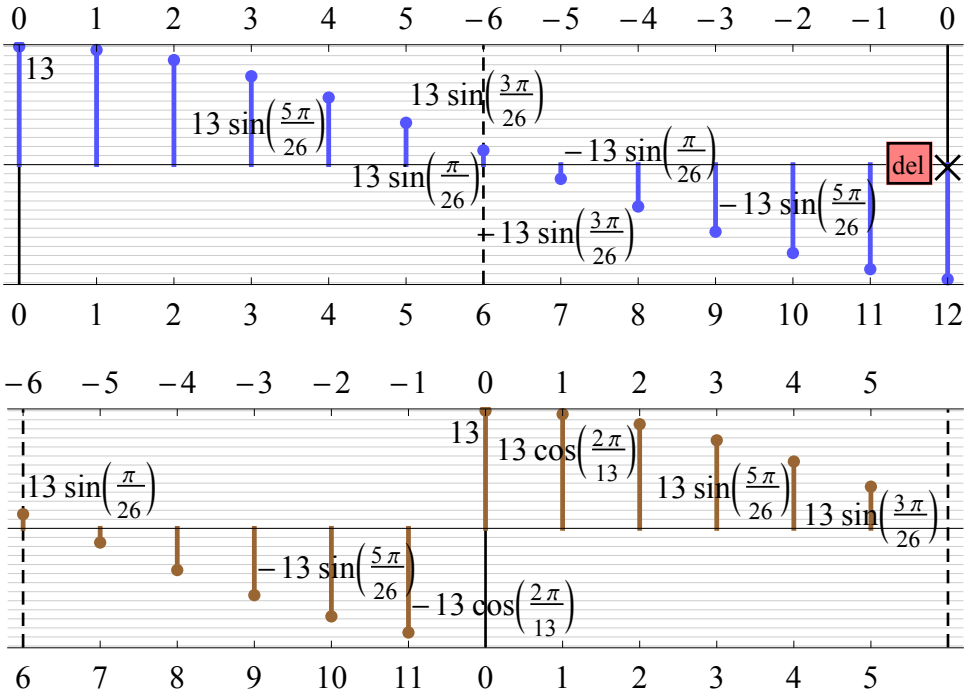




In[272]:=
```
rothlEx3[n Cos[sCircularlySample[{0, π}, n]]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

Out[272]=



Odd length, duplicating

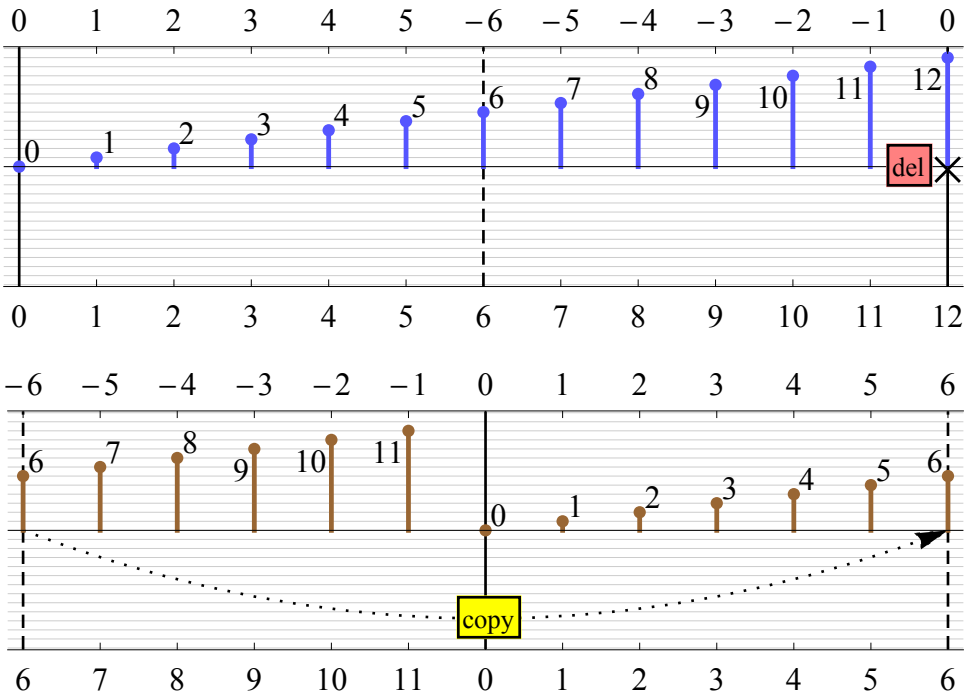
Finally we will demonstrate the behavior with a vector of odd length, but calling sRotateHalfLength[u, 1 | True], hence the first element of the rotated list is duplicated on its right end. Let us define a helper function, and visit Section 2.1 to make acquaintance with the symbol sSamplingSpan behavior:

In[273]:=
```
n = 13;
rothlEx4[u_] := Module[{v = sRotateHalfLength[u, 1], n = Length@u, s},
  s = sSamplingSpan[n];
  If[v =!= $Failed, Column[{
     ListPlot[Labeled[#, #] & /@ u,
      FrameTicks → {{None, None},
        {s, {s, sRotateHalfLength[sSamplingSpan[-n], 1, True &]}^T}},
      PlotStyle → Directive[Lighter[Blue]],
      Prolog → {Black, Line[{{0, -n}, {0, n}}], Line[{{n - 1, -n}, {n - 1, n}}],
        Dashed, Line[{{Floor[n / 2], -n}, {Floor[n / 2], n}}]},
      Epilog → {Black, Text[Style["×", 20], {n - 1, 0}],
        Inset[Framed[Style["del", 10], Background → Pink,
          FrameMargins → Tiny], {0.885 n, 0}]}],
     ListPlot[Labeled[#, #] & /@ v, DataRange → {0, n - 1},
      FrameTicks → {{None, None}, {{s, sRotateHalfLength[s, 1, True &]}^T,
         {s, sSamplingSpan[-n]}^T}}, PlotStyle → Directive[Brown],
      Prolog → {Black, Line[{{Floor[n / 2], -n}, {Floor[n / 2], n}}],
        Dashed, Line[{{0, -n}, {0, n}}], Line[{{n - 1, -n}, {n - 1, n}}],
        Dotted, Arrowheads[Medium],
        Arrow@BezierCurve[{{0, 0}, {(n - 1) / 2, -1.5 n}, {n - 1, 0}}]},
      Epilog →
       {Inset[Framed[Style["copy", 10], Background → Yellow,
          FrameMargins → Tiny], {0.465 n, -3 n / 4}]}]
    }, Alignment → Left, Spacings → {Scaled[0.025], Scaled[0.025]}]]]
```

In[275]:=
```
rothlEx4[sSamplingSpan[n]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

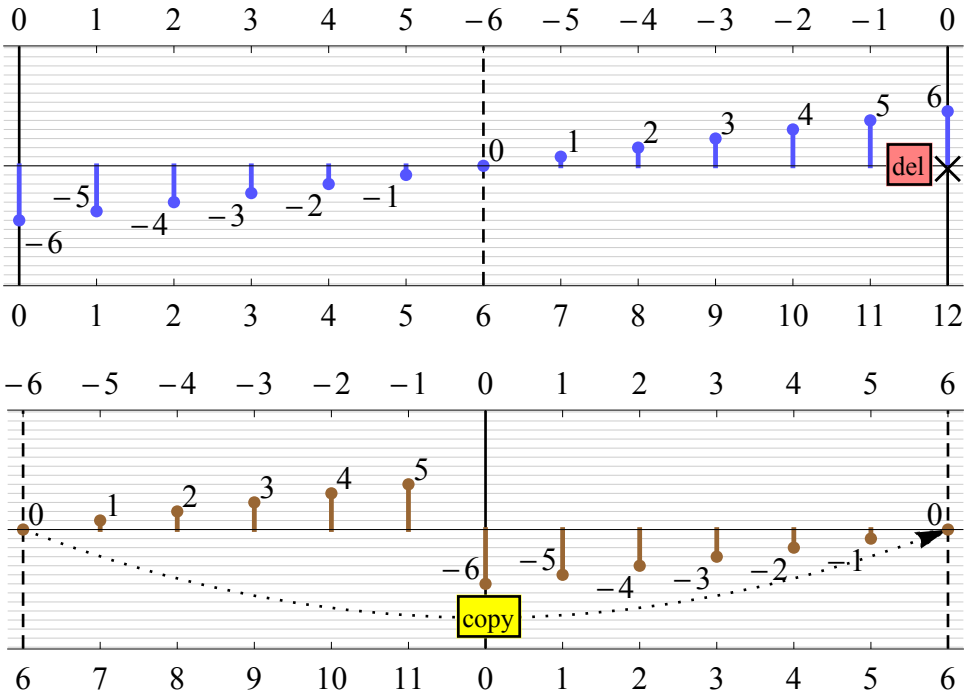
Out[275]=



In[276]:=
```
rothlEx4[sSamplingSpan[-n]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

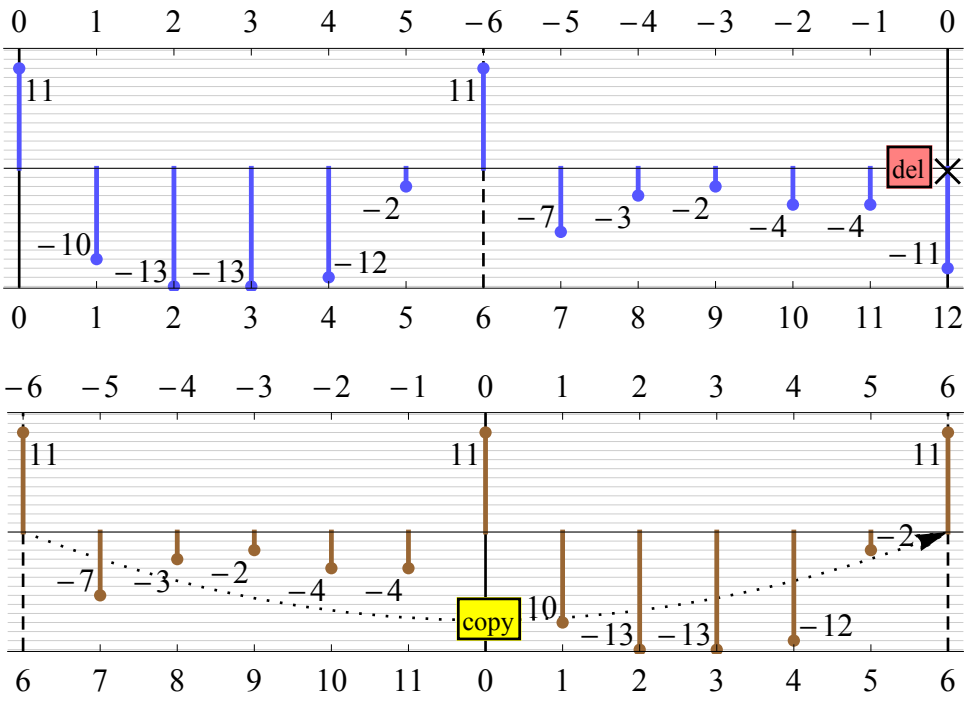
Out[276]=



In[277]:=
```
rothlEx4[RandomInteger[{-n, n}, n]]
```

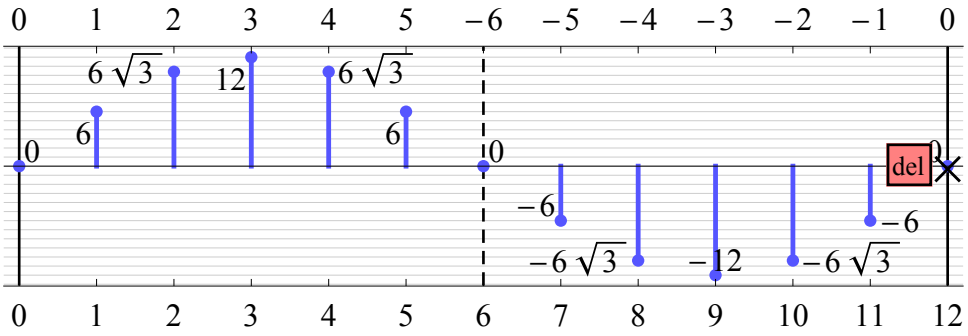
sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

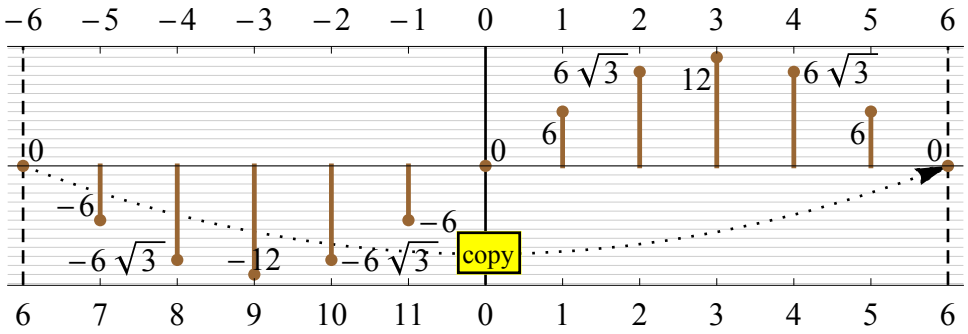
Out[277]=



In[278]:=
```
rothlEx4[(n - 1) Join[Sin[sCircularlySample[{0, 2 π}, n - 1]], {0}]]
```

Out[278]=

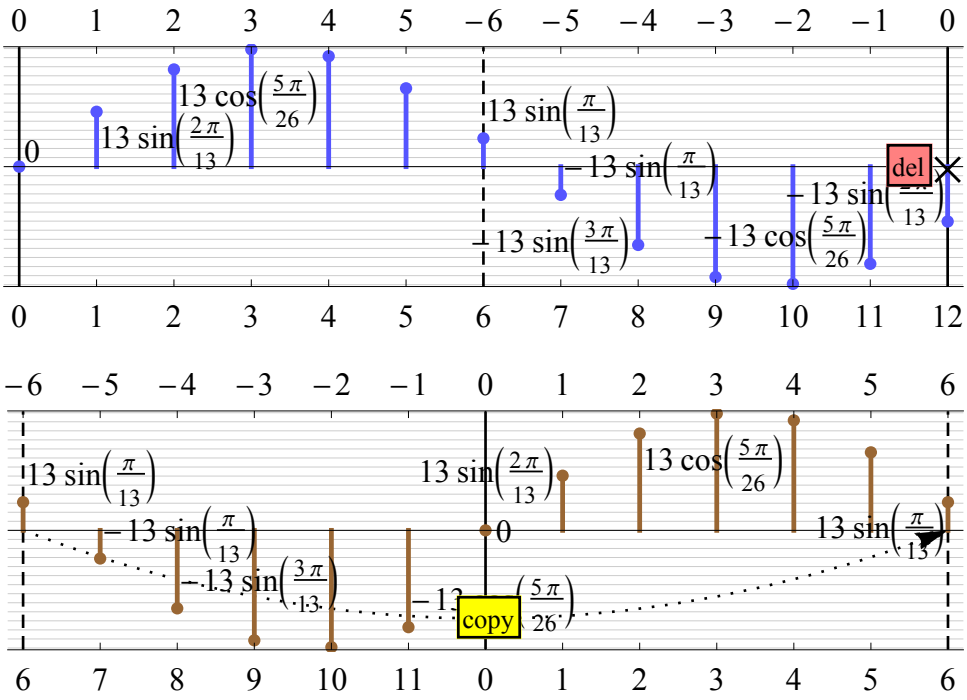




In[279]:=
```
rothlEx4[n Sin[sCircularlySample[{0, 2 π}, n]]]
```

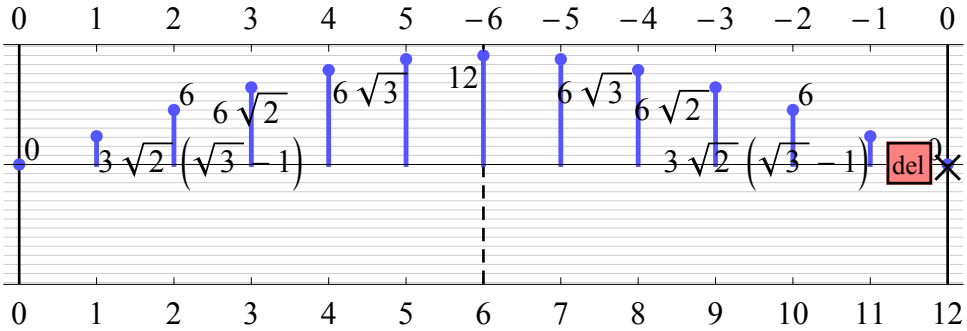
sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

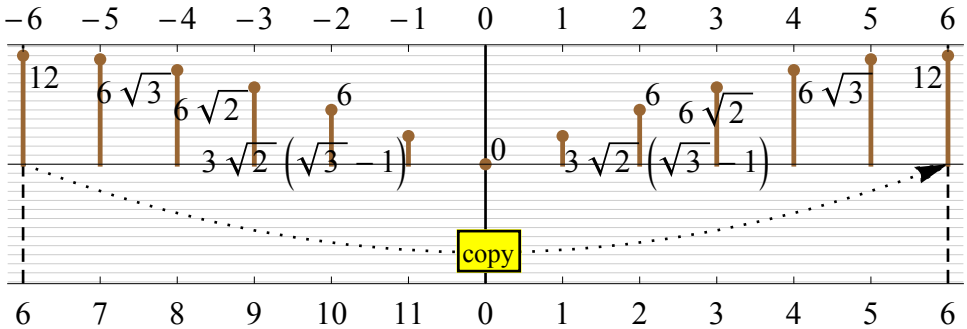
Out[279]=



In[280]:=
```
rothlEx4[Join[(n - 1) Sin[sCircularlySample[{0, π}, n - 1]], {0}]]
```

Out[280]=

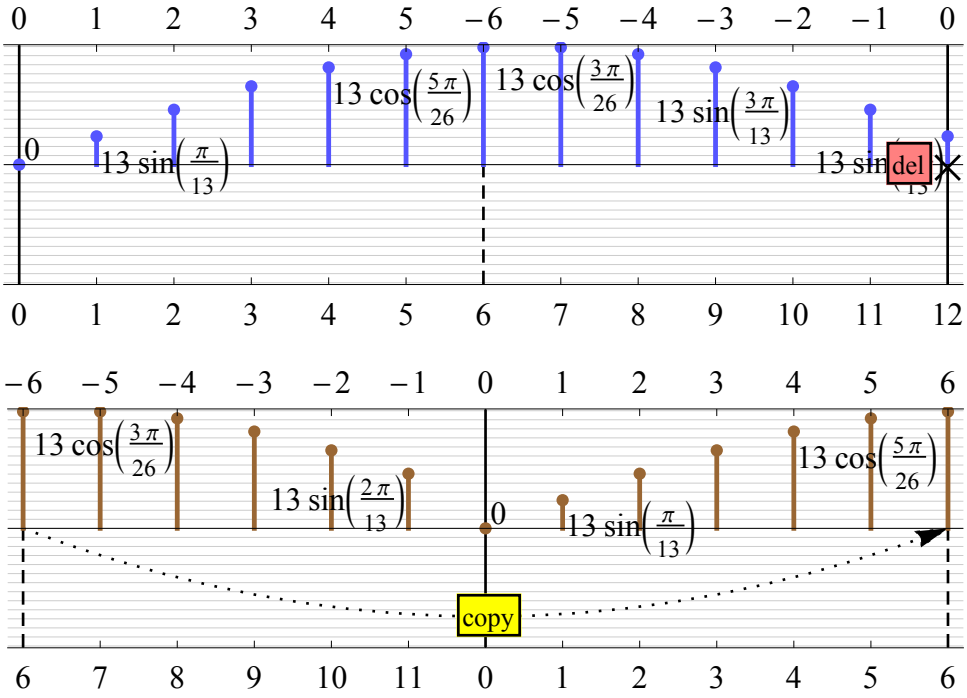




In[281]:=
```
rothlEx4[n Sin[sCircularlySample[{0, π}, n]]]
```

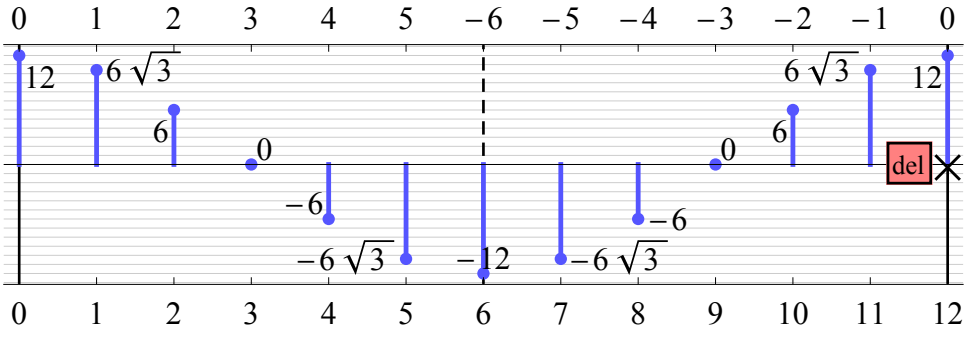
sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

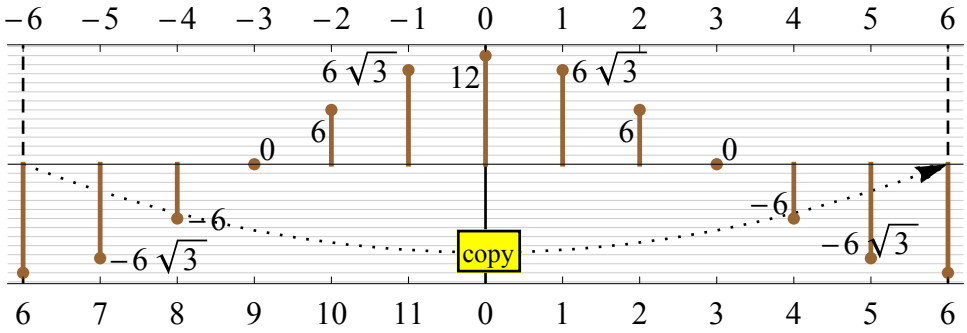
Out[281]=



In[282]:=
```
rothlEx4[Join[(n - 1) Cos[sCircularlySample[{0, 2 π}, n - 1]], {n - 1}]]
```

Out[282]=

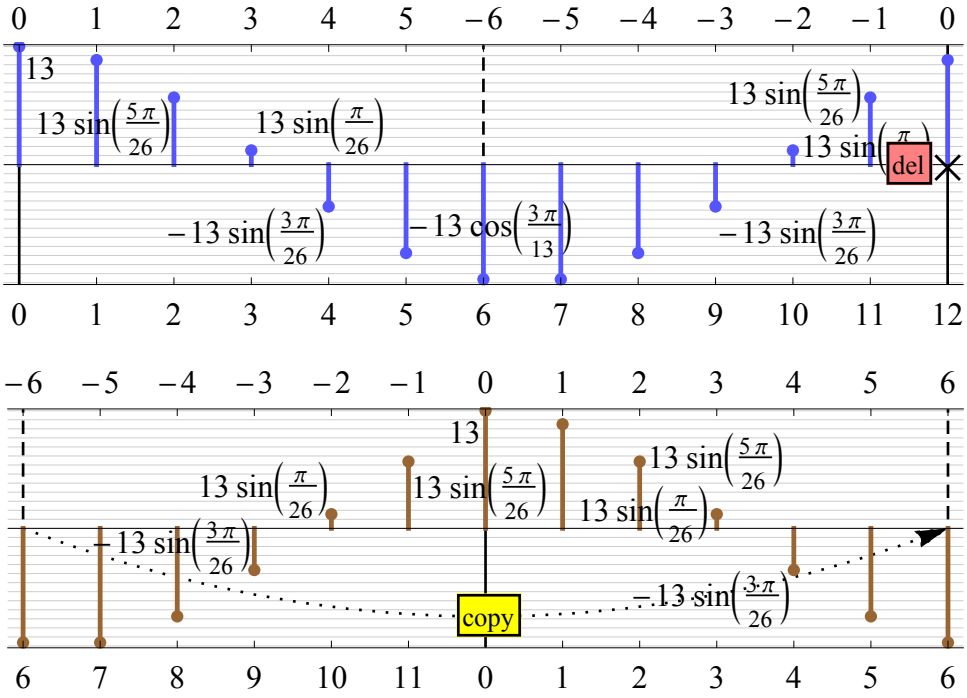




In[283]:=
```
rothlEx4[n Cos[sCircularlySample[{0, 2 π}, n]]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

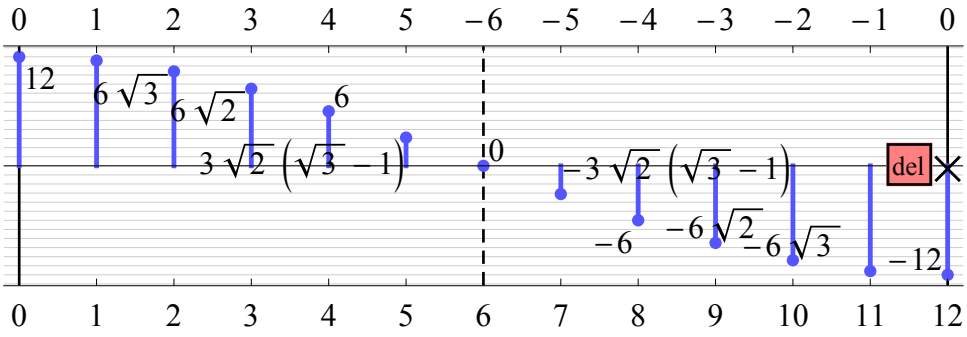
Out[283]=

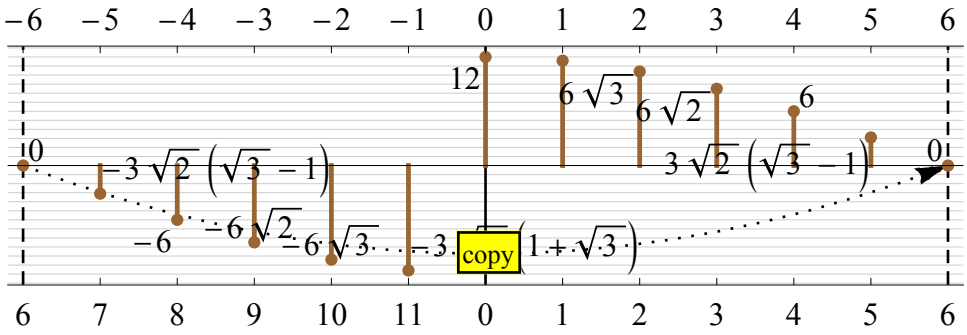


In[284]:=
```
rothlEx4[Join[(n - 1) Cos[sCircularlySample[{0, π}, n - 1]], {-n + 1}]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

Out[284]=

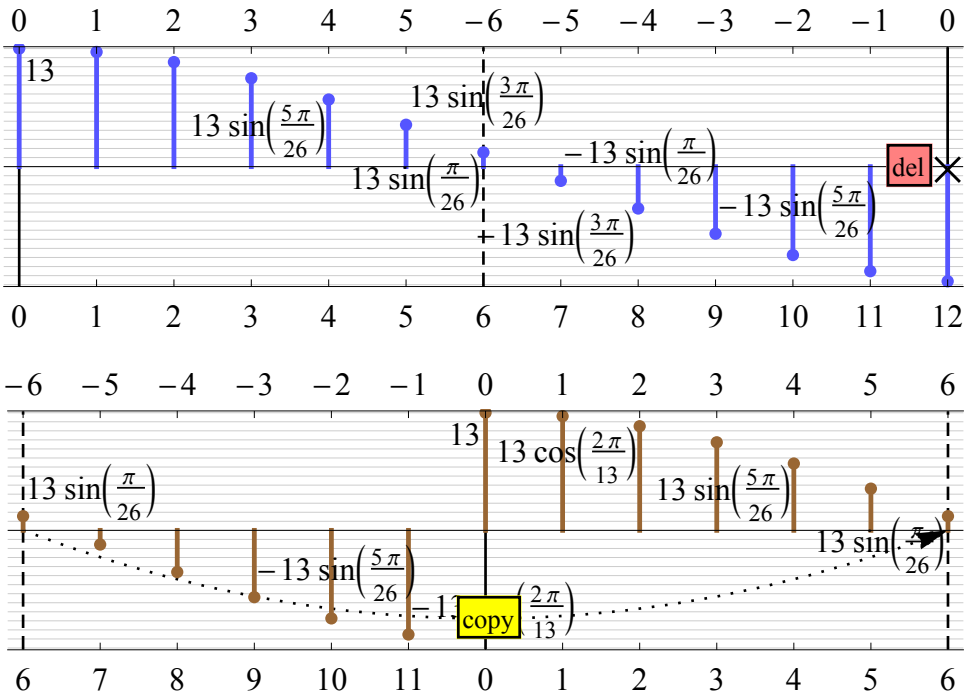




In[285]:=
```
rothlEx4[n Cos[sCircularlySample[{0, π}, n]]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

Out[285]=



Now we can restore the ListPlot options setting to its original value:

In[286]:=
```
SetOptions[ListPlot, Sequence @@ optsListPlot];
```

Testing relationship of edge elements relationship

The test defaults to SameQ:

In[287]:=
```
With[{u = Range[-2, 2, 1]}, sRotateHalfLength[u]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

Out[287]=
```
{0, 1, -2, -1}
```

We can change the test to meet our needs:

In[288]:=
```
With[{u = Range[-2, 2, 1]}, sRotateHalfLength[u, #1 == #2 &]]
```

sRotateHalfLength: NOTE: edge elements failed to pass test #1 == #2 &.

Out[288]=
```
{0, 1, -2, -1}
```

This eliminates testing completely:

In[289]:=
```
With[{u = Range[-2, 2, 1]}, sRotateHalfLength[u, True &]]
```

Out[289]=
```
{0, 1, -2, -1}
```

And this is (useless) example of how to force the test to be always positive even if the first and the last are aliased:

```
In[290]:= sRotateHalfLength[{0, 1, 2, 3, 2, 1, 0}]
          sRotateHalfLength[{0, 1, 2, 3, 2, 1, 0}, False &]

Out[290]= {3, 2, 1, 0, 1, 2}

          sRotateHalfLength: NOTE: edge elements failed to pass test False &.

Out[291]= {3, 2, 1, 0, 1, 2}
```

This test may come handy when numerical inaccuracies are on the scene:

```
In[292]:= sRotateHalfLength[{0, 1, 2, 3, 2, 1, 10^-12}]

          sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

Out[292]= {3, 2, 1, 0, 1, 2}

In[293]:= sRotateHalfLength[{0, 1, 2, 3, 2, 1, 10^-12}, RealAbs[#1 - #2] < 10^-8 &]

Out[293]= {3, 2, 1, 0, 1, 2}
```

Analogical situations when treating complex conjugates may be useful:

```
In[294]:= sRotateHalfLength[{1 + i, 2 + 2 i, 3, 2 - 2 i, 1 - i}]

          sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

Out[294]= {3, 2 - 2 i, 1 + i, 2 + 2 i}

In[295]:= sRotateHalfLength[{1 + i, 2 + 2 i, 3, 2 - 2 i, 1 - i}, Abs[#1] == Abs[#2] &]

Out[295]= {3, 2 - 2 i, 1 + i, 2 + 2 i}

In[296]:= sRotateHalfLength[{1 + i, 2 + 2 i, 3, 2 - 2 i, 1 - i}, Abs[#1] === Abs[#2] &]

Out[296]= {3, 2 - 2 i, 1 + i, 2 + 2 i}

In[297]:= sRotateHalfLength[{1 + i, 2 + 2 i, 3, 2 - 2 i, 1 - i}, Conjugate[#1] == #2 &]

Out[297]= {3, 2 - 2 i, 1 + i, 2 + 2 i}

In[298]:= sRotateHalfLength[{1 + i, 2 + 2 i, 3, 2 - 2 i, 1 - i}, Conjugate[#1] === #2 &]

Out[298]= {3, 2 - 2 i, 1 + i, 2 + 2 i}
```

Also numerical artifacts may be treating by specifying a suitable test:

```
In[299]:= sRotateHalfLength[{{0, 0}, {1, 1}, {2, 2}, {3, 3}, {2, 2}, {1, 1},
            {10^-12, 10^-12}}]

          sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

Out[299]= {{3, 3}, {2, 2}, {1, 1}, {0, 0}, {1, 1}, {2, 2}}
```

```
In[300]:= sRotateHalfLength[{{0, 0}, {1, 1}, {2, 2}, {3, 3}, {2, 2}, {1, 1},
            {10^-12, 10^-12}}, Norm[#1 - #2] < 10^-8 &]

Out[300]= {{3, 3}, {2, 2}, {1, 1}, {0, 0}, {1, 1}, {2, 2}}
```

2D arrays (matrices)

First we will examine square matrices of both even and odd sizes. Let us generate two test matrices, an even-size and an odd-size one:

```
In[301]:= {u, v} = {Array[10 #1 + #2 &, {4, 4}], Array[10 #1 + #2 &, {5, 5}]};
          MatrixForm /@ {u, v} // Row[#, Spacer[8]] &
```

Out[302]=

$$\begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{pmatrix} \quad \begin{pmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{pmatrix}$$

Row operations

```
In[303]:= MatrixForm /@ sRotateHalfLength /@ {u, v} // Row[#, Spacer[8]] &

          sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.
```

Out[303]=

$$\begin{pmatrix} 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \\ 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \end{pmatrix} \quad \begin{pmatrix} 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \end{pmatrix}$$

```
In[304]:= MatrixForm /@ (sRotateHalfLength[#, 1] & /@ {u, v}) // Row[#, Spacer[8]] &

          sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.
```

Out[304]=

$$\begin{pmatrix} 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \\ 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \end{pmatrix} \quad \begin{pmatrix} 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \end{pmatrix}$$

Column operations

In the following, unless it is on purpose, we suppress the warning message using Quiet:

```
In[305]:= MatrixForm /@ Map[sRotateHalfLength, {u, v}, {2}] // Row[#, Spacer[8]] & //
           Quiet
```

Out[305]=

$$\begin{pmatrix} 13 & 14 & 11 & 12 \\ 23 & 24 & 21 & 22 \\ 33 & 34 & 31 & 32 \\ 43 & 44 & 41 & 42 \end{pmatrix} \quad \begin{pmatrix} 13 & 14 & 11 & 12 \\ 23 & 24 & 21 & 22 \\ 33 & 34 & 31 & 32 \\ 43 & 44 & 41 & 42 \\ 53 & 54 & 51 & 52 \end{pmatrix}$$

In[306]:=
```
MatrixForm /@ Map[sRotateHalfLength[#, 1] &, {u, v}, {2}] //
  Row[#, Spacer[8]] & // Quiet
```

Out[306]=

$$\begin{pmatrix} 13 & 14 & 11 & 12 & 13 \\ 23 & 24 & 21 & 22 & 23 \\ 33 & 34 & 31 & 32 & 33 \\ 43 & 44 & 41 & 42 & 43 \end{pmatrix} \quad \begin{pmatrix} 13 & 14 & 11 & 12 & 13 \\ 23 & 24 & 21 & 22 & 23 \\ 33 & 34 & 31 & 32 & 33 \\ 43 & 44 & 41 & 42 & 43 \\ 53 & 54 & 51 & 52 & 53 \end{pmatrix}$$

Row and column operations

In[307]:=
```
MatrixForm /@ Map[sRotateHalfLength, {u, v}, 2] // Row[#, Spacer[8]] & //
  Quiet
```

Out[307]=

$$\begin{pmatrix} 33 & 34 & 31 & 32 \\ 43 & 44 & 41 & 42 \\ 13 & 14 & 11 & 12 \\ 23 & 24 & 21 & 22 \end{pmatrix} \quad \begin{pmatrix} 33 & 34 & 31 & 32 \\ 43 & 44 & 41 & 42 \\ 13 & 14 & 11 & 12 \\ 23 & 24 & 21 & 22 \end{pmatrix}$$

In[308]:=
```
MatrixForm /@ Map[sRotateHalfLength[#, 1] &, {u, v}, 2] //
  Row[#, Spacer[8]] & // Quiet
```

Out[308]=

$$\begin{pmatrix} 33 & 34 & 31 & 32 & 33 \\ 43 & 44 & 41 & 42 & 43 \\ 13 & 14 & 11 & 12 & 13 \\ 23 & 24 & 21 & 22 & 23 \\ 33 & 34 & 31 & 32 & 33 \end{pmatrix} \quad \begin{pmatrix} 33 & 34 & 31 & 32 & 33 \\ 43 & 44 & 41 & 42 & 43 \\ 13 & 14 & 11 & 12 & 13 \\ 23 & 24 & 21 & 22 & 23 \\ 33 & 34 & 31 & 32 & 33 \end{pmatrix}$$

The "Fold idiom" allows easily appending aliased element for each level separately (warning messages are eliminated using the True & in place of the test):

In[309]:=
```
Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &], #1, {#2[[1]]}] &,
  u, {Range[0, ArrayDepth[u] - 1], {0, 1} (*r,c*)}^T] // MatrixForm
```

Out[309]//MatrixForm=

$$\begin{pmatrix} 33 & 34 & 31 & 32 & 33 \\ 43 & 44 & 41 & 42 & 43 \\ 13 & 14 & 11 & 12 & 13 \\ 23 & 24 & 21 & 22 & 23 \end{pmatrix}$$

In[310]:=
```
Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &], #1, {#2[[1]]}] &,
  u, {Range[0, ArrayDepth[u] - 1], {1, 0} (*r,c*)}^T] // MatrixForm
```

Out[310]//MatrixForm=

$$\begin{pmatrix} 33 & 34 & 31 & 32 \\ 43 & 44 & 41 & 42 \\ 13 & 14 & 11 & 12 \\ 23 & 24 & 21 & 22 \\ 33 & 34 & 31 & 32 \end{pmatrix}$$

In[311]:=
```
Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &], #1, {#2[[1]]}] &,
  u, {Range[0, ArrayDepth[u] - 1], {1, 1}(*r,c*)}^T] // MatrixForm
```

Out[311]//MatrixForm=

$$\begin{pmatrix} 33 & 34 & 31 & 32 & 33 \\ 43 & 44 & 41 & 42 & 43 \\ 13 & 14 & 11 & 12 & 13 \\ 23 & 24 & 21 & 22 & 23 \\ 33 & 34 & 31 & 32 & 33 \end{pmatrix}$$

In[312]:=
```
Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &], #1, {#2[[1]]}] &,
  v, {Range[0, ArrayDepth[v] - 1], {0, 1}(*r,c*)}^T] // MatrixForm
```

Out[312]//MatrixForm=

$$\begin{pmatrix} 33 & 34 & 31 & 32 & 33 \\ 43 & 44 & 41 & 42 & 43 \\ 13 & 14 & 11 & 12 & 13 \\ 23 & 24 & 21 & 22 & 23 \end{pmatrix}$$

In[313]:=
```
Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &], #1, {#2[[1]]}] &,
  v, {Range[0, ArrayDepth[v] - 1], {1, 0}(*r,c*)}^T] // MatrixForm
```

Out[313]//MatrixForm=

$$\begin{pmatrix} 33 & 34 & 31 & 32 \\ 43 & 44 & 41 & 42 \\ 13 & 14 & 11 & 12 \\ 23 & 24 & 21 & 22 \\ 33 & 34 & 31 & 32 \end{pmatrix}$$

In[314]:=
```
Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &], #1, {#2[[1]]}] &,
  v, {Range[0, ArrayDepth[v] - 1], {1, 0}(*r,c*)}^T] // MatrixForm
```

Out[314]//MatrixForm=

$$\begin{pmatrix} 33 & 34 & 31 & 32 \\ 43 & 44 & 41 & 42 \\ 13 & 14 & 11 & 12 \\ 23 & 24 & 21 & 22 \\ 33 & 34 & 31 & 32 \end{pmatrix}$$

3D arrays (spatial matrices)

First we will examine cube 3D matrices of both even and odd sizes. Let us generate two test 3D matrices, an even-size and an odd-size one:

In[315]:=
```
{u, v} = {Array[100 #1 + 10 #2 + #3 &, {3, 3, 3}],
   Array[100 #1 + 10 #2 + #3 &, {4, 4, 4}]};
MatrixForm /@ {u, v} // Row[#, Spacer[8]] &
```

Out[316]=

$$\begin{pmatrix} \begin{pmatrix} 111 \\ 112 \\ 113 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \end{pmatrix} & \begin{pmatrix} 131 \\ 132 \\ 133 \end{pmatrix} \\ \begin{pmatrix} 211 \\ 212 \\ 213 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \end{pmatrix} & \begin{pmatrix} 231 \\ 232 \\ 233 \end{pmatrix} \\ \begin{pmatrix} 311 \\ 312 \\ 313 \end{pmatrix} & \begin{pmatrix} 321 \\ 322 \\ 323 \end{pmatrix} & \begin{pmatrix} 331 \\ 332 \\ 333 \end{pmatrix} \end{pmatrix} \quad \begin{pmatrix} \begin{pmatrix} 111 \\ 112 \\ 113 \\ 114 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \\ 124 \end{pmatrix} & \begin{pmatrix} 131 \\ 132 \\ 133 \\ 134 \end{pmatrix} & \begin{pmatrix} 141 \\ 142 \\ 143 \\ 144 \end{pmatrix} \\ \begin{pmatrix} 211 \\ 212 \\ 213 \\ 214 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \\ 224 \end{pmatrix} & \begin{pmatrix} 231 \\ 232 \\ 233 \\ 234 \end{pmatrix} & \begin{pmatrix} 241 \\ 242 \\ 243 \\ 244 \end{pmatrix} \\ \begin{pmatrix} 311 \\ 312 \\ 313 \\ 314 \end{pmatrix} & \begin{pmatrix} 321 \\ 322 \\ 323 \\ 324 \end{pmatrix} & \begin{pmatrix} 331 \\ 332 \\ 333 \\ 334 \end{pmatrix} & \begin{pmatrix} 341 \\ 342 \\ 343 \\ 344 \end{pmatrix} \\ \begin{pmatrix} 411 \\ 412 \\ 413 \\ 414 \end{pmatrix} & \begin{pmatrix} 421 \\ 422 \\ 423 \\ 424 \end{pmatrix} & \begin{pmatrix} 431 \\ 432 \\ 433 \\ 434 \end{pmatrix} & \begin{pmatrix} 441 \\ 442 \\ 443 \\ 444 \end{pmatrix} \end{pmatrix}$$

Row operations

In the following, unless it is on purpose, we suppress the warning message using Quiet:

In[317]:=
```
MatrixForm /@ sRotateHalfLength /@ {u, v} // Row[#, Spacer[8]] & //
 Quiet
```

Out[317]=

$$\begin{pmatrix} \begin{pmatrix} 211 \\ 212 \\ 213 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \end{pmatrix} & \begin{pmatrix} 231 \\ 232 \\ 233 \end{pmatrix} \\ \begin{pmatrix} 111 \\ 112 \\ 113 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \end{pmatrix} & \begin{pmatrix} 131 \\ 132 \\ 133 \end{pmatrix} \end{pmatrix} \quad \begin{pmatrix} \begin{pmatrix} 311 \\ 312 \\ 313 \\ 314 \end{pmatrix} & \begin{pmatrix} 321 \\ 322 \\ 323 \\ 324 \end{pmatrix} & \begin{pmatrix} 331 \\ 332 \\ 333 \\ 334 \end{pmatrix} & \begin{pmatrix} 341 \\ 342 \\ 343 \\ 344 \end{pmatrix} \\ \begin{pmatrix} 411 \\ 412 \\ 413 \\ 414 \end{pmatrix} & \begin{pmatrix} 421 \\ 422 \\ 423 \\ 424 \end{pmatrix} & \begin{pmatrix} 431 \\ 432 \\ 433 \\ 434 \end{pmatrix} & \begin{pmatrix} 441 \\ 442 \\ 443 \\ 444 \end{pmatrix} \\ \begin{pmatrix} 111 \\ 112 \\ 113 \\ 114 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \\ 124 \end{pmatrix} & \begin{pmatrix} 131 \\ 132 \\ 133 \\ 134 \end{pmatrix} & \begin{pmatrix} 141 \\ 142 \\ 143 \\ 144 \end{pmatrix} \\ \begin{pmatrix} 211 \\ 212 \\ 213 \\ 214 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \\ 224 \end{pmatrix} & \begin{pmatrix} 231 \\ 232 \\ 233 \\ 234 \end{pmatrix} & \begin{pmatrix} 241 \\ 242 \\ 243 \\ 244 \end{pmatrix} \end{pmatrix}$$

In[318]:=
```
MatrixForm /@ (sRotateHalfLength[#, 1] & /@ {u, v}) // Row[#, Spacer[8]] & //
 Quiet
```

Out[318]=

$$\begin{pmatrix} \begin{pmatrix} 211 \\ 212 \\ 213 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \end{pmatrix} & \begin{pmatrix} 231 \\ 232 \\ 233 \end{pmatrix} \\ \begin{pmatrix} 111 \\ 112 \\ 113 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \end{pmatrix} & \begin{pmatrix} 131 \\ 132 \\ 133 \end{pmatrix} \\ \begin{pmatrix} 211 \\ 212 \\ 213 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \end{pmatrix} & \begin{pmatrix} 231 \\ 232 \\ 233 \end{pmatrix} \end{pmatrix} \quad \begin{pmatrix} \begin{pmatrix} 311 \\ 312 \\ 313 \\ 314 \end{pmatrix} & \begin{pmatrix} 321 \\ 322 \\ 323 \\ 324 \end{pmatrix} & \begin{pmatrix} 331 \\ 332 \\ 333 \\ 334 \end{pmatrix} & \begin{pmatrix} 341 \\ 342 \\ 343 \\ 344 \end{pmatrix} \\ \begin{pmatrix} 411 \\ 412 \\ 413 \\ 414 \end{pmatrix} & \begin{pmatrix} 421 \\ 422 \\ 423 \\ 424 \end{pmatrix} & \begin{pmatrix} 431 \\ 432 \\ 433 \\ 434 \end{pmatrix} & \begin{pmatrix} 441 \\ 442 \\ 443 \\ 444 \end{pmatrix} \\ \begin{pmatrix} 111 \\ 112 \\ 113 \\ 114 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \\ 124 \end{pmatrix} & \begin{pmatrix} 131 \\ 132 \\ 133 \\ 134 \end{pmatrix} & \begin{pmatrix} 141 \\ 142 \\ 143 \\ 144 \end{pmatrix} \\ \begin{pmatrix} 211 \\ 212 \\ 213 \\ 214 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \\ 224 \end{pmatrix} & \begin{pmatrix} 231 \\ 232 \\ 233 \\ 234 \end{pmatrix} & \begin{pmatrix} 241 \\ 242 \\ 243 \\ 244 \end{pmatrix} \\ \begin{pmatrix} 311 \\ 312 \\ 313 \\ 314 \end{pmatrix} & \begin{pmatrix} 321 \\ 322 \\ 323 \\ 324 \end{pmatrix} & \begin{pmatrix} 331 \\ 332 \\ 333 \\ 334 \end{pmatrix} & \begin{pmatrix} 341 \\ 342 \\ 343 \\ 344 \end{pmatrix} \end{pmatrix}$$

Column operations

In[319]:=
```
MatrixForm /@ Map[sRotateHalfLength, {u, v}, {2}] // Row[#, Spacer[8]] & //
 Quiet
```

Out[319]=

$$\begin{pmatrix} \begin{pmatrix} 121 \\ 122 \\ 123 \end{pmatrix} & \begin{pmatrix} 111 \\ 112 \\ 113 \end{pmatrix} \\ \begin{pmatrix} 221 \\ 222 \\ 223 \end{pmatrix} & \begin{pmatrix} 211 \\ 212 \\ 213 \end{pmatrix} \\ \begin{pmatrix} 321 \\ 322 \\ 323 \end{pmatrix} & \begin{pmatrix} 311 \\ 312 \\ 313 \end{pmatrix} \end{pmatrix} \quad \begin{pmatrix} \begin{pmatrix} 131 \\ 132 \\ 133 \\ 134 \end{pmatrix} & \begin{pmatrix} 141 \\ 142 \\ 143 \\ 144 \end{pmatrix} & \begin{pmatrix} 111 \\ 112 \\ 113 \\ 114 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \\ 124 \end{pmatrix} \\ \begin{pmatrix} 231 \\ 232 \\ 233 \\ 234 \end{pmatrix} & \begin{pmatrix} 241 \\ 242 \\ 243 \\ 244 \end{pmatrix} & \begin{pmatrix} 211 \\ 212 \\ 213 \\ 214 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \\ 224 \end{pmatrix} \\ \begin{pmatrix} 331 \\ 332 \\ 333 \\ 334 \end{pmatrix} & \begin{pmatrix} 341 \\ 342 \\ 343 \\ 344 \end{pmatrix} & \begin{pmatrix} 311 \\ 312 \\ 313 \\ 314 \end{pmatrix} & \begin{pmatrix} 321 \\ 322 \\ 323 \\ 324 \end{pmatrix} \\ \begin{pmatrix} 431 \\ 432 \\ 433 \\ 434 \end{pmatrix} & \begin{pmatrix} 441 \\ 442 \\ 443 \\ 444 \end{pmatrix} & \begin{pmatrix} 411 \\ 412 \\ 413 \\ 414 \end{pmatrix} & \begin{pmatrix} 421 \\ 422 \\ 423 \\ 424 \end{pmatrix} \end{pmatrix}$$

In[320]:=
```
MatrixForm /@ Map[sRotateHalfLength[#, 1] &, {u, v}, {2}] //
  Row[#, Spacer[8]] & // Quiet
```

Out[320]=

$$\begin{pmatrix} \begin{pmatrix} 121 \\ 122 \\ 123 \end{pmatrix} & \begin{pmatrix} 111 \\ 112 \\ 113 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \end{pmatrix} \\ \begin{pmatrix} 221 \\ 222 \\ 223 \end{pmatrix} & \begin{pmatrix} 211 \\ 212 \\ 213 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \end{pmatrix} \\ \begin{pmatrix} 321 \\ 322 \\ 323 \end{pmatrix} & \begin{pmatrix} 311 \\ 312 \\ 313 \end{pmatrix} & \begin{pmatrix} 321 \\ 322 \\ 323 \end{pmatrix} \end{pmatrix} \quad \begin{pmatrix} \begin{pmatrix} 131 \\ 132 \\ 133 \\ 134 \end{pmatrix} & \begin{pmatrix} 141 \\ 142 \\ 143 \\ 144 \end{pmatrix} & \begin{pmatrix} 111 \\ 112 \\ 113 \\ 114 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \\ 124 \end{pmatrix} & \begin{pmatrix} 131 \\ 132 \\ 133 \\ 134 \end{pmatrix} \\ \begin{pmatrix} 231 \\ 232 \\ 233 \\ 234 \end{pmatrix} & \begin{pmatrix} 241 \\ 242 \\ 243 \\ 244 \end{pmatrix} & \begin{pmatrix} 211 \\ 212 \\ 213 \\ 214 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \\ 224 \end{pmatrix} & \begin{pmatrix} 231 \\ 232 \\ 233 \\ 234 \end{pmatrix} \\ \begin{pmatrix} 331 \\ 332 \\ 333 \\ 334 \end{pmatrix} & \begin{pmatrix} 341 \\ 342 \\ 343 \\ 344 \end{pmatrix} & \begin{pmatrix} 311 \\ 312 \\ 313 \\ 314 \end{pmatrix} & \begin{pmatrix} 321 \\ 322 \\ 323 \\ 324 \end{pmatrix} & \begin{pmatrix} 331 \\ 332 \\ 333 \\ 334 \end{pmatrix} \\ \begin{pmatrix} 431 \\ 432 \\ 433 \\ 434 \end{pmatrix} & \begin{pmatrix} 441 \\ 442 \\ 443 \\ 444 \end{pmatrix} & \begin{pmatrix} 411 \\ 412 \\ 413 \\ 414 \end{pmatrix} & \begin{pmatrix} 421 \\ 422 \\ 423 \\ 424 \end{pmatrix} & \begin{pmatrix} 431 \\ 432 \\ 433 \\ 434 \end{pmatrix} \end{pmatrix}$$

Row and column operations

In[321]:=
```
MatrixForm /@ Map[sRotateHalfLength, {u, v}, 2] // Row[#, Spacer[8]] & //
  Quiet
```

Out[321]=

$$\begin{pmatrix} \begin{pmatrix} 221 \\ 222 \\ 223 \end{pmatrix} & \begin{pmatrix} 211 \\ 212 \\ 213 \end{pmatrix} \\ \begin{pmatrix} 121 \\ 122 \\ 123 \end{pmatrix} & \begin{pmatrix} 111 \\ 112 \\ 113 \end{pmatrix} \end{pmatrix} \quad \begin{pmatrix} \begin{pmatrix} 331 \\ 332 \\ 333 \\ 334 \end{pmatrix} & \begin{pmatrix} 341 \\ 342 \\ 343 \\ 344 \end{pmatrix} & \begin{pmatrix} 311 \\ 312 \\ 313 \\ 314 \end{pmatrix} & \begin{pmatrix} 321 \\ 322 \\ 323 \\ 324 \end{pmatrix} \\ \begin{pmatrix} 431 \\ 432 \\ 433 \\ 434 \end{pmatrix} & \begin{pmatrix} 441 \\ 442 \\ 443 \\ 444 \end{pmatrix} & \begin{pmatrix} 411 \\ 412 \\ 413 \\ 414 \end{pmatrix} & \begin{pmatrix} 421 \\ 422 \\ 423 \\ 424 \end{pmatrix} \\ \begin{pmatrix} 131 \\ 132 \\ 133 \\ 134 \end{pmatrix} & \begin{pmatrix} 141 \\ 142 \\ 143 \\ 144 \end{pmatrix} & \begin{pmatrix} 111 \\ 112 \\ 113 \\ 114 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \\ 123 \\ 124 \end{pmatrix} \\ \begin{pmatrix} 231 \\ 232 \\ 233 \\ 234 \end{pmatrix} & \begin{pmatrix} 241 \\ 242 \\ 243 \\ 244 \end{pmatrix} & \begin{pmatrix} 211 \\ 212 \\ 213 \\ 214 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \\ 223 \\ 224 \end{pmatrix} \end{pmatrix}$$

In[322]:=
```
MatrixForm /@ Map[sRotateHalfLength[#, 1] &, {u, v}, 2] //
  Row[#, Spacer[8]] & // Quiet
```

Out[322]=

$$\begin{pmatrix} \begin{pmatrix}221\\222\\223\end{pmatrix} & \begin{pmatrix}211\\212\\213\end{pmatrix} & \begin{pmatrix}221\\222\\223\end{pmatrix} \\ \begin{pmatrix}121\\122\\123\end{pmatrix} & \begin{pmatrix}111\\112\\113\end{pmatrix} & \begin{pmatrix}121\\122\\123\end{pmatrix} \\ \begin{pmatrix}221\\222\\223\end{pmatrix} & \begin{pmatrix}211\\212\\213\end{pmatrix} & \begin{pmatrix}221\\222\\223\end{pmatrix} \end{pmatrix} \quad \begin{pmatrix} \begin{pmatrix}331\\332\\333\\334\end{pmatrix} & \begin{pmatrix}341\\342\\343\\344\end{pmatrix} & \begin{pmatrix}311\\312\\313\\314\end{pmatrix} & \begin{pmatrix}321\\322\\323\\324\end{pmatrix} & \begin{pmatrix}331\\332\\333\\334\end{pmatrix} \\ \begin{pmatrix}431\\432\\433\\434\end{pmatrix} & \begin{pmatrix}441\\442\\443\\444\end{pmatrix} & \begin{pmatrix}411\\412\\413\\414\end{pmatrix} & \begin{pmatrix}421\\422\\423\\424\end{pmatrix} & \begin{pmatrix}431\\432\\433\\434\end{pmatrix} \\ \begin{pmatrix}131\\132\\133\\134\end{pmatrix} & \begin{pmatrix}141\\142\\143\\144\end{pmatrix} & \begin{pmatrix}111\\112\\113\\114\end{pmatrix} & \begin{pmatrix}121\\122\\123\\124\end{pmatrix} & \begin{pmatrix}131\\132\\133\\134\end{pmatrix} \\ \begin{pmatrix}231\\232\\233\\234\end{pmatrix} & \begin{pmatrix}241\\242\\243\\244\end{pmatrix} & \begin{pmatrix}211\\212\\213\\214\end{pmatrix} & \begin{pmatrix}221\\222\\223\\224\end{pmatrix} & \begin{pmatrix}231\\232\\233\\234\end{pmatrix} \\ \begin{pmatrix}331\\332\\333\\334\end{pmatrix} & \begin{pmatrix}341\\342\\343\\344\end{pmatrix} & \begin{pmatrix}311\\312\\313\\314\end{pmatrix} & \begin{pmatrix}321\\322\\323\\324\end{pmatrix} & \begin{pmatrix}331\\332\\333\\334\end{pmatrix} \end{pmatrix}$$

Row, column and depth (inner column) operations

In[323]:=
```
Map[sRotateHalfLength, Map[sRotateHalfLength, Map[sRotateHalfLength, u, {0}],
  {1}], {2}] // MatrixForm
```

▪▪▪ sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

▪▪▪ sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

▪▪▪ sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

▪▪▪ General: Further output of sRotateHalfLength::noptes will be suppressed during this calculation.

Out[323]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix}222\\221\end{pmatrix} & \begin{pmatrix}212\\211\end{pmatrix} \\ \begin{pmatrix}122\\121\end{pmatrix} & \begin{pmatrix}112\\111\end{pmatrix} \end{pmatrix}$$

In[324]:=
```
Map[sRotateHalfLength,
  Map[sRotateHalfLength, Map[sRotateHalfLength[#, 1] &, u, {0}], {1}], {2}] //
 MatrixForm
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

General: Further output of sRotateHalfLength::noptes will be suppressed during this calculation.

Out[324]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} 222 \\ 221 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \end{pmatrix} \\ \begin{pmatrix} 122 \\ 121 \end{pmatrix} & \begin{pmatrix} 112 \\ 111 \end{pmatrix} \\ \begin{pmatrix} 222 \\ 221 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \end{pmatrix} \end{pmatrix}$$

In[325]:=
```
Map[sRotateHalfLength, Map[sRotateHalfLength[#, 1] &,
   Map[sRotateHalfLength[#, 1] &, u, {0}], {1}], {2}] // MatrixForm
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

General: Further output of sRotateHalfLength::noptes will be suppressed during this calculation.

Out[325]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} 222 \\ 221 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \end{pmatrix} & \begin{pmatrix} 222 \\ 221 \end{pmatrix} \\ \begin{pmatrix} 122 \\ 121 \end{pmatrix} & \begin{pmatrix} 112 \\ 111 \end{pmatrix} & \begin{pmatrix} 122 \\ 121 \end{pmatrix} \\ \begin{pmatrix} 222 \\ 221 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \end{pmatrix} & \begin{pmatrix} 222 \\ 221 \end{pmatrix} \end{pmatrix}$$

In[326]:=
```
Map[sRotateHalfLength[#, 1] &,
  Map[sRotateHalfLength, Map[sRotateHalfLength[#, 1] &, u, {0}], {1}], {2}] //
 MatrixForm
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

General: Further output of sRotateHalfLength::noptes will be suppressed during this calculation.

Out[326]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \\ 212 \end{pmatrix} \\ \begin{pmatrix} 122 \\ 121 \\ 122 \end{pmatrix} & \begin{pmatrix} 112 \\ 111 \\ 112 \end{pmatrix} \\ \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \\ 212 \end{pmatrix} \end{pmatrix}$$

In[327]:=
```
Map[sRotateHalfLength[#, 1] &,
  Map[sRotateHalfLength[#, 1] &, Map[sRotateHalfLength[#, 1] &, u, {0}], {1}],
  {2}] // MatrixForm
```

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

sRotateHalfLength: NOTE: edge elements failed to pass test SameQ.

General: Further output of sRotateHalfLength::noptes will be suppressed during this calculation.

Out[327]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \\ 212 \end{pmatrix} & \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} \\ \begin{pmatrix} 122 \\ 121 \\ 122 \end{pmatrix} & \begin{pmatrix} 112 \\ 111 \\ 112 \end{pmatrix} & \begin{pmatrix} 122 \\ 121 \\ 122 \end{pmatrix} \\ \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \\ 212 \end{pmatrix} & \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} \end{pmatrix}$$

The "Fold idiom" allows easily appending aliased element for each level separately (warning messages are eliminated using the True& in place of the test):

In[328]:=
```
Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &], #1, {#2[[1]]}] &,
  u, {Range[0, ArrayDepth[u] - 1], {0, 1, 1}(*r,c,d*)}^T] // MatrixForm
```

Out[328]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \\ 212 \end{pmatrix} & \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} \\ \begin{pmatrix} 122 \\ 121 \\ 122 \end{pmatrix} & \begin{pmatrix} 112 \\ 111 \\ 112 \end{pmatrix} & \begin{pmatrix} 122 \\ 121 \\ 122 \end{pmatrix} \end{pmatrix}$$

In[329]:=
```
Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &], #1, {#2[[1]]}] &,
  u, {Range[0, ArrayDepth[u] - 1], {1, 0, 1}(*r,c,d*)}^T] // MatrixForm
```

Out[329]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \\ 212 \end{pmatrix} \\ \begin{pmatrix} 122 \\ 121 \\ 122 \end{pmatrix} & \begin{pmatrix} 112 \\ 111 \\ 112 \end{pmatrix} \\ \begin{pmatrix} 222 \\ 221 \\ 222 \end{pmatrix} & \begin{pmatrix} 212 \\ 211 \\ 212 \end{pmatrix} \end{pmatrix}$$

In[330]:=
```
Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &], #1, {#2[[1]]}] &,
  v, {Range[0, ArrayDepth[v] - 1], {1, 1, 1}(*r,c,d*)}^T] // MatrixForm
```

Out[330]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} 333 \\ 334 \\ 331 \\ 332 \\ 333 \end{pmatrix} & \begin{pmatrix} 343 \\ 344 \\ 341 \\ 342 \\ 343 \end{pmatrix} & \begin{pmatrix} 313 \\ 314 \\ 311 \\ 312 \\ 313 \end{pmatrix} & \begin{pmatrix} 323 \\ 324 \\ 321 \\ 322 \\ 323 \end{pmatrix} & \begin{pmatrix} 333 \\ 334 \\ 331 \\ 332 \\ 333 \end{pmatrix} \\ \begin{pmatrix} 433 \\ 434 \\ 431 \\ 432 \\ 433 \end{pmatrix} & \begin{pmatrix} 443 \\ 444 \\ 441 \\ 442 \\ 443 \end{pmatrix} & \begin{pmatrix} 413 \\ 414 \\ 411 \\ 412 \\ 413 \end{pmatrix} & \begin{pmatrix} 423 \\ 424 \\ 421 \\ 422 \\ 423 \end{pmatrix} & \begin{pmatrix} 433 \\ 434 \\ 431 \\ 432 \\ 433 \end{pmatrix} \\ \begin{pmatrix} 133 \\ 134 \\ 131 \\ 132 \\ 133 \end{pmatrix} & \begin{pmatrix} 143 \\ 144 \\ 141 \\ 142 \\ 143 \end{pmatrix} & \begin{pmatrix} 113 \\ 114 \\ 111 \\ 112 \\ 113 \end{pmatrix} & \begin{pmatrix} 123 \\ 124 \\ 121 \\ 122 \\ 123 \end{pmatrix} & \begin{pmatrix} 133 \\ 134 \\ 131 \\ 132 \\ 133 \end{pmatrix} \\ \begin{pmatrix} 233 \\ 234 \\ 231 \\ 232 \\ 233 \end{pmatrix} & \begin{pmatrix} 243 \\ 244 \\ 241 \\ 242 \\ 243 \end{pmatrix} & \begin{pmatrix} 213 \\ 214 \\ 211 \\ 212 \\ 213 \end{pmatrix} & \begin{pmatrix} 223 \\ 224 \\ 221 \\ 222 \\ 223 \end{pmatrix} & \begin{pmatrix} 233 \\ 234 \\ 231 \\ 232 \\ 233 \end{pmatrix} \\ \begin{pmatrix} 333 \\ 334 \\ 331 \\ 332 \\ 333 \end{pmatrix} & \begin{pmatrix} 343 \\ 344 \\ 341 \\ 342 \\ 343 \end{pmatrix} & \begin{pmatrix} 313 \\ 314 \\ 311 \\ 312 \\ 313 \end{pmatrix} & \begin{pmatrix} 323 \\ 324 \\ 321 \\ 322 \\ 323 \end{pmatrix} & \begin{pmatrix} 333 \\ 334 \\ 331 \\ 332 \\ 333 \end{pmatrix} \end{pmatrix}$$

Applications

Spectral analysis application

This allows restoring the options of the built-in symbol ListPlot to the original state:

```
In[331]:= optsListPlot = Options[ListPlot];
```

Let us first define some (restorable) settings and a helper function, and visit Section 2.1 to make acquaintance with the symbol sSamplingSpan behavior:

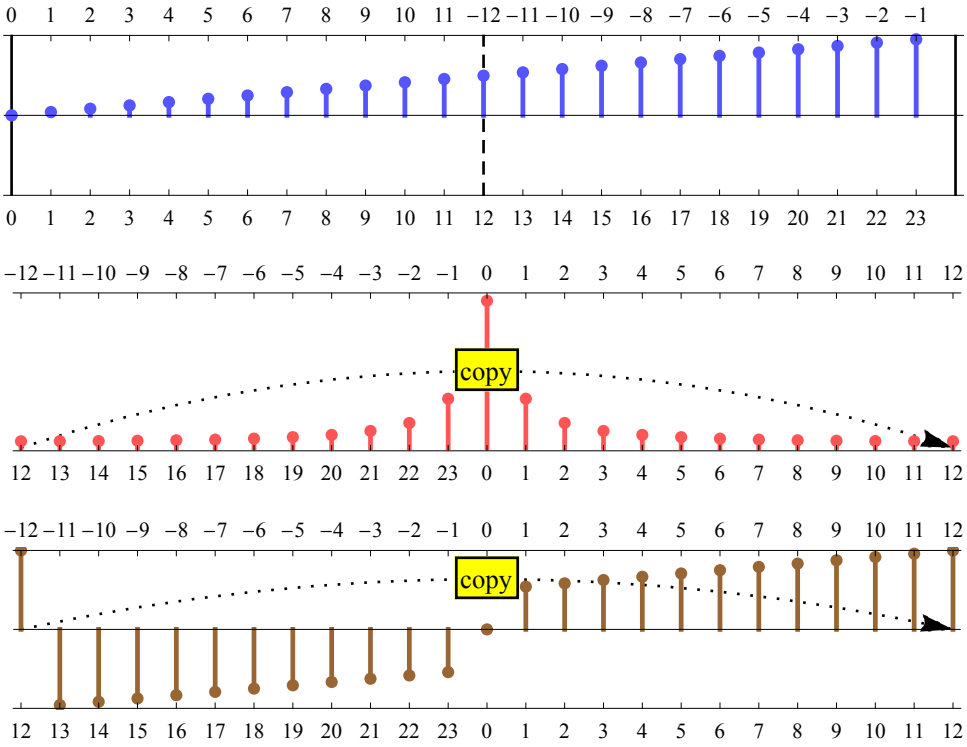
```
In[332]:= n = 24;
SetOptions[ListPlot, PlotRange → {{-0.2, n + 0.2}, {-n - 0.2, n + 0.2}},
  Axes → True, Frame → {{False, False}, {True, True}}, DataRange → {0, n - 1},
  PlotStyle → Directive[PointSize[Large]], Filling → Axis,
  FillingStyle → Directive[Thickness[0.005]], GridLines → None,
  LabelStyle → 8, AspectRatio → 1 / 6, ImageSize → Medium];
rothlApp[u_List, arc1_List, arc2_List] :=
 Module[
  {ft = Chop@sRotateHalfLength[Fourier[u, FourierParameters → {1, -1}], 1],
   s = sSamplingSpan[n], s1 = sSamplingSpan[n + 1], v, w},
  {v, w} = {Abs@ft, Arg@ft};
  If[ft =!= $Failed, Column[{
     ListPlot[u, FrameTicks → {{None, None},
        {s, {s, sRotateHalfLength[sSamplingSpan[-n]]}^T}},
      PlotStyle → Directive[Lighter@Blue],
      Prolog → {Black, Line[{{0, -n}, {0, n}}], Line[{{n, -n}, {n, n}}],
        Dashed, Line[{{Floor[n / 2], -n}, {Floor[n / 2], n}}]}],
     ListPlot[v, DataRange → {0, n}, PlotRange → {{-0.2, n + 0.2}, Full},
      FrameTicks → {{None, None}, {{s1, sRotateHalfLength[s1, 1, True &]}^T,
         {s1, sSamplingSpan[-n - 1]}^T}}, PlotStyle → Directive[Lighter@Red],
      Prolog → {Black, Dotted, Arrowheads[Medium],
        Arrow@BezierCurve[{{0, 0}, arc1, {n, 0}}]},
      Epilog →
       {Inset[Framed[Style["copy", 10], Background → Yellow,
          FrameMargins → Tiny], {n / 2, arc1[[2]] / 2}]}],
     ListPlot[w, DataRange → {0, n}, PlotRange → {{-0.2, n + 0.2}, {-π, π}},
      FrameTicks → {{None, None}, {{s1, sRotateHalfLength[s1, 1, True &]}^T,
         {s1, sSamplingSpan[-n - 1]}^T}}, PlotStyle → Directive[Brown],
      Prolog → {Black, Dotted, Arrowheads[Medium],
        Arrow@BezierCurve[{{0, 0}, arc2, {n, 0}}]},
      Epilog →
       {Inset[Framed[Style["copy", 10], Background → Yellow,
          FrameMargins → Tiny], {n / 2, arc2[[2]] / 2}]}]
    }, Alignment → Left, Spacings → {Scaled[0.025], Scaled[0.025]}]]]
```

Now we pick a sampled signal (with an even number of samples, 24 for these examples), submit it to the DFT in the form of the built-in function Fourier with FourierParameters → {1, −1}, and

apply sRotateHalfLength[#, 1] which appends the duplicate of the first element. Finally, we plot the original signal (top, blue style), amplitude spectrum (middle, red style), ans phase spectrum (bottom, brown style, plot range $\{\pi, -\pi\}$). The results for some typical signals are as follows. Unless the reader is familiar with discrete signal processing, and particularly the DFT, reading, e.g., [Lyons, 2011] should help.

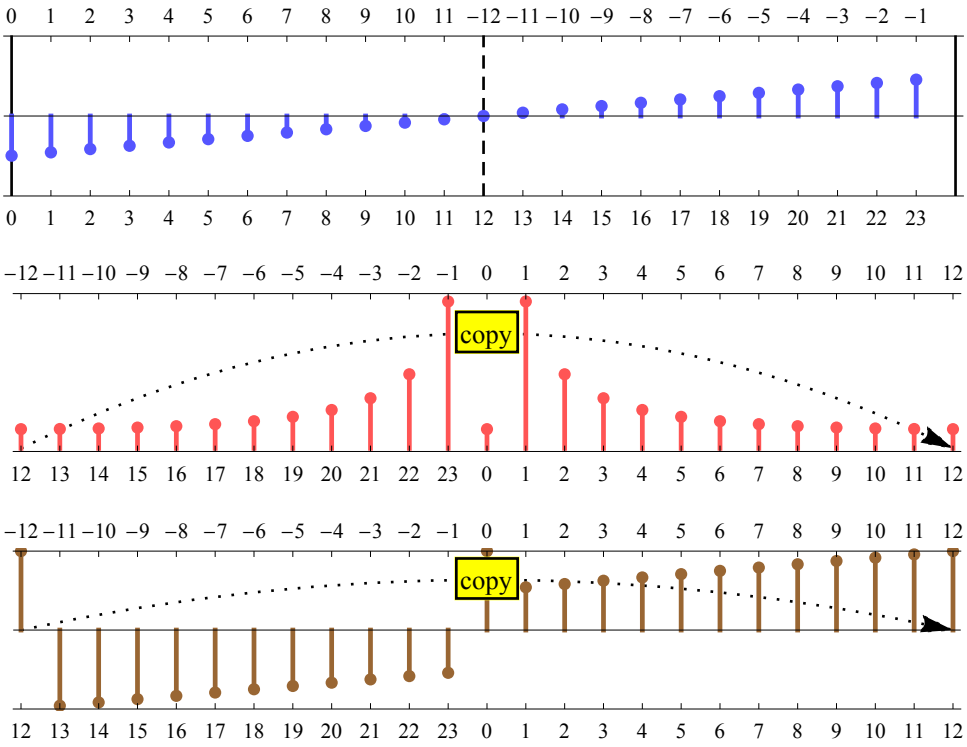
In[335]:=
```
rothlApp[sSamplingSpan[n], {n / 2, n^2 / 2}, {n / 2, 4}]
```

Out[335]=



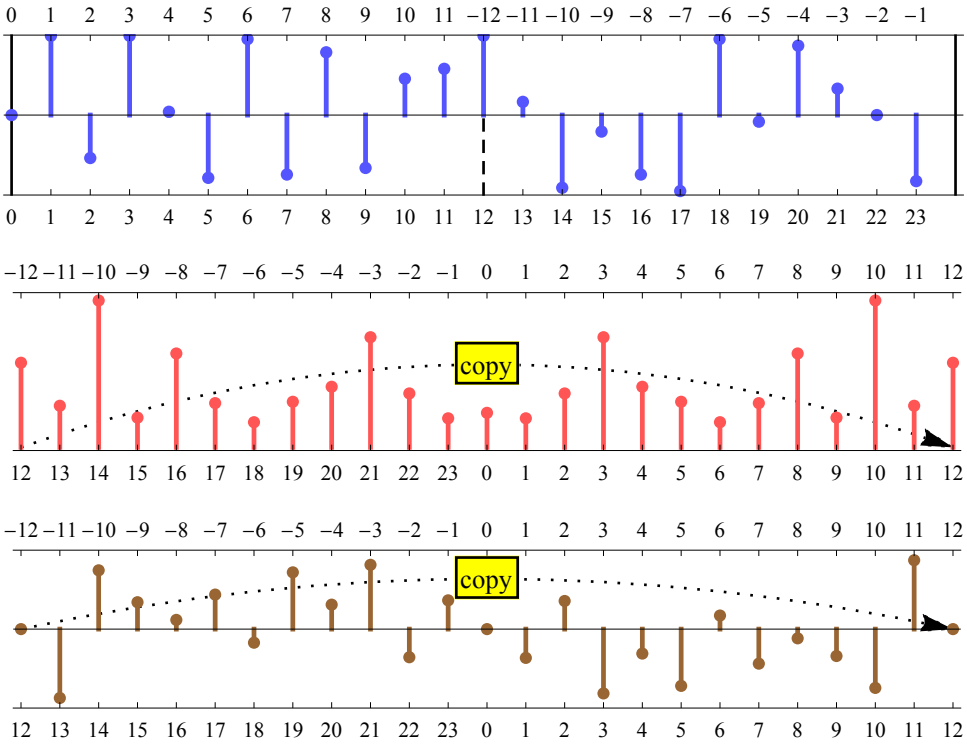
In[336]:=
```
rothlApp[sSamplingSpan[-n], {n / 2, n^2 / 4}, {n / 2, 4}]
```

Out[336]=



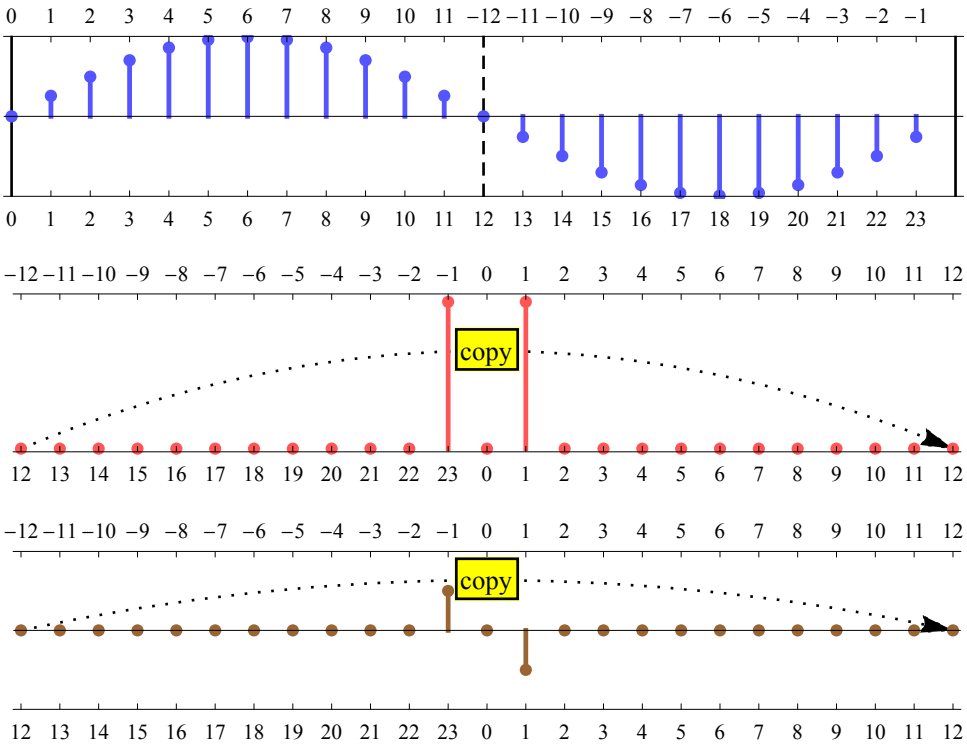
In[337]:=
```
rothlApp[RandomInteger[{-n, n}, n], {n / 2, n^2 / 3}, {n / 2, 4}]
```

Out[337]=



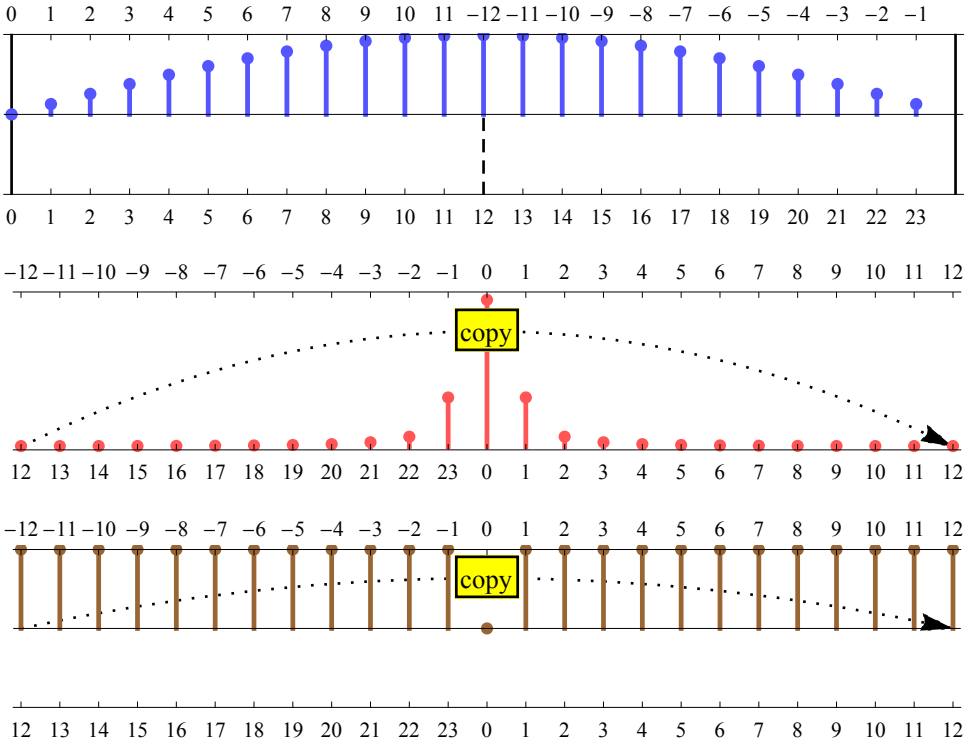
In[338]:=
```
rothlApp[n Sin[sCircularlySample[{0, 2 π}, n]], {n / 2, n^2 / 1.5},
 {n / 2, 4}]
```

Out[338]=



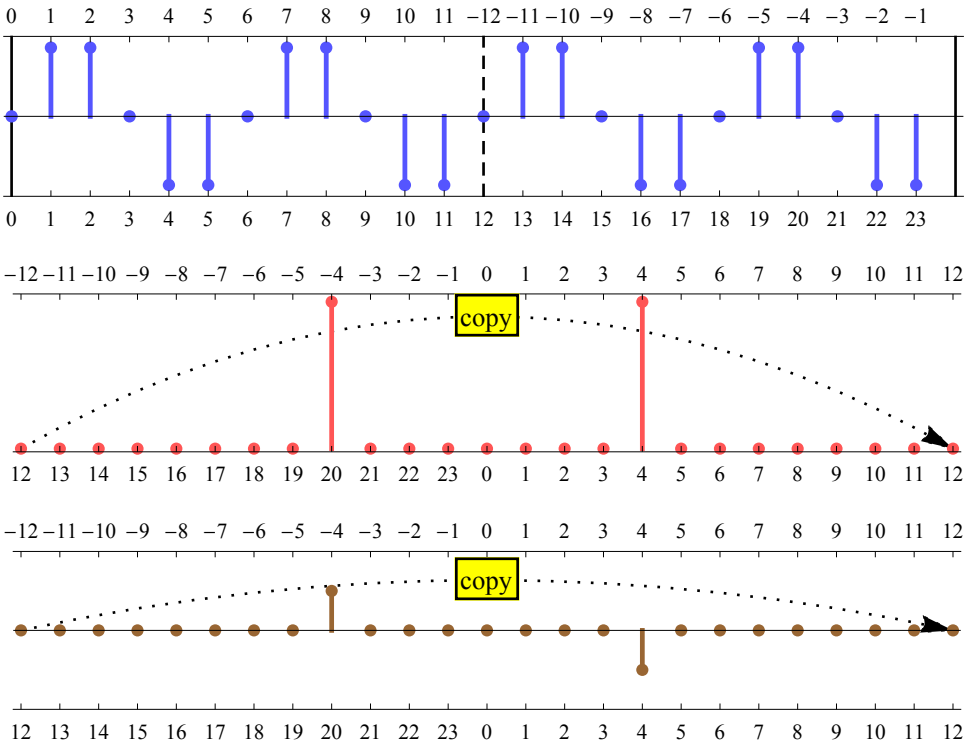
In[339]:=
```
rothlApp[n Sin[sCircularlySample[{0, π}, n]], {n / 2, n^2}, {n / 2, 4}]
```

Out[339]=



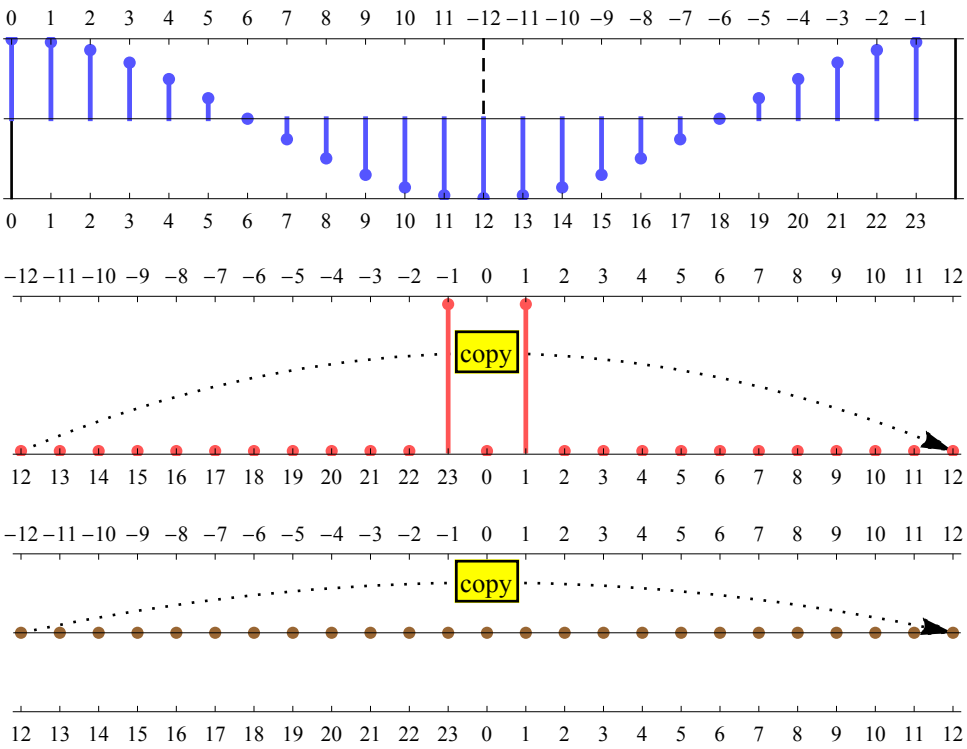
In[340]:=
```
rothlApp[n Sin[sCircularlySample[{-4 π, 4 π}, n]], {n / 2, 0.9 n^2},
 {n / 2, 4}]
```

Out[340]=



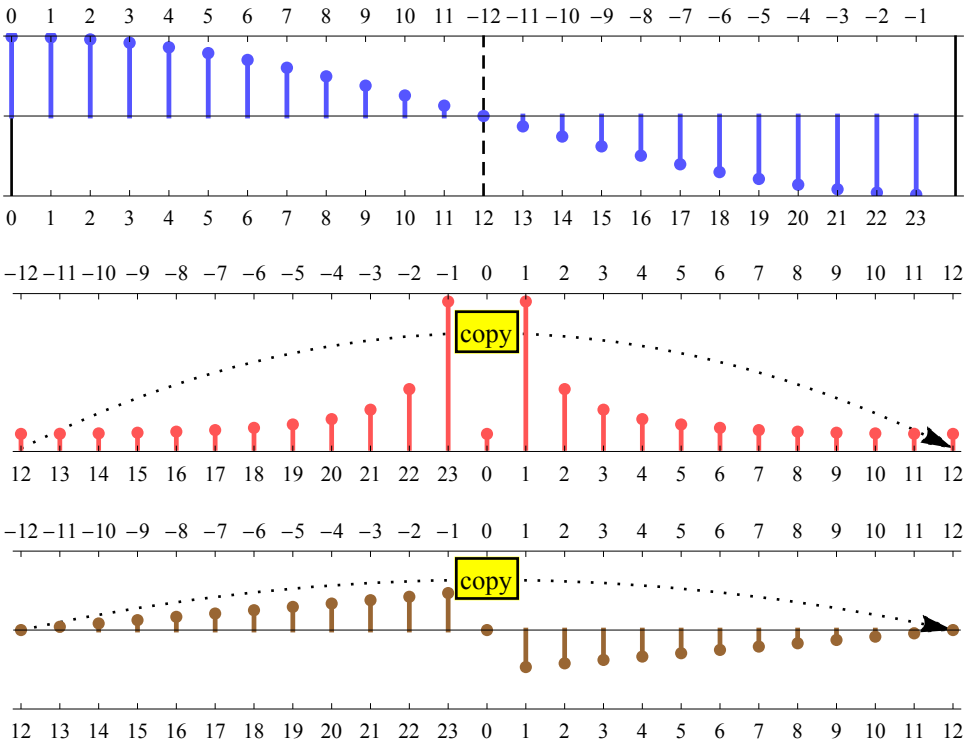
In[341]:=
```
rothlApp[n Cos[sCircularlySample[{0, 2 π}, n]], {n / 2, n^2 / 1.5},
 {n / 2, 4}]
```

Out[341]=



In[342]:=
```
rothlApp[n Cos[sCircularlySample[{0, π}, n]], {n / 2, n^2 / 1.5}, {n / 2, 4}]
```

Out[342]=



Now we can restore the ListPlot options setting to its original value:

In[343]:=
```
SetOptions[ListPlot, Sequence @@ optsListPlot];
```

Audio

Consider the following audio signal:

In[344]:=
```
aud = AudioGenerator[{"Sin", 200 π # &}, 8, SampleRate → 16000]
adat = AudioData[aud];
apar = Through[{AudioSampleRate, AudioType, AudioChannels, Duration}[aud]]
```

Out[344]=

Out[346]=
```
{16000 Hz, Real32, 1, 8. s}
```

This is the audio signal half-length rotated:

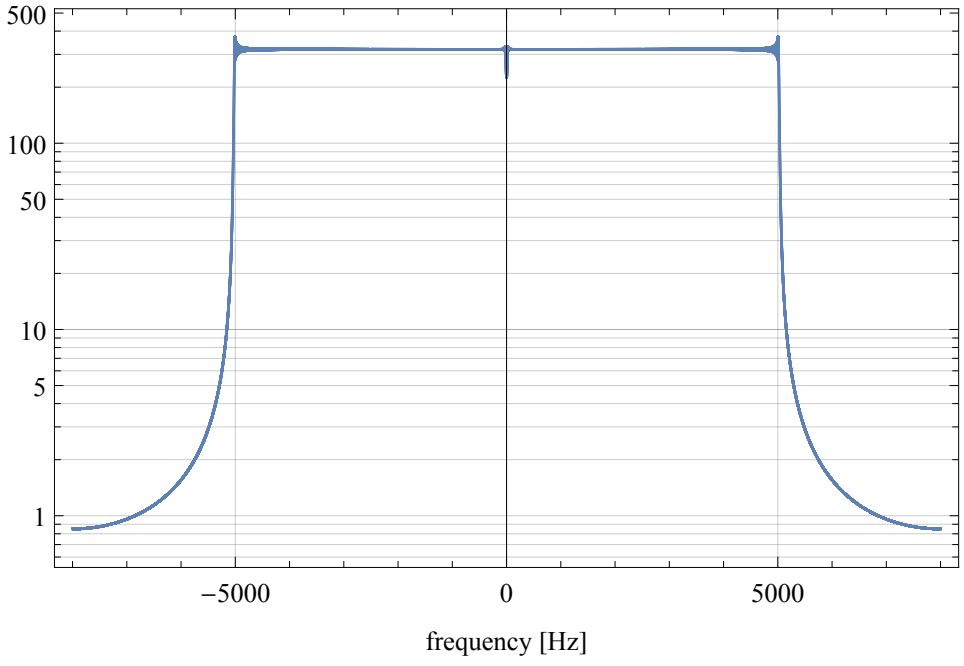
In[347]:=
```
Audio[sRotateHalfLength /@ adat, SampleRate → apar[[1, 1]]]
```

Out[347]=

sRotateHalfLength is useful for correctly plotting the amplitude spectrum:

In[348]:=
```
spectrum = sRotateHalfLength[
    Fourier[First[adat], FourierParameters → {1, -1}], 1];
ListLogPlot[Abs[spectrum],
 DataRange → (apar[[1, 1]] / (Length[spectrum] - 1))
   {-(Length[spectrum] - 1) / 2, (Length[spectrum] - 1) / 2}, Frame → True,
 FrameLabel → {"frequency [Hz]", None},
 GridLines →
  {Automatic, Join[Range[0.6, 0.9, 0.1], Range[1, 10], Range[10, 100, 10],
    {200, 300, 400}]}]
```

Out[349]=



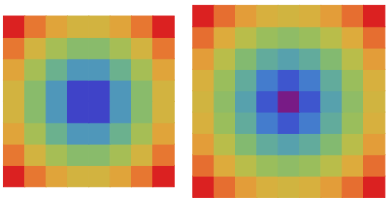
We can easily convince ourselves that the half-rotated audio has exactly the same amplitude spectrum—just replace the "adat" symbol in the first row of the above code with "sRotateHalfLength/@adat".

Images

Generate an even-size and an odd-size color images:

In[350]:=
```
idesc = {(*ImageType,*)ImageDimensions(*,ImageChannels*)};
i2 = {i2e = RadialGradientImage["Rainbow", {8, 8}, ImageSize → {64, 64}],
   i2o = RadialGradientImage["Rainbow", {9, 9}, ImageSize → {72, 72}]};
Grid@{i2, i2d = {Through[idesc[i2e]], Through[idesc[i2o]]}}
```

Out[352]=



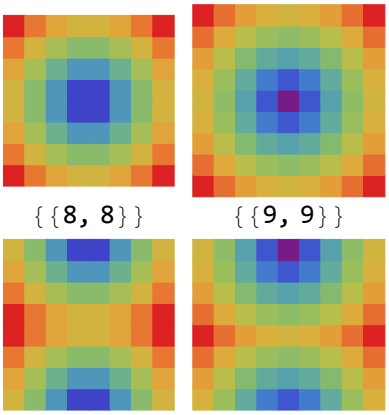
{{8, 8}} {{9, 9}}

In the following examples, the images are arranged in a grid. The two original images defined above are displayed in the upper row of the grid, while those with rows and/or columns rotated by sRotateHalfLength are displayed in the bottom row of the grid. Each image is labeled with a {{width, height}} caption.

Row operation

In[353]:=
```
res = {Image[sRotateHalfLength[ImageData@i2e], ImageSize → {64, 64}],
   Image[sRotateHalfLength[ImageData@i2o], ImageSize → {72, 64}]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

Out[354]=

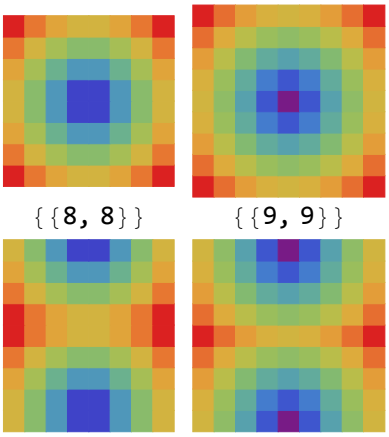


{{8, 8}} {{9, 9}}

{{8, 8}} {{9, 8}}

In[355]:=
```
res = {Image[sRotateHalfLength[ImageData@i2e, 1], ImageSize → {64, 72}],
    Image[sRotateHalfLength[ImageData@i2o, 1], ImageSize → {72, 72}]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

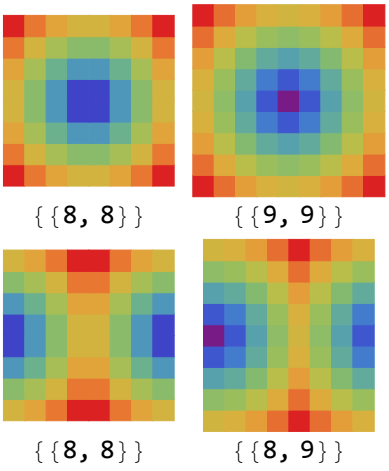
Out[356]=



Column operation

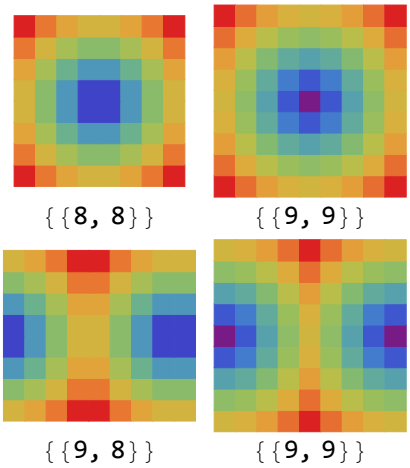
In[357]:=
```
res = {Image[sRotateHalfLength[#] & /@ ImageData@i2e, ImageSize → {64, 64}],
    Image[sRotateHalfLength[#] & /@ ImageData@i2o, ImageSize → {64, 72}]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

Out[358]=



In[359]:=
```
res = {Image[sRotateHalfLength[#, 1] & /@ ImageData@i2e, ImageSize → {72, 64}],
   Image[sRotateHalfLength[#, 1] & /@ ImageData@i2o, ImageSize → {72, 72}]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

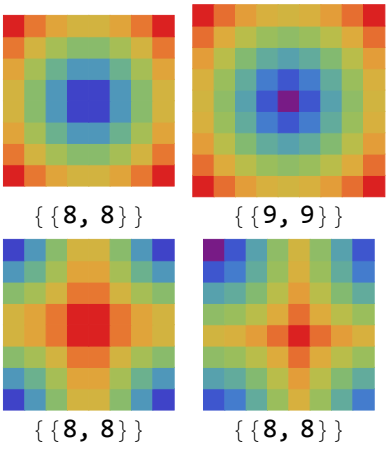
Out[360]=



Row and column operation

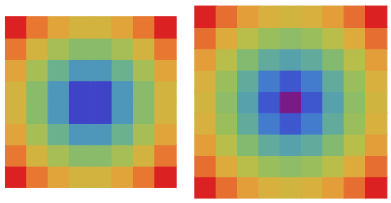
In[361]:=
```
res =
  {Image[Map[sRotateHalfLength[#] &, sRotateHalfLength[ImageData@i2e], {1}],
    ImageSize → {64, 64}],
   Image[Map[sRotateHalfLength[#] &, sRotateHalfLength[ImageData@i2o], {1}],
    ImageSize → {64, 64}]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

Out[362]=

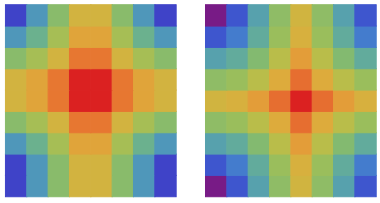


In[363]:=
```
res =
  {Image[Map[sRotateHalfLength[#] &, sRotateHalfLength[ImageData@i2e, 1],
     {1}], ImageSize → {64, 72}],
   Image[Map[sRotateHalfLength[#] &, sRotateHalfLength[ImageData@i2o, 1],
     {1}], ImageSize → {64, 72}]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

Out[364]=



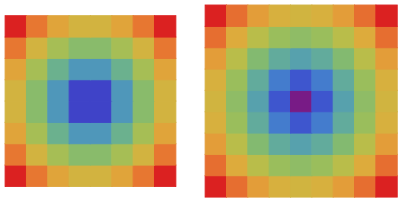
{{8, 8}} {{9, 9}}



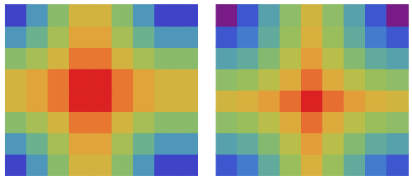
{{8, 9}} {{8, 9}}

In[365]:=
```
res =
  {Image[Map[sRotateHalfLength[#, 1] &, sRotateHalfLength[ImageData@i2e],
     {1}], ImageSize → {72, 64}],
   Image[Map[sRotateHalfLength[#, 1] &, sRotateHalfLength[ImageData@i2o],
     {1}], ImageSize → {72, 64}]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

Out[366]=



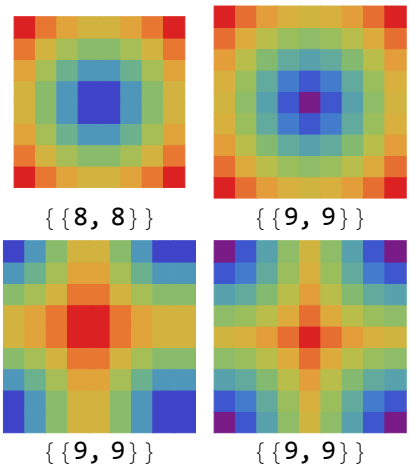
{{8, 8}} {{9, 9}}



{{9, 8}} {{9, 8}}

In[367]:=
```
res =
  {Image[Map[sRotateHalfLength[#, 1] &, sRotateHalfLength[ImageData@i2e, 1],
     {1}], ImageSize → {72, 72}],
   Image[Map[sRotateHalfLength[#, 1] &, sRotateHalfLength[ImageData@i2o, 1],
     {1}], ImageSize → {72, 72}]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

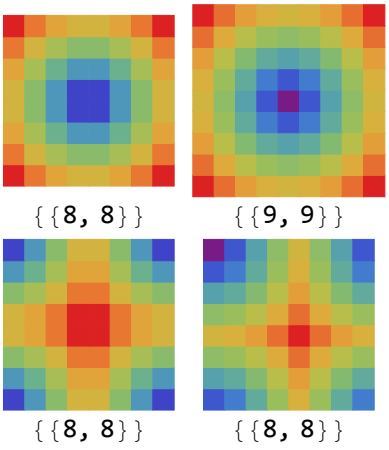
Out[368]=



The "Fold idiom"

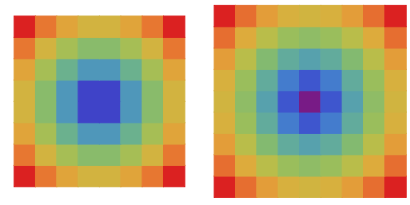
In[369]:=
```
res = {With[{d = ImageData@i2e},
    Image[Fold[Map[sRotateHalfLength[#] &, #1, {#2}] &, d,
      Range[0, ArrayDepth[d] - 2]], ImageSize → {64, 64}]],
   With[{d = ImageData@i2o},
    Image[Fold[Map[sRotateHalfLength[#] &, #1, {#2}] &, d,
      Range[0, ArrayDepth[d] - 2]], ImageSize → {64, 64}]]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

Out[370]=

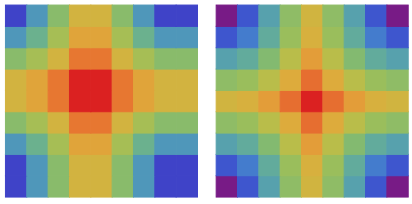


In[371]:=
```
res = {With[{d = ImageData@i2e},
    Image[Fold[Map[sRotateHalfLength[#, 1] &, #1, {#2}] &, d,
      Range[0, ArrayDepth[d] - 2]], ImageSize → {72, 72}]],
   With[{d = ImageData@i2o},
    Image[Fold[Map[sRotateHalfLength[#, 1] &, #1, {#2}] &, d,
      Range[0, ArrayDepth[d] - 2]], ImageSize → {72, 72}]]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

Out[372]=



{{8, 8}} {{9, 9}}

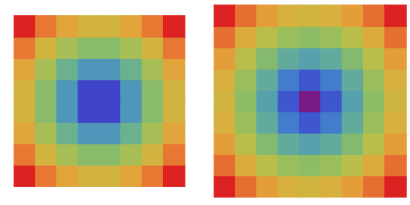


{{9, 9}} {{9, 9}}

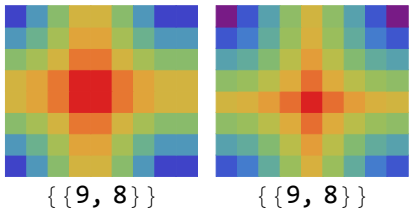
Control appending aliased element for each level separately using the "Fold idiom"

In[373]:=
```
res = {With[{d = ImageData@i2e},
    Image[Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a] &], #1, {#2[[1]]}] &
      d, {Range[0, ArrayDepth[d] - 2],
       {0, 1}(* {rows, columns} *)
       }^T], ImageSize → {72, 64}]], With[{d = ImageData@i2o},
    Image[
     Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a] &], #1, {#2[[1]]}] &,
      d, {Range[0, ArrayDepth[d] - 2],
       {0, 1}(* {rows, columns} *)
       }^T], ImageSize → {72, 64}]]};
Grid@{i2, i2d, res, Through[idesc[#]] & /@ res}
```

Out[374]=



{{8, 8}} {{9, 9}}



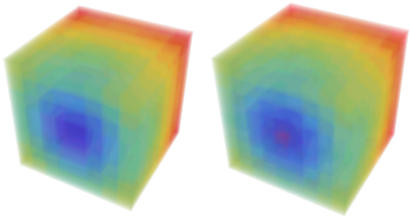
{{9, 8}} {{9, 8}}

3D images

Generate an even-size and an odd-size 3D color images:

In[375]:=
```
i3 =
  {i3e = RadialGradientImage[{Front, {Right, Back, Top}} →
      ColorData["Rainbow"], {8, 8, 8}],
   i3o = RadialGradientImage[{Front, {Right, Back, Top}} →
      ColorData["Rainbow"], {9, 9, 9}]};
Grid@{i3, i3d = {Through[idesc[i3e]], Through[idesc[i3o]]}}
```

Out[376]=



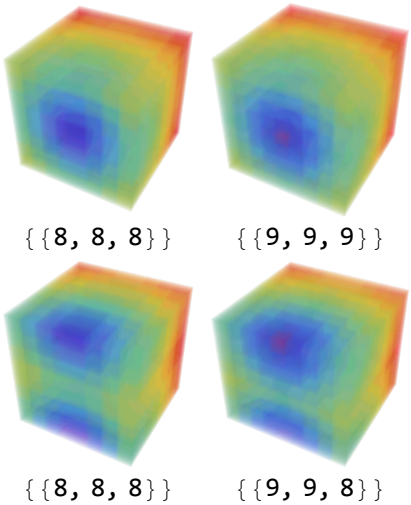
{{8, 8, 8}} {{9, 9, 9}}

In the following examples, the 3D images are arranged in a grid. The two original 3D images defined above are displayed in the upper row of the grid, while those with rows, columns, and depth rotated by sRotateHalfLength are displayed in the bottom row of the grid. Each 3D image is labeled with a {{width (number of columns), depth, height (number of rows)}} caption.

Row operation

In[377]:=
```
res = {Image3D[sRotateHalfLength[ImageData@i3e]],
   Image3D[sRotateHalfLength[ImageData@i3o]]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[378]=

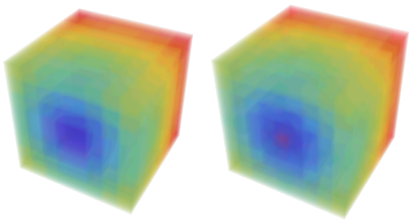


{{8, 8, 8}} {{9, 9, 9}}

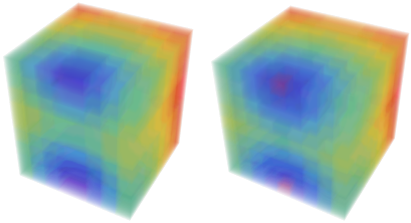
{{8, 8, 8}} {{9, 9, 8}}

In[379]:=
```
res = {Image3D[sRotateHalfLength[ImageData@i3e, 1]],
   Image3D[sRotateHalfLength[ImageData@i3o, 1]]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[380]=



{{8, 8, 8}} {{9, 9, 9}}

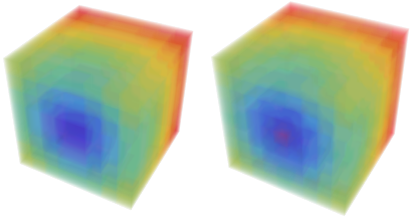


{{8, 8, 9}} {{9, 9, 9}}

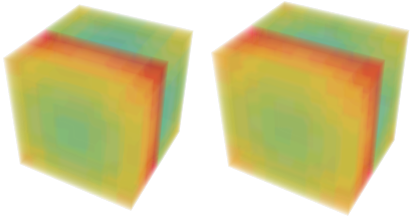
Depth operation

In[381]:=
```
res = {Image3D[sRotateHalfLength[#, 0] & /@ ImageData@i3e],
   Image3D[sRotateHalfLength[#, 0, True &] & /@ ImageData@i3o]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[382]=



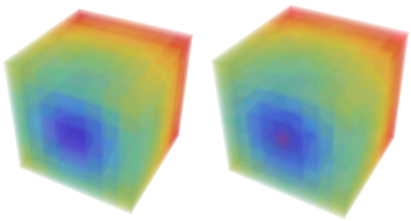
{{8, 8, 8}} {{9, 9, 9}}



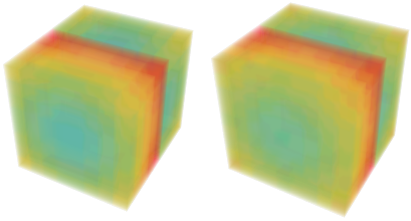
{{8, 8, 8}} {{9, 8, 9}}

In[383]:=
```
res = {Image3D[sRotateHalfLength[#, 1] & /@ ImageData@i3e],
   Image3D[sRotateHalfLength[#, 1, True &] & /@ ImageData@i3o]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[384]=



{{8, 8, 8}} {{9, 9, 9}}

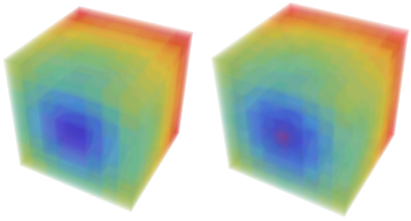


{{8, 9, 8}} {{9, 9, 9}}

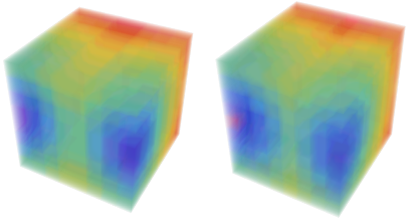
Column operation

In[385]:=
```
res = {Image3D[Map[sRotateHalfLength[#, 0] &, ImageData@i3e, {2}]],
   Image3D[Map[sRotateHalfLength[#, 0] &, ImageData@i3o, {2}]]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[386]=



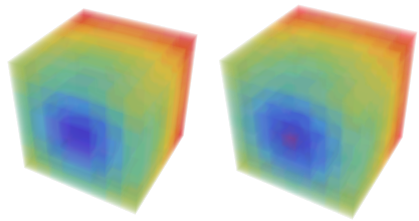
{{8, 8, 8}} {{9, 9, 9}}



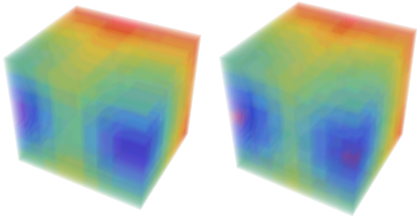
{{8, 8, 8}} {{8, 9, 9}}

In[387]:=
```
res = {Image3D[Map[sRotateHalfLength[#, 1] &, ImageData@i3e, {2}]],
    Image3D[Map[sRotateHalfLength[#, 1] &, ImageData@i3o, {2}]]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[388]=



{{8, 8, 8}} {{9, 9, 9}}

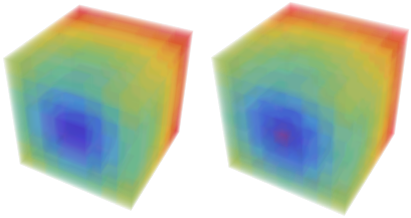


{{9, 8, 8}} {{9, 9, 9}}

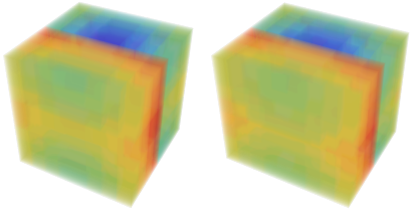
Row and depth operation

In[389]:=
```
res =
  {Image3D[Map[sRotateHalfLength[#, 0] &,
     Map[sRotateHalfLength[#, 0] &, ImageData@i3e, {0}], {1}]],
   Image3D[Map[sRotateHalfLength[#, 0, True &] &,
     Map[sRotateHalfLength[#, 0] &, ImageData@i3o, {0}], {1}]]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[390]=



{{8, 8, 8}} {{9, 9, 9}}

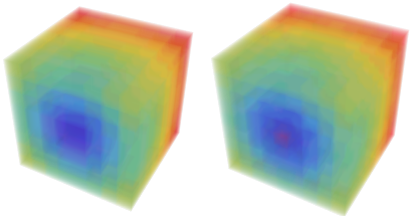


{{8, 8, 8}} {{9, 8, 8}}

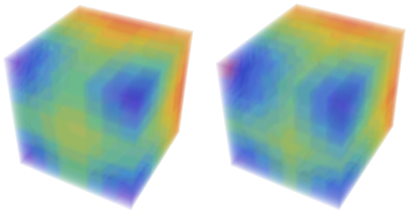
Row and column operation

In[391]:=
```
res =
  {Image3D[Map[sRotateHalfLength[#, 0] &,
     Map[sRotateHalfLength[#, 0] &, ImageData@i3e, {0}], {2}]],
   Image3D[Map[sRotateHalfLength[#, 0, True &] &,
     Map[sRotateHalfLength[#, 0] &, ImageData@i3o, {0}], {2}]]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[392]=



{{8, 8, 8}} {{9, 9, 9}}

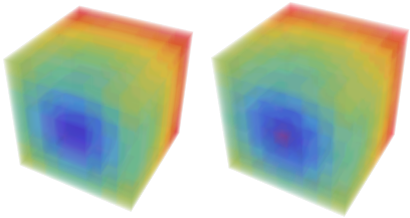


{{8, 8, 8}} {{8, 9, 8}}

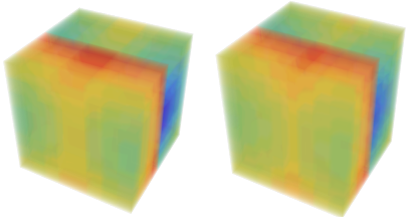
Depth and column operation

In[393]:=
```
res =
  {Image3D[Map[sRotateHalfLength[#, 0] &,
     Map[sRotateHalfLength[#, 0] &, ImageData@i3e, {1}], {2}]],
   Image3D[Map[sRotateHalfLength[#, 0] &,
     Map[sRotateHalfLength[#, 0, True &] &, ImageData@i3o, {1}], {2}]]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[394]=



{{8, 8, 8}} {{9, 9, 9}}

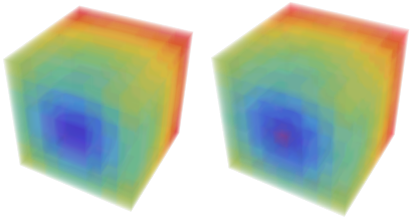


{{8, 8, 8}} {{8, 8, 9}}

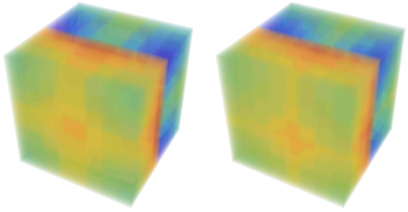
Row, depth and column operation

In[395]:=
```
res =
  {Image3D[Map[sRotateHalfLength[#, 0] &,
     Map[sRotateHalfLength[#, 0] &,
      Map[sRotateHalfLength[#, 0] &, ImageData@i3e, {0}], {1}], {2}]],
   Image3D[Map[sRotateHalfLength[#, 0] &,
     Map[sRotateHalfLength[#, 0, True &] &,
      Map[sRotateHalfLength[#, 0] &, ImageData@i3o, {0}], {1}], {2}]]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[396]=



{{8, 8, 8}} {{9, 9, 9}}

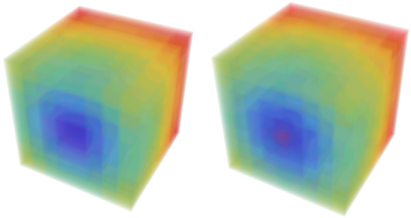


{{8, 8, 8}} {{8, 8, 8}}

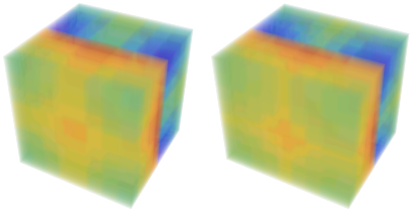
Control appending aliased element for each level separately with the "Fold idiom"

In[397]:=
```
res = {With[{d = ImageData@i3e},
      Image3D[
       Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &],
          #1, {#2[[1]]}] &, d, {Range[0, ArrayDepth[d] - 2],
          {0, 0, 0}(* row, depth, column *)}^T]]], With[{d = ImageData@i3o},
      Image3D[
       Fold[Map[With[{a = #2[[2]]}, sRotateHalfLength[#, a, True &] &],
          #1, {#2[[1]]}] &, d, {Range[0, ArrayDepth[d] - 2],
          {0, 0, 1}(* row, depth, column *)}^T]]]};
Grid@{i3, i3d, res, Through[idesc[#]] & /@ res}
```

Out[398]=



{{8, 8, 8}} {{9, 9, 9}}



{{8, 8, 8}} {{9, 8, 8}}

1.5 sDimensionsBoxed

In[399]:=
```
infoAbout@sDimensionsBoxed
```

sDimensionsBoxed[*expr*] returns {Bb, Ib} with
Bb (Ib) the minimum bounding (maximum inscribed) box of *expr*.

```
Attributes[sDimensionsBoxed] = {Protected, ReadProtected}
```

```
SyntaxInformation[sDimensionsBoxed] = {ArgumentsPattern → {_, _.}}
```

Details

sDimensionsBoxed[*expr*, *level*] returns {Bb, Ib}, where Bb is a minimum bounding box of *expr* (list of dimensions of the tightest hyper-rectangle containing *expr*) and Ib is a maximum inscribed box of *expr* (list of dimensions of the greatest hyper-rectangle contained in *expr*).

- Unlike the Dimensions system function, *expr* need not to be a full rectangular array (it is allowed to be "ragged").
- *expr* is typically a nested list (but any head is accepted, not just List).

- An integer *level* specifies the level to dive dimensions down to (default value −1 causes using exactly the full depth of *expr*). The level behavior differs from that of Dimensions: Infinity is not accepted.
- Caveat: difficulties with correct output for Association still persist.

Examples

Clearing global symbols:

```
In[400]:= Clear[h, x, y, z];
          ClearAll[printBoxes];
```

First let us define a helper function to save space for displaying examples:

```
In[402]:= printBoxes[x_] :=
           TableForm[
            ToString /@
             Through[{Identity, Length, Depth, ArrayDepth, Dimensions,
                sDimensionsBoxed}[x]],
            TableHeadings →
             {{"expression", "Length", "Depth", "ArrayDepth", "Dimensions",
               "sDimensionsBoxed"}, None}]
```

For full arrays (those arrays for which the ArrayQ predicate gives True), Dimensions and each part (minimum bounding box Bb and maximum inscribed box Ib) of sDimensionsBoxed provide identical results:

```
In[403]:= printBoxes[10]
```

Out[403]//TableForm=

expression	10
Length	0
Depth	1
ArrayDepth	0
Dimensions	{}
sDimensionsBoxed	{{}, {}}

```
In[404]:= Row@{printBoxes[{}], Spacer@20, printBoxes[h[]]}
```

Out[404]=

expression	{}	expression	h[]
Length	0	Length	0
Depth	2	Depth	2
ArrayDepth	1	ArrayDepth	1
Dimensions	{0}	Dimensions	{0}
sDimensionsBoxed	{{0}, {0}}	sDimensionsBoxed	{{0}, {0}}

In[405]:=
```
Row@{printBoxes[{{}, {}}], Spacer@20, printBoxes[h[h[], h[]]]}
```

Out[405]=

expression	{{}, {}}	expression	h[h[], h[]]
Length	2	Length	2
Depth	3	Depth	3
ArrayDepth	2	ArrayDepth	2
Dimensions	{2, 0}	Dimensions	{2, 0}
sDimensionsBoxed	{{2, 0}, {2, 0}}	sDimensionsBoxed	{{2, 0}, {2, 0}}

In[406]:=
```
Block[{x = Range@4, h}, Row@{printBoxes[x], Spacer@20, printBoxes[h @@ x]}]
```

Out[406]=

expression	{1, 2, 3, 4}	expression	h[1, 2, 3, 4]
Length	4	Length	4
Depth	2	Depth	2
ArrayDepth	1	ArrayDepth	1
Dimensions	{4}	Dimensions	{4}
sDimensionsBoxed	{{4}, {4}}	sDimensionsBoxed	{{4}, {4}}

In[407]:=
```
printBoxes[x + y + z]
```

Out[407]//TableForm=

expression	x + y + z
Length	3
Depth	2
ArrayDepth	1
Dimensions	{3}
sDimensionsBoxed	{{3}, {3}}

In[408]:=
```
Block[{x = Array[10 #1 + #2 &, {2, 3}], h},
 Row@{printBoxes[x], Spacer@20, printBoxes[Apply[h, x, {0, 1}]]}]
```

Out[408]=

expression	{{11, 12, 13}, {21, 22, 23}}
Length	2
Depth	3
ArrayDepth	2
Dimensions	{2, 3}
sDimensionsBoxed	{{2, 3}, {2, 3}}

expression	h[h[11, 12, 13], h[21, 22, 23]]
Length	2
Depth	3
ArrayDepth	2
Dimensions	{2, 3}
sDimensionsBoxed	{{2, 3}, {2, 3}}

In[409]:=
```
Block[{x = Table[{}, {2}, {3}], h},
 Row@{printBoxes[x], Spacer@20, printBoxes[Apply[h, x, {0, 2}]]}]
```

Out[409]=

expression	{{{}, {}, {}}, {{}, {}, {}}}
Length	2
Depth	4
ArrayDepth	3
Dimensions	{2, 3, 0}
sDimensionsBoxed	{{2, 3, 0}, {2, 3, 0}}

expression	h[h[h[], h[], h[]], h[h[], h[], h[]]]
Length	2
Depth	4
ArrayDepth	3
Dimensions	{2, 3, 0}
sDimensionsBoxed	{{2, 3, 0}, {2, 3, 0}}

In[410]:=
```
Block[{x = Array[100 #1 + 10 #2 + #3 &, {2, 1, 2}], h},
 Row@{printBoxes[x], Spacer@20, printBoxes[Apply[h, x, {0, 2}]]}]
```

Out[410]=

expression	{{{111, 112}}, {{211, 212}}}
Length	2
Depth	4
ArrayDepth	3
Dimensions	{2, 1, 2}
sDimensionsBoxed	{{2, 1, 2}, {2, 1, 2}}

expression	h[h[h[111, 112]], h[h[211, 212]]]
Length	2
Depth	4
ArrayDepth	3
Dimensions	{2, 1, 2}
sDimensionsBoxed	{{2, 1, 2}, {2, 1, 2}}

However, be careful when determining the boxes of expressions like this:

In[411]:=
```
printBoxes[Array[x_FromDigits[{##}] &, {3}]]
```

Out[411]//TableForm=

expression	{x , x , x } 1 2 3
Length	3
Depth	3
ArrayDepth	1
Dimensions	{3}
sDimensionsBoxed	{{3, 2}, {3, 2}}

The reason is due to that elements are Subscript's:

In[412]:=
```
Array[x_FromDigits[{##}] &, {3}] // InputForm
```

Out[412]//InputForm=
```
{Subscript[x, 1], Subscript[x, 2], Subscript[x, 3]}
```

For "ragged" arrays the minimum bounding box Bb and maximum inscribed box Ib however

differ, providing something completely different than Dimensions:

```
In[413]:= printBoxes[Select[#, PrimeQ] & /@ Array[10 #1 + #2 &, {5, 8}]]
```

Out[413]//TableForm=

expression	{{11, 13, 17}, {23}, {31, 37}, {41, 43, 47}, {53}}
Length	5
Depth	3
ArrayDepth	1
Dimensions	{5}
sDimensionsBoxed	{{5, 3}, {5, 1}}

```
In[414]:= Block[
           {x = Delete[Map[Select[#, PrimeQ] &, Array[100 #1 + 10 #2 + #3 &, {2, 3, 8}],
                {2}], {2, -1}], h},
           Column@{printBoxes[x], Spacer@20,
             printBoxes[Apply[h, #, {0, Depth[#]}] &[x]]}]
```

expression	{{{113}, {127}, {131, 137}}, {{211}, {223, 227}}}
Length	2
Depth	4
ArrayDepth	1
Dimensions	{2}
sDimensionsBoxed	{{2, 3, 2}, {2, 2, 1}}

Out[414]=

expression	h[h[h[113], h[127], h[131, 137]], h[h[211], h[223, 227]]]
Length	2
Depth	4
ArrayDepth	1
Dimensions	{2}
sDimensionsBoxed	{{2, 3, 2}, {2, 2, 1}}

Let us define two nested lists, one being a full array, the second a “ragged” nested list, and examine level specification:

```
In[415]:= (x = Array[1000 #1 + 100 #2 + 10 #3 + #4 &, {2, 2, 5, 4}]) // MatrixForm
          (y = {{{{1123}, {}, {}, {1151, 1153}}, {{1213}, {1231}, {}, {}}},
              {{{2213}, {2221}, {}, {2243}, {2251}}}}) // MatrixForm
```

Out[415]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} 1111 & 1112 & 1113 & 1114 \\ 1121 & 1122 & 1123 & 1124 \\ 1131 & 1132 & 1133 & 1134 \\ 1141 & 1142 & 1143 & 1144 \\ 1151 & 1152 & 1153 & 1154 \end{pmatrix} & \begin{pmatrix} 1211 & 1212 & 1213 & 1214 \\ 1221 & 1222 & 1223 & 1224 \\ 1231 & 1232 & 1233 & 1234 \\ 1241 & 1242 & 1243 & 1244 \\ 1251 & 1252 & 1253 & 1254 \end{pmatrix} \\ \begin{pmatrix} 2111 & 2112 & 2113 & 2114 \\ 2121 & 2122 & 2123 & 2124 \\ 2131 & 2132 & 2133 & 2134 \\ 2141 & 2142 & 2143 & 2144 \\ 2151 & 2152 & 2153 & 2154 \end{pmatrix} & \begin{pmatrix} 2211 & 2212 & 2213 & 2214 \\ 2221 & 2222 & 2223 & 2224 \\ 2231 & 2232 & 2233 & 2234 \\ 2241 & 2242 & 2243 & 2244 \\ 2251 & 2252 & 2253 & 2254 \end{pmatrix} \end{pmatrix}$$

Out[416]//MatrixForm=

$$\begin{pmatrix} \{\{\{1123\}, \{\}, \{\}, \{1151, 1153\}\}, \{\{1213\}, \{1231\}, \{\}, \{\}\}\} \\ \{\{\{2213\}, \{2221\}, \{\}, \{2243\}, \{2251\}\}\} \end{pmatrix}$$

```
In[417]:= Column[sDimensionsBoxed[#] & /@ {x, y}]

Out[417]= {{2, 2, 5, 4}, {2, 2, 5, 4}}
          {{2, 2, 5, 2}, {2, 1, 4, 0}}
```

Level specifications:

```
In[418]:= Column[sDimensionsBoxed[#, 0] & /@ {x, y}]

Out[418]= {{}, {}}
          {{}, {}}
```

```
In[419]:= Column[sDimensionsBoxed[#, 1] & /@ {x, y}]

Out[419]= {{2}, {2}}
          {{2}, {2}}
```

```
In[420]:= Column[sDimensionsBoxed[#, 2] & /@ {x, y}]

Out[420]= {{2, 2}, {2, 2}}
          {{2, 2}, {2, 1}}
```

```
In[421]:= Column[sDimensionsBoxed[#, 3] & /@ {x, y}]

Out[421]= {{2, 2, 5}, {2, 2, 5}}
          {{2, 2, 5}, {2, 1, 4}}
```

```
In[422]:= Column[sDimensionsBoxed[#, 4] & /@ {x, y}]

Out[422]= {{2, 2, 5, 4}, {2, 2, 5, 4}}
          {{2, 2, 5, 2}, {2, 1, 4, 0}}
```

Overshooting the maximum level does not matter:

```
In[423]:= Column[sDimensionsBoxed[#, 10] & /@ {x, y}]

Out[423]= {{2, 2, 5, 4}, {2, 2, 5, 4}}
          {{2, 2, 5, 2}, {2, 1, 4, 0}}
```

Applications

One possible application is occupancy rate detection of an nested list (a “ragged” array). This task is resolved by the function sExprOccupancy described in the next Section 1.6.

1.6 sExprOccupancy

```
In[424]:= infoAbout@sExprOccupancy
```

sExprOccupancy[*expr*] gives a rational number
between 0 and 1 (inclusive) that indicates the occupancy rate of *expr*.

```
Attributes[sExprOccupancy] = {Protected, ReadProtected}

SyntaxInformation[sExprOccupancy] = {ArgumentsPattern → {_}}
```

Details

sExprOccupancy[*expr*] returns a rational number between 0 and 1 (inclusive) that indicates the occupancy rate of the expression *expr*.

- Occupancy rate is defined as a ratio of the number of all full depth elements of *expr* to the volume of the minimum bounding box of *expr* as provided by sDimensionsBoxed.
- The argument *expr* is generally a "ragged" array, typically a nested list, but works with any head, not just List.
- Caveat: difficulties with correct output for Association still persist.

Examples

Clearing global symbols:

In[425]:=
```
Clear[a, b, c, d, e, f, f1, f2, f3, g1, g2, h1, h2, x];
ClearAll[printOccupancy];
```

First let us define a helper function to save space:

In[427]:=
```
printOccupancy[x_] :=
 TableForm[{MatrixForm@x, ToString@sDimensionsBoxed[x], sExprOccupancy[x]},
  TableHeadings → {{"expr", "sDimensionsBoxed", "sExprOccupancy"}, None},
  TableDirections → Row]
```

Occupancy of an array (in the sense of "full array") of any depth is always 1:

In[428]:=
```
printOccupancy[Range[3]]
```

Out[428]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	{{3}, {3}}	1

In[429]:=
```
printOccupancy[Array[10 #1 + #2 &, {2, 3}]]
```

Out[429]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{pmatrix}$	{{2, 3}, {2, 3}}	1

In[430]:=
```
printOccupancy[Array[100 #1 + 10 #2 + #3 &, {2, 3, 2}]]
```

Out[430]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} \begin{pmatrix} 111 \\ 112 \end{pmatrix} & \begin{pmatrix} 121 \\ 122 \end{pmatrix} & \begin{pmatrix} 131 \\ 132 \end{pmatrix} \\ \begin{pmatrix} 211 \\ 212 \end{pmatrix} & \begin{pmatrix} 221 \\ 222 \end{pmatrix} & \begin{pmatrix} 231 \\ 232 \end{pmatrix} \end{pmatrix}$	{{2, 3, 2}, {2, 3, 2}}	1

In[431]:=
```
printOccupancy[Array[1000 #1 + 100 #2 + 10 #3 + #4 &, {2, 2, 3, 2}]]
```

Out[431]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} \begin{pmatrix} 1111 & 1112 \\ 1121 & 1122 \\ 1131 & 1132 \end{pmatrix} & \begin{pmatrix} 1211 & 1212 \\ 1221 & 1222 \\ 1231 & 1232 \end{pmatrix} \\ \begin{pmatrix} 2111 & 2112 \\ 2121 & 2122 \\ 2131 & 2132 \end{pmatrix} & \begin{pmatrix} 2211 & 2212 \\ 2221 & 2222 \\ 2231 & 2232 \end{pmatrix} \end{pmatrix}$	{{2, 2, 3, 2}, {2, 2, 3, 2}}	1

Occupancy rate of some expressions gives unexpected results (compare results for the following two expressions):

In[432]:=
```
printOccupancy[Array[10 #1 + #2 &, {2, 3}]]
printOccupancy[Array["a"10 #1+#2 &, {2, 3}]]
```

Out[432]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{pmatrix}$	{{2, 3}, {2, 3}}	1

Out[433]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$	{{2, 3, 2}, {2, 3, 2}}	1

The reason is due to that elements are Subscript's:

In[434]:=
```
Array["a"10 #1+#2 &, {2, 3}] // InputForm
```

Out[434]//InputForm=
```
{{Subscript["a", 11], Subscript["a", 12], Subscript["a", 13]},
 {Subscript["a", 21], Subscript["a", 22], Subscript["a", 23]}}
```

Occupancies of "ragged" arrays (which actually are not arrays, but rather nested lists), are less than 1:

In[435]:=
```
printOccupancy[{{a, a, a}, b, {c}, {d, d}}]
```

Out[435]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} \{a, a, a\} \\ b \\ \{c\} \\ \{d, d\} \end{pmatrix}$	{{4, 3}, {4, 0}}	$\frac{7}{12}$

In[436]:=
```
printOccupancy[{{{a, a}, {b, b}, {c}}, {d, {e}, {f, f}}}]
```

Out[436]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} \{a, a\} & \{b, b\} & \{c\} \\ d & \{e\} & \{f, f\} \end{pmatrix}$	{{2, 3, 2}, {2, 3, 0}}	$\frac{3}{4}$

In[437]:=
```
printOccupancy[{{{{a, a, a}, a, {a}}, {{b, b}, {b, b}}}, {{c, {c}, {c, c}}}}] //
 Style[#, 9] &
```

Out[437]=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} \{\{\{a, a, a\}, a, \{a\}\}, \{\{b, b\}, \{b, b\}\}\} \\ \{\{c, \{c\}, \{c, c\}\}\} \end{pmatrix}$	{{2, 2, 3, 3}, {2, 1, 2, 0}}	$\frac{13}{36}$

In[438]:=
```
printOccupancy[{{{f1[a, a, a], a, f3[a]}, g2[{b, b}, {b, b}]},
   h2[{c, {c}, {c, c}}]}] // Style[#, 8] &
```

Out[438]=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} \{\{f1[a, a, a], a, f3[a]\}, g2[\{b, b\}, \{b, b\}]\} \\ h2[\{c, \{c\}, \{c, c\}\}] \end{pmatrix}$	{{2, 2, 3, 3}, {2, 1, 2, 0}}	$\frac{13}{36}$

In[439]:=
```
printOccupancy[{Range[8], {2}, {3}, {4}, {5}}]
```

Out[439]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\begin{pmatrix} \{1, 2, 3, 4, 5, 6, 7, 8\} \\ \{2\} \\ \{3\} \\ \{4\} \\ \{5\} \end{pmatrix}$	{{5, 8}, {5, 1}}	$\frac{3}{10}$

Occupancy of general expressions can be also determined:

In[440]:=
```
printOccupancy[Hold[Integrate[x^2, {x, 0, 1}]]]
```

Out[440]//TableForm=

expr	sDimensionsBoxed	sExprOccupancy
$\text{Hold}\left[\int_0^1 x^2 \, dx\right]$	{{1, 2, 3}, {1, 2, 2}}	$\frac{5}{6}$

1.7 sDiagonals

In[441]:=
```
infoAbout@sDiagonals
```

sDiagonals[m] gives the list of all diagonals parallel to leading diagonal of the matrix m.
sDiagonals[m, –1] the same for antidiagonals.

```
Attributes[sDiagonals] = {Protected, ReadProtected}
```

```
SyntaxInformation[sDiagonals] = {ArgumentsPattern → {_, _.}}
```

Details

sDiagonals[*m*, *d*] gives the list of all diagonals parallel to leading diagonal of the matrix *m* for $d == 1$ (default), and the list of all antidiagonals parallel to leading antidiagonal of the matrix m for $d == -1$.

- *m* must be a full array of depth greater or equal 2 (i.e., a matrix, cube matrix, hypercube matrix, …), the diagonals are drawn from the top (outmost) level of *m* only. Map can be used to draw diagonals from deeper levels.
- This function is based on the built-in symbol Diagonal.

Examples

Clearing global symbols:

```
In[442]:= Clear[m, a, b, c, d, m, n, μ, λ, c, d];
```

First let us examine a square matrix. This will be our square test matrix:

```
In[443]:= (m = Array["a"10 #1+#2 &, {4, 4}]) // MatrixForm
```

Out[443]//MatrixForm=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Here one can see the diagonals and the antidiagonals of the test matrix:

```
In[444]:= Row[Column /@ {sDiagonals[m], sDiagonals[m, -1]}, Spacer[8]]
```

```
Out[444]= {a14}                  {a11}
          {a13, a24}             {a21, a12}
          {a12, a23, a34}        {a31, a22, a13}
          {a11, a22, a33, a44}   {a41, a32, a23, a14}
          {a21, a32, a43}        {a42, a33, a24}
          {a31, a42}             {a43, a34}
          {a41}                  {a44}
```

```
In[445]:= Row[Column /@ {Plus @@@ sDiagonals[m], Plus @@@ sDiagonals[m, -1]},
           Spacer[8]]
```

```
Out[445]= a14                    a11
          a13 + a24              a12 + a21
          a12 + a23 + a34        a13 + a22 + a31
          a11 + a22 + a33 + a44  a14 + a23 + a32 + a41
          a21 + a32 + a43        a24 + a33 + a42
          a31 + a42              a34 + a43
          a41                    a44
```

The missing positions can be padded:

```
In[446]:= PadRight[sDiagonals[m]] // MatrixForm
```

Out[446]//MatrixForm=

$$\begin{pmatrix} a_{14} & 0 & 0 & 0 \\ a_{13} & a_{24} & 0 & 0 \\ a_{12} & a_{23} & a_{34} & 0 \\ a_{11} & a_{22} & a_{33} & a_{44} \\ a_{21} & a_{32} & a_{43} & 0 \\ a_{31} & a_{42} & 0 & 0 \\ a_{41} & 0 & 0 & 0 \end{pmatrix}$$

First let us examine a non-square matrix. This will be our non-square test matrix:

```
In[447]:= (m = Array["a"10 #1+#2 &, {3, 5}]) // MatrixForm
```

Out[447]//MatrixForm=

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \end{pmatrix}$$

Here one can see the diagonals and the antidiagonals of the test matrix:

```
In[448]:= Row[Column /@ {sDiagonals[m], sDiagonals[m, -1]}, Spacer[8]]
```

```
Out[448]= {a15}              {a11}
          {a14, a25}         {a21, a12}
          {a13, a24, a35}    {a31, a22, a13}
          {a12, a23, a34}    {a32, a23, a14}
          {a11, a22, a33}    {a33, a24, a15}
          {a21, a32}         {a34, a25}
          {a31}              {a35}
```

```
In[449]:= Row[Column /@ {Plus @@@ sDiagonals[m], Plus @@@ sDiagonals[m, -1]},
           Spacer[8]]
```

```
Out[449]= a15                a11
          a14 + a25          a12 + a21
          a13 + a24 + a35    a13 + a22 + a31
          a12 + a23 + a34    a14 + a23 + a32
          a11 + a22 + a33    a15 + a24 + a33
          a21 + a32          a25 + a34
          a31                a35
```

Now let us examine a non-cube 3D (spatial) matrix. This will be our test matrix:

```
In[450]:= (m = Array["a"100 #1+10 #2+#3 &, {4, 5, 2}]) // MatrixForm
```

Out[450]//MatrixForm=

$$\begin{pmatrix} \begin{pmatrix} a_{111} \\ a_{112} \end{pmatrix} & \begin{pmatrix} a_{121} \\ a_{122} \end{pmatrix} & \begin{pmatrix} a_{131} \\ a_{132} \end{pmatrix} & \begin{pmatrix} a_{141} \\ a_{142} \end{pmatrix} & \begin{pmatrix} a_{151} \\ a_{152} \end{pmatrix} \\ \begin{pmatrix} a_{211} \\ a_{212} \end{pmatrix} & \begin{pmatrix} a_{221} \\ a_{222} \end{pmatrix} & \begin{pmatrix} a_{231} \\ a_{232} \end{pmatrix} & \begin{pmatrix} a_{241} \\ a_{242} \end{pmatrix} & \begin{pmatrix} a_{251} \\ a_{252} \end{pmatrix} \\ \begin{pmatrix} a_{311} \\ a_{312} \end{pmatrix} & \begin{pmatrix} a_{321} \\ a_{322} \end{pmatrix} & \begin{pmatrix} a_{331} \\ a_{332} \end{pmatrix} & \begin{pmatrix} a_{341} \\ a_{342} \end{pmatrix} & \begin{pmatrix} a_{351} \\ a_{352} \end{pmatrix} \\ \begin{pmatrix} a_{411} \\ a_{412} \end{pmatrix} & \begin{pmatrix} a_{421} \\ a_{422} \end{pmatrix} & \begin{pmatrix} a_{431} \\ a_{432} \end{pmatrix} & \begin{pmatrix} a_{441} \\ a_{442} \end{pmatrix} & \begin{pmatrix} a_{451} \\ a_{452} \end{pmatrix} \end{pmatrix}$$

Here one can see the diagonals and the antidiagonals of the test matrix:

In[451]:=
```
Column[Column /@ {sDiagonals[m], sDiagonals[m, -1]}, Spacings → 1]
```

Out[451]=
```
{{a151, a152}}
{{a141, a142}, {a251, a252}}
{{a131, a132}, {a241, a242}, {a351, a352}}
{{a121, a122}, {a231, a232}, {a341, a342}, {a451, a452}}
{{a111, a112}, {a221, a222}, {a331, a332}, {a441, a442}}
{{a211, a212}, {a321, a322}, {a431, a432}}
{{a311, a312}, {a421, a422}}
{{a411, a412}}

{{a111, a112}}
{{a211, a212}, {a121, a122}}
{{a311, a312}, {a221, a222}, {a131, a132}}
{{a411, a412}, {a321, a322}, {a231, a232}, {a141, a142}}
{{a421, a422}, {a331, a332}, {a241, a242}, {a151, a152}}
{{a431, a432}, {a341, a342}, {a251, a252}}
{{a441, a442}, {a351, a352}}
{{a451, a452}}
```

Applications

Let us consider the following approximate representations of two functions f, g in terms of the Chebyshev 1st kind (T) orthgonal polynomial coefficients a_j, b_j, $j = 0, \ \ldots, \ N-1$,

$$f(x) = -\frac{1}{2} a_0 + \sum_{j=0}^{N-1} a_j \, T_j(x) = \frac{1}{2} a_0 \, T_0(x) + \sum_{j=1}^{N-1} a_j \, T_j(x) \, ,$$

$$g(x) = -\frac{1}{2} b_0 + \sum_{j=0}^{N-1} b_j \, T_j(x) = \frac{1}{2} b_0 \, T_0(x) + \sum_{j=1}^{N-1} b_j \, T_j(x) \, .$$

Simplifying the notation by relabeling

$$\frac{1}{2} a_0 \to a_0, \ \frac{1}{2} b_0 \to b_0$$

(leaving the remaining coefficients unchanged) and using the identity

$$2 \, T_j(x) \, T_k(x) = T_{j+k}(x) + T_{|j-k|}(x) \, ,$$

it can easily be shown that the product of both functions reads

$$f(x) \, g(x) = \frac{1}{2} \sum_{j,k=0}^{N-1} a_j \, b_k \, T_{j+k}(x) + \frac{1}{2} \sum_{j,k=0}^{N-1} a_j \, b_k \, T_{|j-k|}(x) \, .$$

The first and the second term on the right hand side can be rewritten as

$$\sum_{j=0}^{2N-2} c_j \, T_j(x) \ \text{ and } \ \sum_{j=0}^{N-1} d_j \, T_j(x) \, ,$$

respectively, where the coefficients $c_j,\ j = 0,\ \ldots,\ 2\,N-2$ are the sums in all $2\,N-1$ antidiagonals of the matrix

$$\begin{pmatrix} a_0\,b_0 & a_0\,b_1 & a_0\,b_2 & a_0\,b_3 & \cdots & a_0\,b_{N-3} & a_0\,b_{N-2} & a_0\,b_{N-1} \\ a_1\,b_0 & a_1\,b_1 & a_1\,b_2 & a_1\,b_3 & \cdots & a_1\,b_{N-3} & a_1\,b_{N-2} & a_1\,b_{N-1} \\ a_2\,b_0 & a_2\,b_1 & a_2\,b_2 & a_2\,b_3 & \cdots & a_2\,b_{N-3} & a_2\,b_{N-2} & a_2\,b_{N-1} \\ a_3\,b_0 & a_3\,b_1 & a_3\,b_2 & a_3\,b_3 & \cdots & a_3\,b_{N-3} & a_3\,b_{N-2} & a_3\,b_{N-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{N-3}\,b_0 & a_{N-3}\,b_1 & a_{N-3}\,b_2 & a_{N-3}\,b_3 & \cdots & a_{N-3}\,b_{N-3} & a_{N-3}\,b_{N-2} & a_{N-3}\,b_{N-1} \\ a_{N-2}\,b_0 & a_{N-2}\,b_1 & a_{N-2}\,b_2 & a_{N-2}\,b_3 & \cdots & a_{N-2}\,b_{N-3} & a_{N-2}\,b_{N-2} & a_{N-2}\,b_{N-1} \\ a_{N-1}\,b_0 & a_{N-1}\,b_1 & a_{N-1}\,b_2 & a_{N-1}\,b_3 & \cdots & a_{N-1}\,b_{N-3} & a_{N-1}\,b_{N-2} & a_{N-1}\,b_{N-1} \end{pmatrix}$$

Similarly, the coefficients $d_j,\ j = 0,\ \ldots,\ N-1$ represent N symmetrized sums in the diagonals of the above matrix, i.e., in the leading diagonal, total in the 1st diagonals above and below, total in the 2nd diagonals above and below, …, total in the $(N-1)$-th diagonals above and below. It can therefore be seen that this procedure is easily implementable with the help of sDiagonals:

```
In[452]:= (m = With[{n = 8}, Outer[Times, Array[a_# &, n, 0], Array[b_# &, n, 0]]]) //
          MatrixForm
```

Out[452]//MatrixForm=

$$\begin{pmatrix} a_0\,b_0 & a_0\,b_1 & a_0\,b_2 & a_0\,b_3 & a_0\,b_4 & a_0\,b_5 & a_0\,b_6 & a_0\,b_7 \\ a_1\,b_0 & a_1\,b_1 & a_1\,b_2 & a_1\,b_3 & a_1\,b_4 & a_1\,b_5 & a_1\,b_6 & a_1\,b_7 \\ a_2\,b_0 & a_2\,b_1 & a_2\,b_2 & a_2\,b_3 & a_2\,b_4 & a_2\,b_5 & a_2\,b_6 & a_2\,b_7 \\ a_3\,b_0 & a_3\,b_1 & a_3\,b_2 & a_3\,b_3 & a_3\,b_4 & a_3\,b_5 & a_3\,b_6 & a_3\,b_7 \\ a_4\,b_0 & a_4\,b_1 & a_4\,b_2 & a_4\,b_3 & a_4\,b_4 & a_4\,b_5 & a_4\,b_6 & a_4\,b_7 \\ a_5\,b_0 & a_5\,b_1 & a_5\,b_2 & a_5\,b_3 & a_5\,b_4 & a_5\,b_5 & a_5\,b_6 & a_5\,b_7 \\ a_6\,b_0 & a_6\,b_1 & a_6\,b_2 & a_6\,b_3 & a_6\,b_4 & a_6\,b_5 & a_6\,b_6 & a_6\,b_7 \\ a_7\,b_0 & a_7\,b_1 & a_7\,b_2 & a_7\,b_3 & a_7\,b_4 & a_7\,b_5 & a_7\,b_6 & a_7\,b_7 \end{pmatrix}$$

The coefficients $c_j,\ j = 0,\ \ldots,\ 15$ then can easily be computed as

```
In[453]:= (c = Total /@ sDiagonals[m, -1]) // Column
```

Out[453]=

$$\begin{aligned} &a_0\,b_0 \\ &a_1\,b_0 + a_0\,b_1 \\ &a_2\,b_0 + a_1\,b_1 + a_0\,b_2 \\ &a_3\,b_0 + a_2\,b_1 + a_1\,b_2 + a_0\,b_3 \\ &a_4\,b_0 + a_3\,b_1 + a_2\,b_2 + a_1\,b_3 + a_0\,b_4 \\ &a_5\,b_0 + a_4\,b_1 + a_3\,b_2 + a_2\,b_3 + a_1\,b_4 + a_0\,b_5 \\ &a_6\,b_0 + a_5\,b_1 + a_4\,b_2 + a_3\,b_3 + a_2\,b_4 + a_1\,b_5 + a_0\,b_6 \\ &a_7\,b_0 + a_6\,b_1 + a_5\,b_2 + a_4\,b_3 + a_3\,b_4 + a_2\,b_5 + a_1\,b_6 + a_0\,b_7 \\ &a_7\,b_1 + a_6\,b_2 + a_5\,b_3 + a_4\,b_4 + a_3\,b_5 + a_2\,b_6 + a_1\,b_7 \\ &a_7\,b_2 + a_6\,b_3 + a_5\,b_4 + a_4\,b_5 + a_3\,b_6 + a_2\,b_7 \\ &a_7\,b_3 + a_6\,b_4 + a_5\,b_5 + a_4\,b_6 + a_3\,b_7 \\ &a_7\,b_4 + a_6\,b_5 + a_5\,b_6 + a_4\,b_7 \\ &a_7\,b_5 + a_6\,b_6 + a_5\,b_7 \\ &a_7\,b_6 + a_6\,b_7 \\ &a_7\,b_7 \end{aligned}$$

while the coefficients $d_j,\ j = 0,\ \ldots,\ 7$ are

In[454]:=
```
(d = Module[{s = Total /@ sDiagonals[m], l = Length@m},
    Reverse@MapAt[Simplify, Take[s + Reverse@s, l] /. {μ___, λ_} → {μ, λ / 2},
      l]]) // Column // Style[#, 8] &
```

Out[454]=

$$a_0 b_0 + a_1 b_1 + a_2 b_2 + a_3 b_3 + a_4 b_4 + a_5 b_5 + a_6 b_6 + a_7 b_7$$
$$a_1 b_0 + a_0 b_1 + a_2 b_1 + a_1 b_2 + a_3 b_2 + a_2 b_3 + a_4 b_3 + a_3 b_4 + a_5 b_4 + a_4 b_5 + a_6 b_5 + a_5 b_6 + a_7 b_6 + a_6 b_7$$
$$a_2 b_0 + a_3 b_1 + a_0 b_2 + a_4 b_2 + a_1 b_3 + a_5 b_3 + a_2 b_4 + a_6 b_4 + a_3 b_5 + a_7 b_5 + a_4 b_6 + a_5 b_7$$
$$a_3 b_0 + a_4 b_1 + a_5 b_2 + a_0 b_3 + a_6 b_3 + a_1 b_4 + a_7 b_4 + a_2 b_5 + a_3 b_6 + a_4 b_7$$
$$a_4 b_0 + a_5 b_1 + a_6 b_2 + a_7 b_3 + a_0 b_4 + a_1 b_5 + a_2 b_6 + a_3 b_7$$
$$a_5 b_0 + a_6 b_1 + a_7 b_2 + a_0 b_5 + a_1 b_6 + a_2 b_7$$
$$a_6 b_0 + a_7 b_1 + a_0 b_6 + a_1 b_7$$
$$a_7 b_0 + a_0 b_7$$

2

Signal analysis

This chapter is mostly devoted to functions facilitating sampling, spectral analysis, convolving and related topics that fall into the field of signal analysis and processing. Conversely, some of them may ambiguously fall into Chapter 1.

2.1 sSamplingSpan

In[455]:=
```
infoAbout@sSamplingSpan
```

sSamplingSpan[m] gives list of $n = |m|$ indices: {0, …, $n-1$} if $m > 0$, wrapped around {$-\lfloor n/2 \rfloor$, …, 0, …, $\lceil n/2 \rceil - 1$} if $m < 0$.

```
Attributes[sSamplingSpan] = {Protected, ReadProtected}
```

```
SyntaxInformation[sSamplingSpan] = {ArgumentsPattern → {_}}
```

Details

sSamplingSpan[m] generates a zero-offset list of m integer sampling indices

$$\{0,\ \ldots,\ m-1\}$$

if m is a positive integer, and a wrapped around list of $n = |m|$ integer sampling indices

$$\{-\lfloor n/2 \rfloor,\ \ldots,\ 0,\ \ldots,\ \lceil n/2 \rceil - 1\}$$

if m is a negative integer.

- Index 0 that corresponds to zero time or frequency is in position $\lfloor n/2 \rfloor + 1$, counted from 1.
- Advantageously applicable with sRotateHalfLength.

- To get real time or frequency ticks, multiply the list with sampling interval or $(1/n)\times$ sample rate, respectively.
- Null is returned in case of improper arguments.

Examples

Clearing global symbols:

```
In[456]:= Clear[Δ, n, fs, n];
```

Generating zero-offset list of sampling indices is trivial and can easily be substituted with the built-in symbol Range:

```
In[457]:= Table[Row@{n, Spacer[15], sSamplingSpan[n]}, {n, 8}] // Column

Out[457]=
1   {0}
2   {0, 1}
3   {0, 1, 2}
4   {0, 1, 2, 3}
5   {0, 1, 2, 3, 4}
6   {0, 1, 2, 3, 4, 5}
7   {0, 1, 2, 3, 4, 5, 6}
8   {0, 1, 2, 3, 4, 5, 6, 7}
```

On the other hand, sSamplingSpan comes handy when creating a wrapped around list of sampling indices:

```
In[458]:= Table[Row@{n, Spacer[15], sSamplingSpan[-n]}, {n, 8}] // Column

Out[458]=
1   {0}
2   {-1, 0}
3   {-1, 0, 1}
4   {-2, -1, 0, 1}
5   {-2, -1, 0, 1, 2}
6   {-3, -2, -1, 0, 1, 2}
7   {-3, -2, -1, 0, 1, 2, 3}
8   {-4, -3, -2, -1, 0, 1, 2, 3}
```

Applications

Determine sampling points for sampling period Δ set to a particular value (here 0.00125 seconds):

```
In[459]:= Δ = 0.000125; Δ sSamplingSpan[n = 16]

Out[459]= {0., 0.000125, 0.00025, 0.000375, 0.0005, 0.000625, 0.00075, 0.000875,
  0.001, 0.001125, 0.00125, 0.001375, 0.0015, 0.001625, 0.00175, 0.001875}
```

The sampling frequency is

$$f_s = \frac{1}{\Delta} = 8000\ \text{Hz}\,,$$

and the corresponding discrete frequencies [in units of Hz] for the DFT are

In[460]:=
```
Print[fs = 1/Δ]; (fs/n) sSamplingSpan[-n]
```

```
8000.
```

Out[460]=
```
{-4000., -3500., -3000., -2500., -2000., -1500., -1000.,
 -500., 0., 500., 1000., 1500., 2000., 2500., 3000., 3500.}
```

2.2 sCircularlySample

In[461]:=
```
infoAbout@sCircularlySample
```

sCircularlySample[*ival*, *n*] yields *n*–element vector of
sampling points evenly spaced around a circular interval *ival* = {a, b}.

```
Attributes[sCircularlySample] = {Protected, ReadProtected}

Options[sCircularlySample] = {Debug → False}

SyntaxInformation[sCircularlySample] =
 {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

Details

sCircularlySample[*ival*, *n*, *opts*] yields a list representing the vector $\mathcal{S}$ of n sampling points evenly spaced around a circular interval $\langle a, b)$ (whose endpoints a, b are mutually identified) specified by *ival* = {*a*, *b*} or Interval[{*a*, *b*}].

- If $0 \in ival$, the first element of the resulting vector $\mathcal{S}$ coincides with 0.
- If $0 \notin ival$, the first element of the resulting vector $\mathcal{S}$ coincides with a unique periodic translation $\mathcal{T}$ of 0 by a suitable (unique) multiple of the interval length $b - a$ into *ival*.
- The remaining elements of the resulting vector $\mathcal{S}$ (sampling points) proceed rightward receding the first one and approaching the right endpoint b; after reaching a maximum they are wrapped back to the minimum close the left endpoint a, eventually approaching the first element from below. This way of wrapping is suitable for passing to the DFT.
- A function *f* defined on *ival* can be sampled by mapping *f* /@$\mathcal{S}$ (or simply by *f* [$\mathcal{S}$] in case of the Listable attribute). The resulting samples represent periodization of the function *f*.
- Normal behavior outlined above applies for a positive integer n. Specifying a negative integer n causes using $|n|$ in place of n, and returning an association in the form

  $$\langle \left| \mathcal{I} \to \mathcal{S}, \mathcal{T}^{-1}[\mathcal{I}] \to \mathcal{T}^{-1}[\mathcal{S}] \right| \rangle$$

  to be returned (here $\mathcal{I} \equiv ival$ and $\mathcal{S}$ denote the original interval *ival* and the vector of samples, respectively; $\mathcal{T}^{-1}[\mathcal{I}]$ and $\mathcal{T}^{-1}[\mathcal{S}]$ denote their respective inverse unique periodic translations). If $\mathcal{I}$ coincides (in the sense of being "same") with $\mathcal{T}^{-1}[\mathcal{I}]$ (hence $\mathcal{S}$ coincides with $\mathcal{T}^{-1}[\mathcal{S}]$), only association

$\langle|\mathcal{I} \to \mathcal{S}|\rangle$

is returned.

- For a negative integer n, mapping of a function f defined on *ival* is carried out as follows:

$$\text{Map}\left[f,\ \langle\left|\mathcal{I} \to \mathcal{S},\ \mathcal{T}^{-1}[\mathcal{I}] \to \mathcal{T}^{-1}[\mathcal{S}]\right|\rangle,\ \{2\}\right]$$

- The "Debug" → True option (default is False) causes printing a table and a number line plot elucidating how the function works.

Examples

Clearing global symbols:

```
In[462]:= Clear[cosT, cosF, cosT2, cosF2];
```

In the following examples, we exceptionally switch on the "Debug" option due to illustrative revealing how the function operates. Normal output you should use looks like

```
In[463]:= sCircularlySample[{15, 18}, 8]
```

$$\text{Out[463]=}\ \left\{15,\ \frac{123}{8},\ \frac{63}{4},\ \frac{129}{8},\ \frac{33}{2},\ \frac{135}{8},\ \frac{69}{4},\ \frac{141}{8}\right\}$$

or

```
In[464]:= sCircularlySample[{15, 18}, -8]
```

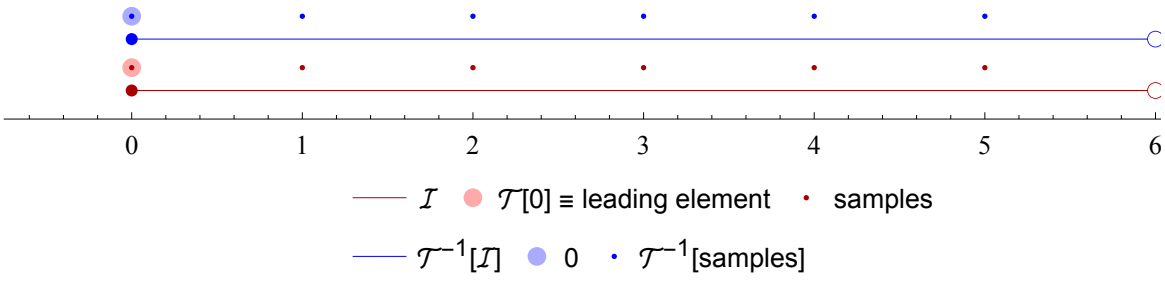
$$\text{Out[464]=}\ \langle\left|\{15, 18\} \to \left\{15,\ \frac{123}{8},\ \frac{63}{4},\ \frac{129}{8},\ \frac{33}{2},\ \frac{135}{8},\ \frac{69}{4},\ \frac{141}{8}\right\},\right.$$
$$\left.\{0, 3\} \to \left\{0,\ \frac{3}{8},\ \frac{3}{4},\ \frac{9}{8},\ \frac{3}{2},\ \frac{15}{8},\ \frac{9}{4},\ \frac{21}{8}\right\}\right|\rangle$$

Interval left endpoint is zero

If the left endpoint $a = 0$, the interval is evenly sampled starting from $a = 0$ with sampling interval $\Delta = (b - a)/|n|$. The elements of the sample vector $\mathcal{S}$ form a growing sequence with the greatest (last) element at a distance Δ from the right endpoint b. The red (blue) stuff refer to the original (translated) interval; detailed legend is printed below each number line plot.

```
In[465]:= sCircularlySample[{0, 6}, -6, "Debug" → True]
```

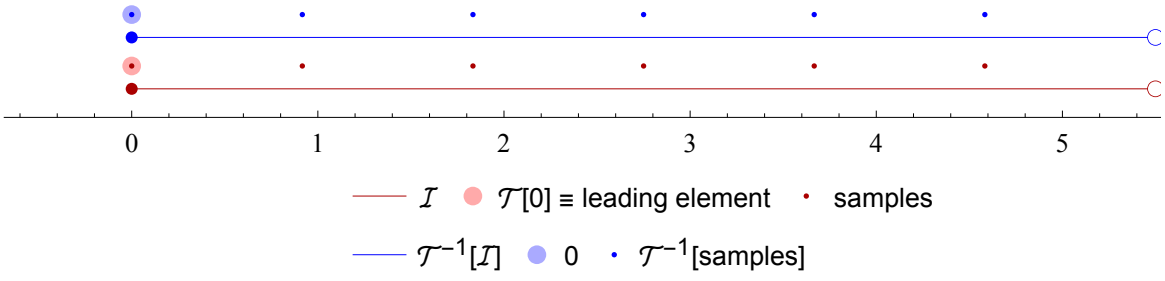
# of samples: {n, p ≡ |n|}	{-6, 6}
interval length: b - a	6
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	0
samp. interval: Δ ≡ (b-a)/p	1
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{0, 5}
$\mathcal{I} \equiv \langle a, b)$	{0, 6}
samples	{0, 1, 2, 3, 4, 5}
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	{0, 6}
$\mathcal{T}^{-1}$[samples]	{0, 1, 2, 3, 4, 5}



Out[465]= ‹| {0, 6} → {0, 1, 2, 3, 4, 5} |›

In[466]:= **sCircularlySample[{0, 11 / 2}, -6, "Debug" → True]**

# of samples: {n, p ≡ |n|}	{-6, 6}
interval length: b - a	$\frac{11}{2}$
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	0
samp. interval: Δ ≡ (b-a)/p	$\frac{11}{12}$
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{0, 5}
$\mathcal{I} \equiv \langle a, b)$	$\left\{0, \frac{11}{2}\right\}$
samples	$\left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, \frac{11}{3}, \frac{55}{12}\right\}$
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	$\left\{0, \frac{11}{2}\right\}$
$\mathcal{T}^{-1}$[samples]	$\left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, \frac{11}{3}, \frac{55}{12}\right\}$

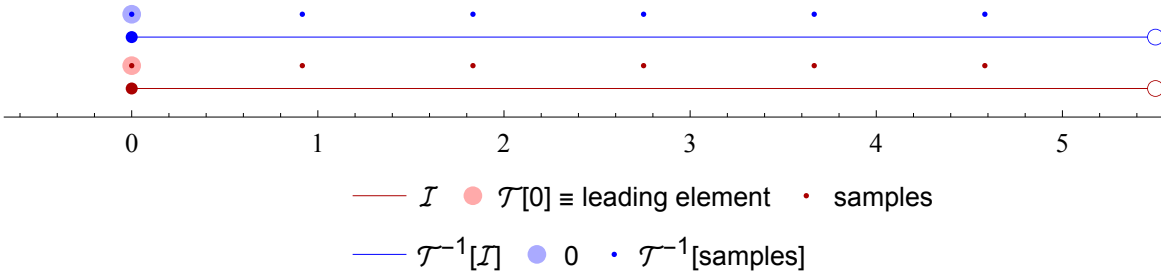


Out[466]= $\left\langle \left| \left\{0, \frac{11}{2}\right\} \to \left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, \frac{11}{3}, \frac{55}{12}\right\} \right| \right\rangle$

Although for $a = 0$ the interval $\mathcal{I}$ coincides (in the mathematical sense) with $\mathcal{T}^{-1}[\mathcal{I}]$, using inexact numbers may cause returning an association with both intervals since their "sameness" in the sense of *Mathematica* is invalid.

In[467]:= **sCircularlySample[{0, 5.5}, -6, "Debug" → True]**

```
# of samples: {n, p ≡ |n|}          | {-6, 6}
interval length: b - a              | 5.5
transl. 𝒯 factor: k ≡ ⌈a/(b-a)⌉     | 0
samp. interval: Δ ≡ (b-a)/p         | 0.916667
{j_min, j_max} ≡ {⌈a_0/Δ⌉, ⌈b_0/Δ⌉-1} | {0, 5}
ℐ ≡ ⟨a, b)                          | {0, 5.5}
samples                             | {0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333
𝒯^-1[ℐ] ≡ ⟨a_0, b_0)                | {0., 5.5}
𝒯^-1[samples]                       | {0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333
```



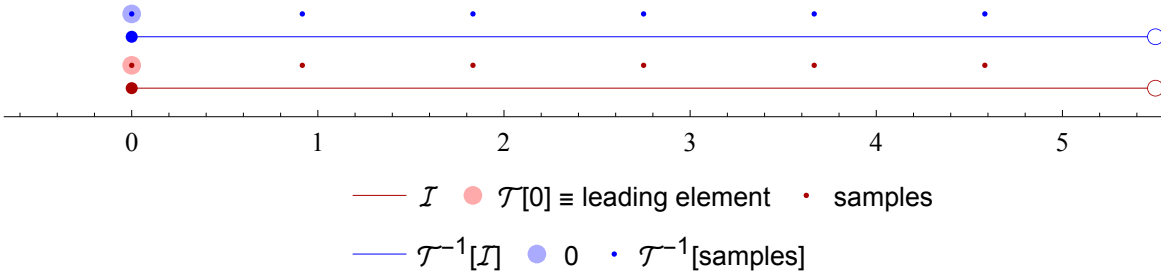
$\mathcal{I}$ $\mathcal{T}[0]$ ≡ leading element samples

$\mathcal{T}^{-1}[\mathcal{I}]$ 0 $\mathcal{T}^{-1}$[samples]

Out[467]= ⟨| {0, 5.5} → {0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333},
{0., 5.5} → {0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333} |⟩

In[468]:= **sCircularlySample[{0., 11 / 2}, -6, "Debug" → True]**

```
# of samples: {n, p ≡ |n|}          | {-6, 6}
interval length: b - a              | 5.5
transl. 𝒯 factor: k ≡ ⌈a/(b-a)⌉     | 0
samp. interval: Δ ≡ (b-a)/p         | 0.916667
{j_min, j_max} ≡ {⌈a_0/Δ⌉, ⌈b_0/Δ⌉-1} | {0, 5}
ℐ ≡ ⟨a, b)                          | {0., 11/2}
samples                             | {0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333
𝒯^-1[ℐ] ≡ ⟨a_0, b_0)                | {0., 5.5}
𝒯^-1[samples]                       | {0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333
```



$\mathcal{I}$ $\mathcal{T}[0]$ ≡ leading element samples

$\mathcal{T}^{-1}[\mathcal{I}]$ 0 $\mathcal{T}^{-1}$[samples]

Out[468]= $\left\langle\left| \left\{0., \frac{11}{2}\right\} \to \{0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333\},\right.\right.$
$\left.\left. \{0., 5.5\} \to \{0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333\} \right|\right\rangle$

Zero is the internal point of the interval

If $0 \in (a, b)$, the interval $\mathcal{I}$ is evenly sampled starting from $0 > a$ with sampling interval $\Delta = (b-a)/|n|$. The elements of the sample vector $\mathcal{S}$ do not form a growing sequence; instead,

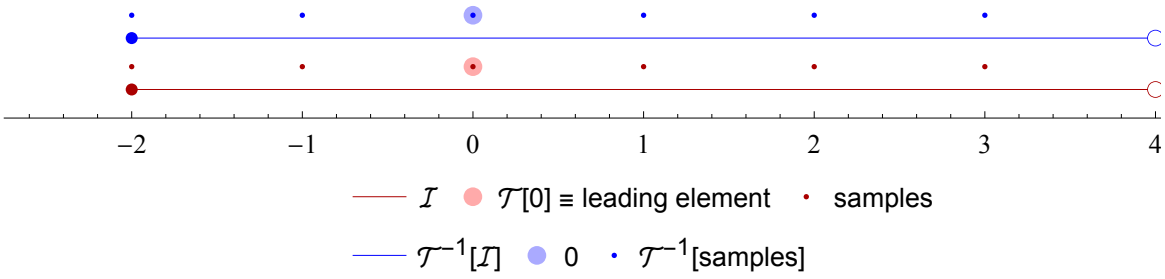
the elements of $\mathcal{S}$ proceed rightward receding zero and approaching the right endpoint b; after reaching a maximum they are wrapped back to the minimum close the left endpoint a, eventually approaching zero from below. Consequently, the equality

$$(b - \text{Max}[\mathcal{S}]) + (\text{Min}[\mathcal{S}] - a) = \Delta$$

applies, and is tested at the end of each subsequent example (see the number next to the resulting vector of samples).

```
In[469]:= Module[{a = -2, b = 4, n = 6, r}, r = sCircularlySample[{a, b}, n, "Debug" → True];
           Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

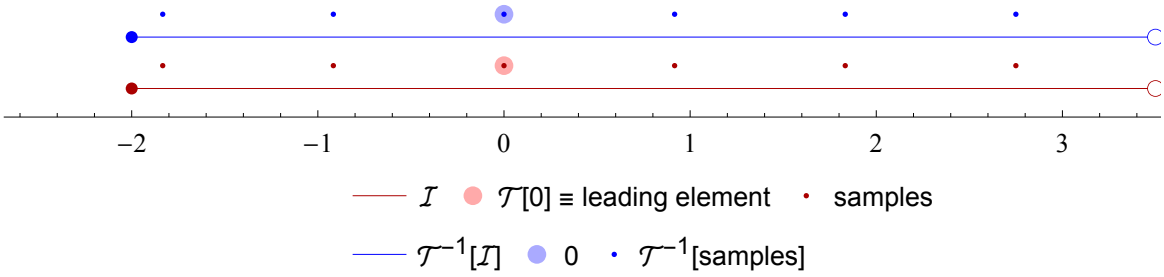
# of samples: {n, p ≡ |n|}	{6, 6}
interval length: b - a	6
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	0
samp. interval: Δ ≡ (b-a)/p	1
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{-2, 3}
$\mathcal{I} \equiv \langle a, b)$	{-2, 4}
samples	{0, 1, 2, 3, -2, -1}
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	{-2, 4}
$\mathcal{T}^{-1}$[samples]	{0, 1, 2, 3, -2, -1}




```
Out[469]= {0, 1, 2, 3, -2, -1}   1
```

```
In[470]:= Module[{a = -2, b = 7 / 2, n = 6, r},
           r = sCircularlySample[{a, b}, n, "Debug" → True];
           Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

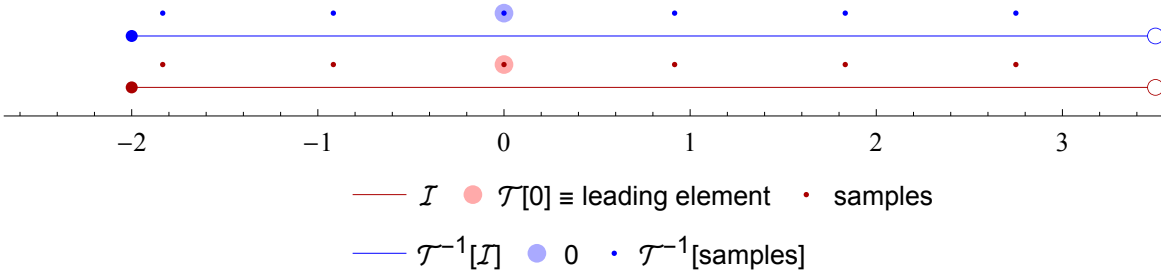
# of samples: {n, p ≡ |n|}	{6, 6}
interval length: b - a	$\frac{11}{2}$
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	0
samp. interval: Δ ≡ (b-a)/p	$\frac{11}{12}$
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{-2, 3}
$\mathcal{I} \equiv \langle a, b)$	$\left\{-2, \frac{7}{2}\right\}$
samples	$\left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, -\frac{11}{6}, -\frac{11}{12}\right\}$
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	$\left\{-2, \frac{7}{2}\right\}$
$\mathcal{T}^{-1}$[samples]	$\left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, -\frac{11}{6}, -\frac{11}{12}\right\}$



Out[470]= $\left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, -\frac{11}{6}, -\frac{11}{12}\right\}$ $\frac{11}{12}$

In[471]:=
```
Module[{a = -2, b = 3.5, n = 6, r},
 r = sCircularlySample[{a, b}, n, "Debug" → True];
 Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

# of samples: {n, p ≡ |n|}	{6, 6}
interval length: b - a	5.5
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	0
samp. interval: Δ ≡ (b-a)/p	0.916667
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{-2, 3}
$\mathcal{I} \equiv \langle a, b)$	{-2, 3.5}
samples	{0., 0.916667, 1.83333, 2.75, -1.83333, -0.916
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	{-2., 3.5}
$\mathcal{T}^{-1}$[samples]	{0., 0.916667, 1.83333, 2.75, -1.83333, -0.916



Out[471]= {0., 0.916667, 1.83333, 2.75, -1.83333, -0.916667} 0.916667

Although also in the case $0 \in (a, b)$ the interval $\mathcal{I}$ coincides (in the mathematical sense) with

$\mathcal{T}^{-1}[\mathcal{I}]$, using inexact numbers may cause returning an association with both intervals since their "sameness" in the sense of *Mathematica* is invalid.

```
In[472]:= sCircularlySample[{-2, 7 / 2}, -6]
```

$$\text{Out[472]= } \left\langle \left| \left\{-2, \frac{7}{2}\right\} \to \left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, -\frac{11}{6}, -\frac{11}{12}\right\} \right| \right\rangle$$

```
In[473]:= sCircularlySample[{-2, 3.5}, -6]
```

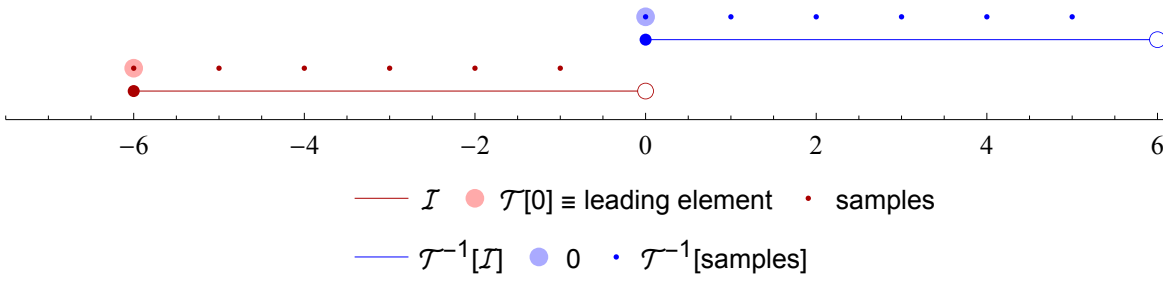
```
Out[473]= ⟨| {-2, 3.5} → {0., 0.916667, 1.83333, 2.75, -1.83333, -0.916667},
            {-2., 3.5} → {0., 0.916667, 1.83333, 2.75, -1.83333, -0.916667} |⟩
```

Interval right endpoint is zero

If the right endpoint $b = 0$, the interval is evenly sampled starting from the left endpoint a with sampling interval $\Delta = (b - a)/|n|$. The elements of the sample vector $\mathcal{S}$ form a growing sequence with the greatest (last) element at a distance Δ from the right endpoint $b = 0$. Effectively the interval is translated by its length to the right using translation $\mathcal{T}^{-1}$ (see case $a = 0$), sampled, and then the samples are translated back using translation $\mathcal{T}$.

```
In[474]:= Module[{a = -6, b = 0, n = 6, r}, r = sCircularlySample[{a, b}, n, "Debug" → True];
           Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

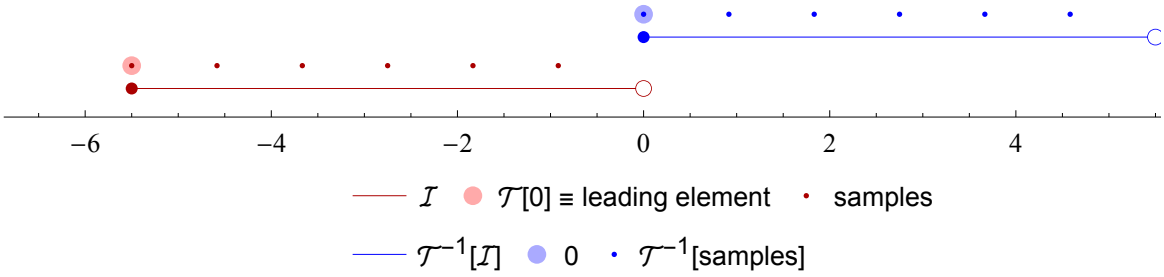
```
# of samples: {n, p ≡ |n|}                  | {6, 6}
interval length: b - a                      | 6
transl. 𝒯 factor: k ≡ ⌈a/(b-a)⌉              | -1
samp. interval: Δ ≡ (b-a)/p                 | 1
{j_min, j_max} ≡ {⌈a_0/Δ⌉, ⌈b_0/Δ⌉-1}       | {0, 5}
ℐ ≡ ⟨a, b)                                  | {-6, 0}
samples                                     | {-6, -5, -4, -3, -2, -1}
𝒯^-1[ℐ] ≡ ⟨a_0, b_0)                        | {0, 6}
𝒯^-1[samples]                               | {0, 1, 2, 3, 4, 5}
```




```
Out[474]= {-6, -5, -4, -3, -2, -1}   1
```

```
In[475]:= Module[{a = -11 / 2, b = 0, n = 6, r},
           r = sCircularlySample[{a, b}, n, "Debug" → True];
           Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

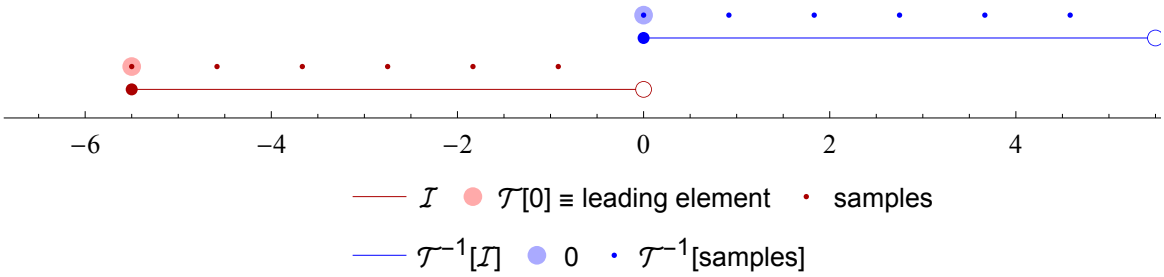
# of samples: {n, p ≡ |n|}	{6, 6}
interval length: b - a	$\frac{11}{2}$
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	-1
samp. interval: Δ ≡ (b-a)/p	$\frac{11}{12}$
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{0, 5}
$\mathcal{I} \equiv \langle a, b)$	$\left\{-\frac{11}{2}, 0\right\}$
samples	$\left\{-\frac{11}{2}, -\frac{55}{12}, -\frac{11}{3}, -\frac{11}{4}, -\frac{11}{6}, -\frac{11}{12}\right\}$
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	$\left\{0, \frac{11}{2}\right\}$
$\mathcal{T}^{-1}$[samples]	$\left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, \frac{11}{3}, \frac{55}{12}\right\}$



Out[475]= $\left\{-\frac{11}{2}, -\frac{55}{12}, -\frac{11}{3}, -\frac{11}{4}, -\frac{11}{6}, -\frac{11}{12}\right\}$ $\frac{11}{12}$

In[476]:=
```
Module[{a = -5.5, b = 0, n = 6, r},
 r = sCircularlySample[{a, b}, n, "Debug" → True];
 Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

# of samples: {n, p ≡ |n|}	{6, 6}
interval length: b - a	5.5
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	-1
samp. interval: Δ ≡ (b-a)/p	0.916667
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{0, 5}
$\mathcal{I} \equiv \langle a, b)$	{-5.5, 0}
samples	{-5.5, -4.58333, -3.66667, -2.75, -1.83333, -
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	{0., 5.5}
$\mathcal{T}^{-1}$[samples]	{0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333



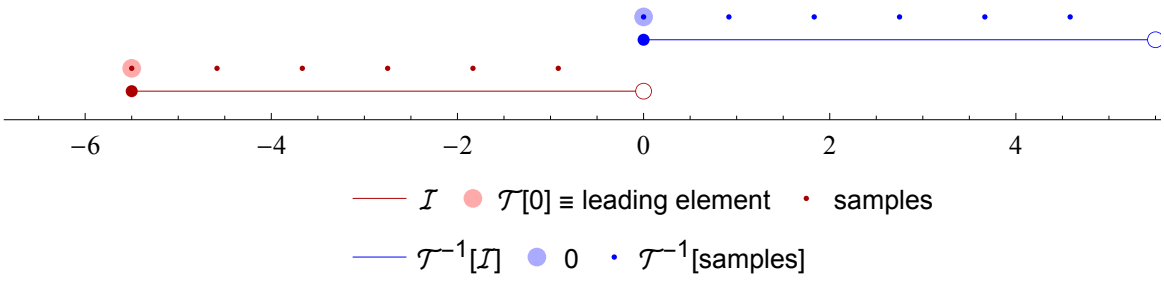
Out[476]= {-5.5, -4.58333, -3.66667, -2.75, -1.83333, -0.916667} 0.916667

In the case of $b = 0$ the interval $\mathcal{I}$ never coincides (in the mathematical sense) with $\mathcal{T}^{-1}[\mathcal{I}]$,

consequently, associations of two elements is always returned.

In[477]:=
```
sCircularlySample[{-11 / 2, 0}, -6, "Debug" → True]
```

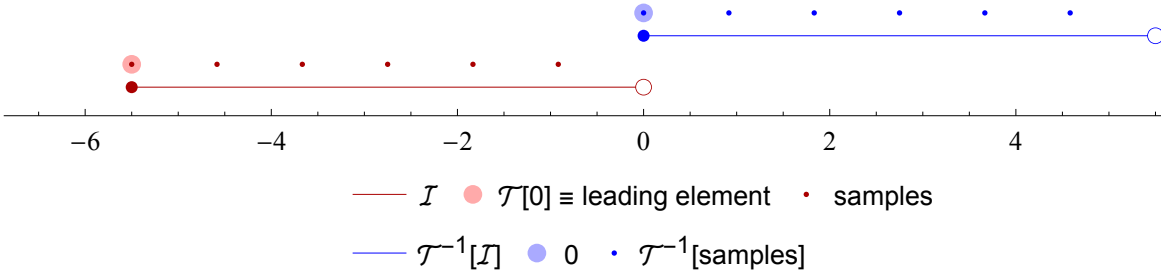
# of samples: {n, p ≡ |n|}	{-6, 6}
interval length: b - a	$\frac{11}{2}$
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	-1
samp. interval: Δ ≡ (b-a)/p	$\frac{11}{12}$
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{0, 5}
$\mathcal{I} \equiv \langle a, b \rangle$	$\left\{-\frac{11}{2}, 0\right\}$
samples	$\left\{-\frac{11}{2}, -\frac{55}{12}, -\frac{11}{3}, -\frac{11}{4}, -\frac{11}{6}, -\frac{11}{12}\right\}$
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0 \rangle$	$\left\{0, \frac{11}{2}\right\}$
$\mathcal{T}^{-1}$[samples]	$\left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, \frac{11}{3}, \frac{55}{12}\right\}$



Out[477]=
$$\left\langle \left| \left\{-\frac{11}{2}, 0\right\} \to \left\{-\frac{11}{2}, -\frac{55}{12}, -\frac{11}{3}, -\frac{11}{4}, -\frac{11}{6}, -\frac{11}{12}\right\}, \left\{0, \frac{11}{2}\right\} \to \left\{0, \frac{11}{12}, \frac{11}{6}, \frac{11}{4}, \frac{11}{3}, \frac{55}{12}\right\} \right| \right\rangle$$

In[478]:=
```
sCircularlySample[{-11 / 2, 0.}, -6, "Debug" → True]
```

# of samples: {n, p ≡ |n|}	{-6, 6}
interval length: b - a	5.5
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	-1
samp. interval: Δ ≡ (b-a)/p	0.916667
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{0, 5}
$\mathcal{I} \equiv \langle a, b \rangle$	$\left\{-\frac{11}{2}, 0.\right\}$
samples	{-5.5, -4.58333, -3.66667, -2.75, -1.83333, -
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0 \rangle$	{0., 5.5}
$\mathcal{T}^{-1}$[samples]	{0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333



Out[478]= $\langle \left| \left\{-\frac{11}{2}, 0.\right\} \to \{-5.5, -4.58333, -3.66667, -2.75, -1.83333, -0.916667\}, \{0., 5.5\} \to \{0., 0.916667, 1.83333, 2.75, 3.66667, 4.58333\} \right| \rangle$

Interval closure does not contain zero

$$0 \notin \overline{\mathcal{I}} = \langle a, b \rangle$$

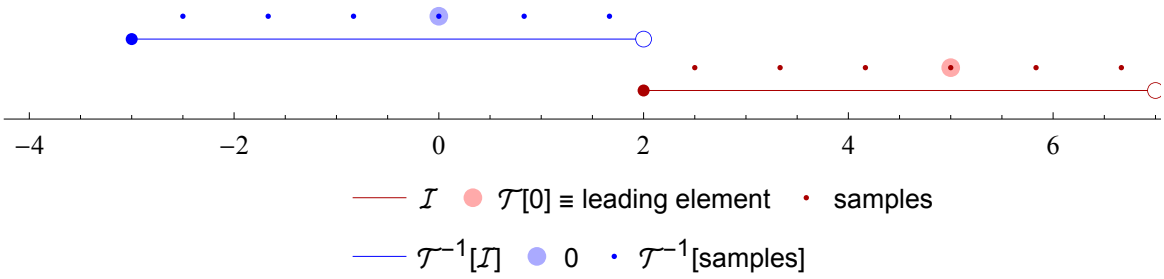
If $0 \notin \overline{\mathcal{I}} = \langle a, b \rangle$, the interval is evenly sampled starting from the unique translation of 0 with sampling interval $\Delta = (b-a)/|n|$. The elements of the sample vector $\mathcal{S}$ do not form a growing sequence; instead, the elements of $\mathcal{S}$ proceed rightward receding translated zero and approaching the right endpoint b; after reaching a maximum they are wrapped back to the minimum close the left endpoint a, eventually approaching trenslated zero from below. Consequently, the equality

$$(b - \text{Max}[\mathcal{S}]) + (\text{Min}[\mathcal{S}] - a) = \Delta$$

applies, and is tested at the end of each subsequent example (see the number next to the resulting vector of samples). Effectively the interval is translated by its length using translation $\mathcal{T}^{-1}$ (see case 0 ∈ (a, b)), sampled, and then the samples are translated back using translation $\mathcal{T}$.

In[479]:=
```
Module[{a = 2, b = 7, n = 6, r}, r = sCircularlySample[{a, b}, n, "Debug" → True];
 Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

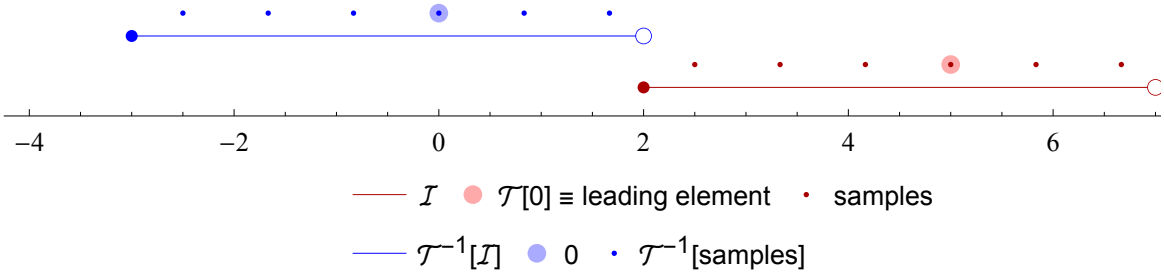
# of samples: {n, p ≡ |n|}	{6, 6}
interval length: b − a	5
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b−a)⌉	1
samp. interval: Δ ≡ (b−a)/p	$\frac{5}{6}$
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{−3, 2}
$\mathcal{I} \equiv \langle a, b)$	{2, 7}
samples	$\left\{5, \frac{35}{6}, \frac{20}{3}, \frac{5}{2}, \frac{10}{3}, \frac{25}{6}\right\}$
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	{−3, 2}
$\mathcal{T}^{-1}$[samples]	$\left\{0, \frac{5}{6}, \frac{5}{3}, -\frac{5}{2}, -\frac{5}{3}, -\frac{5}{6}\right\}$



Out[479]= $\left\{5, \frac{35}{6}, \frac{20}{3}, \frac{5}{2}, \frac{10}{3}, \frac{25}{6}\right\}$ $\frac{5}{6}$

In[480]:=
```
Module[{a = 2, b = 7., n = 6, r}, r = sCircularlySample[{a, b}, n, "Debug" → True];
 Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

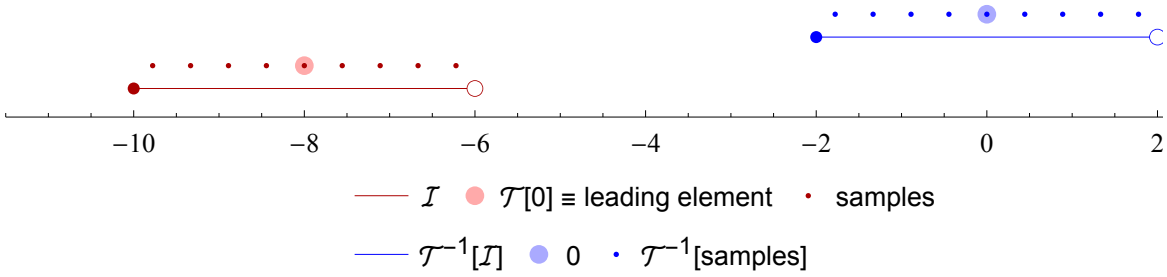
# of samples: {n, p ≡ |n|}	{6, 6}
interval length: b − a	5.
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b−a)⌉	1
samp. interval: Δ ≡ (b−a)/p	0.833333
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{−3, 2}
$\mathcal{I} \equiv \langle a, b)$	{2, 7.}
samples	{5., 5.83333, 6.66667, 2.5, 3.33333, 4.16667}
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	{−3., 2.}
$\mathcal{T}^{-1}$[samples]	{0., 0.833333, 1.66667, −2.5, −1.66667, −0.83



Out[480]= {5., 5.83333, 6.66667, 2.5, 3.33333, 4.16667} 0.833333

In[481]:=
```
Module[{a = -10, b = -6, n = 9, r},
 r = sCircularlySample[{a, b}, n, "Debug" → True];
 Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

# of samples: {n, p ≡ |n|}	{9, 9}
interval length: b − a	4
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b−a)⌉	−2
samp. interval: Δ ≡ (b−a)/p	$\frac{4}{9}$
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{−4, 4}
$\mathcal{I} \equiv \langle a, b)$	{−10, −6}
samples	$\left\{-8, -\frac{68}{9}, -\frac{64}{9}, -\frac{20}{3}, -\frac{56}{9}, -\frac{88}{9}, -\frac{28}{3}, -\frac{80}{9}, -\frac{7}{9}\right.$
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	{−2, 2}
$\mathcal{T}^{-1}$[samples]	$\left\{0, \frac{4}{9}, \frac{8}{9}, \frac{4}{3}, \frac{16}{9}, -\frac{16}{9}, -\frac{4}{3}, -\frac{8}{9}, -\frac{4}{9}\right\}$



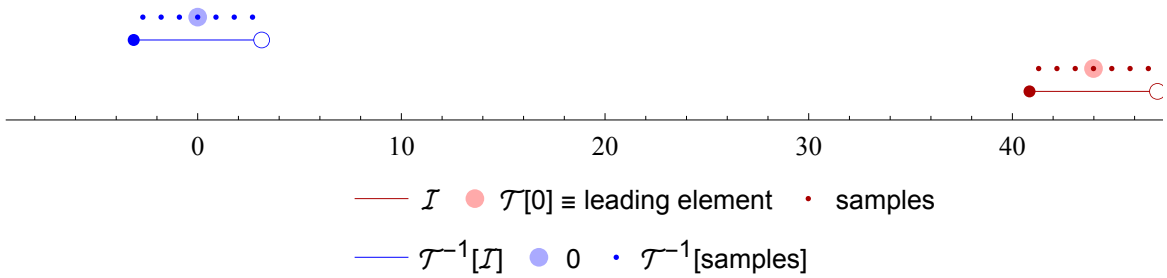
$\mathcal{I}$ ● $\mathcal{T}$[0] ≡ leading element • samples

$\mathcal{T}^{-1}[\mathcal{I}]$ ● 0 • $\mathcal{T}^{-1}$[samples]

Out[481]= $\left\{-8, -\frac{68}{9}, -\frac{64}{9}, -\frac{20}{3}, -\frac{56}{9}, -\frac{88}{9}, -\frac{28}{3}, -\frac{80}{9}, -\frac{76}{9}\right\}$ $\frac{4}{9}$

In[482]:=
```
Module[{a = 13 π, b = 15 π, n = 7, r},
 r = sCircularlySample[{a, b}, n, "Debug" → True];
 Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

# of samples: {n, p ≡ |n|}	{7, 7}
interval length: b − a	2 π
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b−a)⌉	7
samp. interval: Δ ≡ (b−a)/p	$\frac{2\pi}{7}$
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{−3, 3}
$\mathcal{I} \equiv \langle a, b)$	{13 π, 15 π}
samples	$\left\{14\pi, \frac{100\pi}{7}, \frac{102\pi}{7}, \frac{104\pi}{7}, \frac{92\pi}{7}, \frac{94\pi}{7}, \frac{96\pi}{7}\right\}$
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0)$	{−π, π}
$\mathcal{T}^{-1}$[samples]	$\left\{0, \frac{2\pi}{7}, \frac{4\pi}{7}, \frac{6\pi}{7}, -\frac{6\pi}{7}, -\frac{4\pi}{7}, -\frac{2\pi}{7}\right\}$



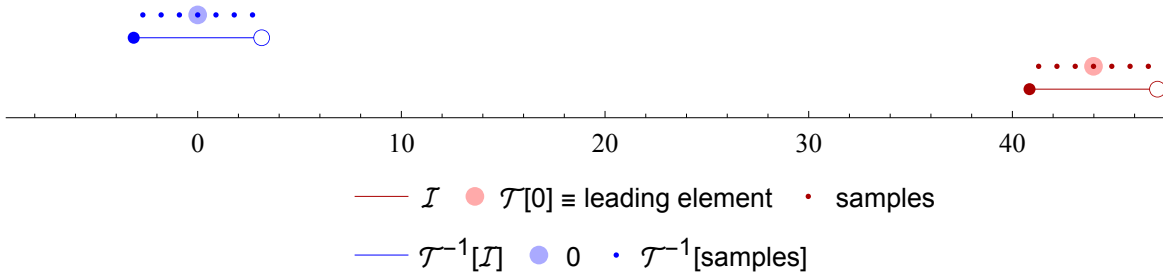
$\mathcal{I}$ ● $\mathcal{T}$[0] ≡ leading element • samples

$\mathcal{T}^{-1}[\mathcal{I}]$ ● 0 • $\mathcal{T}^{-1}$[samples]

Out[482]= $\left\{14\pi, \frac{100\pi}{7}, \frac{102\pi}{7}, \frac{104\pi}{7}, \frac{92\pi}{7}, \frac{94\pi}{7}, \frac{96\pi}{7}\right\}$ $\frac{2\pi}{7}$

In[483]:=
```
Module[{a = 13 π, b = 15. π, n = 7, r},
 r = sCircularlySample[{a, b}, n, "Debug" → True];
 Row[{r, (b - Max[r]) + (Min[r] - a)}, Spacer[8]]]
```

# of samples: {n, p ≡ |n|}	{7, 7}
interval length: b - a	6.28319
transl. $\mathcal{T}$ factor: k ≡ ⌈a/(b-a)⌉	7
samp. interval: Δ ≡ (b-a)/p	0.897598
$\{j_{min}, j_{max}\} \equiv \{\lceil a_0/\Delta \rceil, \lceil b_0/\Delta \rceil - 1\}$	{-3, 3}
$\mathcal{I} \equiv \langle a, b \rangle$	{13 π, 47.1239}
samples	{43.9823, 44.8799, 45.7775, 46.6751, 41.2895,
$\mathcal{T}^{-1}[\mathcal{I}] \equiv \langle a_0, b_0 \rangle$	{-3.14159, 3.14159}
$\mathcal{T}^{-1}$[samples]	{0., 0.897598, 1.7952, 2.69279, -2.69279, -1.7



Out[483]= {43.9823, 44.8799, 45.7775, 46.6751, 41.2895, 42.1871, 43.0847} 0.897598

Applications

Correct sampling the cosine function in a wrapped around order on the interval $\langle -\pi, \pi)$ is easy:

In[484]:=
```
(cosT = Cos[sCircularlySample[{-π, π}, 24]]) // Pane[Style[#, 8], 540] &
cosT // Length
```

Out[484]=
$$\left\{1, \frac{1+\sqrt{3}}{2\sqrt{2}}, \frac{\sqrt{3}}{2}, \frac{1}{\sqrt{2}}, \frac{1}{2}, \frac{-1+\sqrt{3}}{2\sqrt{2}}, 0, -\frac{-1+\sqrt{3}}{2\sqrt{2}}, -\frac{1}{2}, -\frac{1}{\sqrt{2}}, -\frac{\sqrt{3}}{2}, -\frac{1+\sqrt{3}}{2\sqrt{2}}, -1, -\frac{1+\sqrt{3}}{2\sqrt{2}}, -\frac{\sqrt{3}}{2}, -\frac{1}{\sqrt{2}}, -\frac{1}{2}, -\frac{-1+\sqrt{3}}{2\sqrt{2}}, 0, \frac{-1+\sqrt{3}}{2\sqrt{2}}, \frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{\sqrt{3}}{2}, \frac{1+\sqrt{3}}{2\sqrt{2}}\right\}$$

Out[485]= 24

When passing this result into the DFT, we correctly obtain for real even input an real even output:

In[486]:=
```
cosF = Chop@Fourier[cosT, FourierParameters → {1, -1}]
sRotateHalfLength[cosF, 1]
```

Out[486]= {0, 12., 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12.}

Out[487]= {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12., 0, 12., 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

Had we sampled the cosine without taking the wrapped around order as in the following code,

In[488]:=
```
(cosT2 = Cos[Most[Range[-π, π, π / 12]]]) // Pane[Style[#, 8], 540] &
cosT2 // Length
```

Out[488]=
$$\left\{-1, -\frac{1+\sqrt{3}}{2\sqrt{2}}, -\frac{\sqrt{3}}{2}, -\frac{1}{\sqrt{2}}, -\frac{1}{2}, -\frac{-1+\sqrt{3}}{2\sqrt{2}}, 0, \frac{-1+\sqrt{3}}{2\sqrt{2}}, \frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{\sqrt{3}}{2}, \frac{1+\sqrt{3}}{2\sqrt{2}}, 1, \frac{1+\sqrt{3}}{2\sqrt{2}}, \frac{\sqrt{3}}{2}, \frac{1}{\sqrt{2}}, \frac{1}{2}, \frac{-1+\sqrt{3}}{2\sqrt{2}}, 0, -\frac{-1+\sqrt{3}}{2\sqrt{2}}, -\frac{1}{2}, -\frac{1}{\sqrt{2}}, -\frac{\sqrt{3}}{2}, -\frac{1+\sqrt{3}}{2\sqrt{2}}\right\}$$

Out[489]= 24

we would obtain an incorrect Fourier image:

In[490]:=
```
cosF2 = Chop@Fourier[cosT2, FourierParameters → {1, -1}]
sRotateHalfLength[cosF2, 1]
```

Out[490]= {0, -12., 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -12.}

Out[491]= {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -12., 0, -12., 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

The two Fourier images differ in sign, i.e., they will result in identical amplitude spectra, but different phase spectra, since the latter sampling corresponds to minus cosine rather than cosine.

2.3 sCVectorDFTUnpack

In[492]:=
```
infoAbout@sCVectorDFTUnpack
```

sCVectorDFTUnpack[*h*] untangles *h*, the DFT image of a sRVectorCPack[u] complex recast.
sCVectorDFTUnpack[*h*, 2] untangles *h*, the DFT image of a sRVectorCPack[{u, v}] complex recast.

```
Attributes[sCVectorDFTUnpack] = {Protected, ReadProtected}

SyntaxInformation[sCVectorDFTUnpack] = {ArgumentsPattern → {_, _., _., _.}}
```

Details

sCVectorDFTUnpack[*h*, *mod*, *b*, *full*] generalizes untanglement as defined by “twofft”/“realft,” Sec. 12.3 in [Press et al., 1992].

- *mod* == 1 (default) untangles the *n*-element DFT image *h* (DFT represented by the Fourier built-in symbol) of a sRVectorCPack[*u*] complex recast.
- *mod* == 2 untangles the *n*-element DFT image *h* a sRVectorCPack[{*u*, *v*}] complex recast.
- The argument *b* is the second of the FourierParameters. The only permitted values are –1 (signal processing, default) or 1 (as in [Press et al., 1992]). Requires corresponding settings FourierParameters → {1, *b*} for any use of the built-in symbol Fourier outside the sCVectorDFTUnpack (e.g., for comparison purposes in the examples that follow).

- Only the nonnegative frequency half ($n+1$ frequencies $0, \ldots, n$) is returned unless the argument *full* is se to True (default False). The remaining elements can be restored using the DFT's symmetry.
- Typical usage (keep in mind correct setting FourierParameters → {1, b}):

f = sCVectorDFTUnpack@Fourier@sRVectorCPack@u

g = sCVectorDFTUnpack[Fourier@sRVectorCPack[{u, v}], 2]

- Note: The first example above makes no sense if fed with a vector of odd length since this causes irrecoverable information loss by truncating (addition by padding), thus returning incorrect answer.

Examples

Clearing global symbol:

In[493]:=
```
ClearAll[untangleEx];
```

First we define the following helper function to save space:

In[494]:=
```
untangleEx[u_List, b_Integer: -1, full_: False] :=
 Module[{h, fu, fh, ans}, h = sRVectorCPack[u];
  fh = Chop@Fourier[h, FourierParameters → {1, b}];
  ans = Chop@sCVectorDFTUnpack[fh, 1, b, full];
  fu = Chop@Fourier[u, FourierParameters → {1, b}];
  TableForm[{Pane /@ {u, h, fh, ans, fu}, Length /@ {u, h, fh, ans, fu}},
   TableDirections → Row,
   TableHeadings →
    {None, {"u", "h = sRVectorCPack[u]", "g = Fourier[h]",
      "sCVectorDFTUnpack[g]", "Fourier[u]"}}, TableSpacing → {2, 1}]]
untangleEx[{u_List, v_List}, b_Integer: -1, full_: False] :=
 Module[{h, fuu, fuv, fh, ans}, h = sRVectorCPack[{u, v}];
  fh = Chop@Fourier[h, FourierParameters → {1, b}];
  ans = Chop@sCVectorDFTUnpack[fh, 2, b, full];
  {fuu, fuv} = Chop@Fourier[#, FourierParameters → {1, b}] & /@ {u, v};
  TableForm[{Pane /@ {u, v, h, fh, Column@ans, fuu, fuv},
    Length /@ {u, v, h, fh, ans, fuu, fuv}}, TableDirections → Row,
   TableHeadings →
    {None, {"u", "v", "h = sRVectorCPack[{u, v}]", "g = Fourier[h]",
      "sCVectorDFTUnpack[g]", "Fourier[u]", "Fourier[v]"}},
   TableSpacing → {2, 1}]]
```

DFT of a single real vector

Examples are printed in the following format: the first row contains the real vector u passed to the helper function untangleEx as the first argument, the second row shows its complex repack h using the sRVectorCPack function, h = sRVectorCPack[u]. In the third row, Fourier image

g = Fourier[h] of the complex repack h is displayed, and the following one shows the result after untangling g with sCVectorDFTUnpack. Finally, the direct Fourier[u] is in the last row for comparison. The last column displays lenghts of corresponding lists.

```
In[496]:= untangleEx[Table[N@Cos[π k], {k, 0, 15}]] // Style[#, 6] &
```

Out[496]=

u	{1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1.}	16
h = sRVectorCPack[u]	{1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i}	8
g = Fourier[h]	{8. − 8. i, 0, 0, 0, 0, 0, 0, 0}	8
sCVectorDFTUnpack[g]	{0, 0, 0, 0, 0, 0, 0, 0, 16.}	9
Fourier[u]	{0, 0, 0, 0, 0, 0, 0, 0, 16., 0, 0, 0, 0, 0, 0, 0}	16

Giving the "FullSpectrum" → True option changes the result as follows:

```
In[497]:= untangleEx[Table[N@Cos[π k], {k, 0, 15}], -1, True] // Style[#, 6] &
```

Out[497]=

u	{1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1.}	16
h = sRVectorCPack[u]	{1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i}	8
g = Fourier[h]	{8. − 8. i, 0, 0, 0, 0, 0, 0, 0}	8
sCVectorDFTUnpack[g]	{0, 0, 0, 0, 0, 0, 0, 0, 16., 0, 0, 0, 0, 0, 0, 0}	16
Fourier[u]	{0, 0, 0, 0, 0, 0, 0, 0, 16., 0, 0, 0, 0, 0, 0, 0}	16

Here we used Fourier parameters {1, 1} instead of the default {1, −1}:

```
In[498]:= untangleEx[Table[N@Cos[π k], {k, 0, 15}], 1] // Style[#, 6] &
```

Out[498]=

u	{1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1.}	16
h = sRVectorCPack[u]	{1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i}	8
g = Fourier[h]	{8. − 8. i, 0, 0, 0, 0, 0, 0, 0}	8
sCVectorDFTUnpack[g]	{0, 0, 0, 0, 0, 0, 0, 0, 16.}	9
Fourier[u]	{0, 0, 0, 0, 0, 0, 0, 0, 16., 0, 0, 0, 0, 0, 0, 0}	16

```
In[499]:= untangleEx[Table[N@Cos[π k], {k, 0, 15}], 1, True] // Style[#, 6] &
```

Out[499]=

u	{1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1., 1., −1.}	16
h = sRVectorCPack[u]	{1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i, 1. − 1. i}	8
g = Fourier[h]	{8. − 8. i, 0, 0, 0, 0, 0, 0, 0}	8
sCVectorDFTUnpack[g]	{0, 0, 0, 0, 0, 0, 0, 0, 16., 0, 0, 0, 0, 0, 0, 0}	16
Fourier[u]	{0, 0, 0, 0, 0, 0, 0, 0, 16., 0, 0, 0, 0, 0, 0, 0}	16

Other easy-to-understand examples:

```
In[500]:= untangleEx[{1, 1}]
```

Out[500]//TableForm=

u	{1, 1}	2
h = sRVectorCPack[u]	{1 + i}	1
g = Fourier[h]	{1. + 1. i}	1
sCVectorDFTUnpack[g]	{2., 0}	2
Fourier[u]	{2., 0}	2

In[501]:=
```
untangleEx[{1, 1, 0, 0, 0, 0, 0, 1}] // Style[#, 8] &
```

Out[501]=

u	{1, 1, 0, 0, 0, 0, 0, 1}	8
h = sRVectorCPack[u]	{1 + i, 0, 0, i}	4
g = Fourier[h]	{1. + 2. i, 0. + 1. i, 1., 2. + 1. i}	4
sCVectorDFTUnpack[g]	{3., 2.41421, 1., −0.414214, −1.}	5
Fourier[u]	{3., 2.41421, 1., −0.414214, −1., −0.414214, 1., 2.41421}	8

In[502]:=
```
untangleEx[Table[N@Cos[π k / 2], {k, 0, 15}], -1, True] // Style[#, 6] &
```

Out[502]=

u	{1., 0., −1., 0., 1., 0., −1., 0., 1., 0., −1., 0., 1., 0., −1., 0.}	16
h = sRVectorCPack[u]	{1. + 0. i, −1. + 0. i, 1. + 0. i, −1. + 0. i, 1. + 0. i, −1. + 0. i, 1. + 0. i, −1. + 0. i}	8
g = Fourier[h]	{0, 0, 0, 0, 8., 0, 0, 0}	8
sCVectorDFTUnpack[g]	{0, 0, 0, 0, 8., 0, 0, 0, 0, 0, 0, 0, 8., 0, 0, 0}	16
Fourier[u]	{0, 0, 0, 0, 8., 0, 0, 0, 0, 0, 0, 0, 8., 0, 0, 0}	16

DFT of two real vectors

Examples are printed in the following format: the first two rows contain the real vectors $\{u, v\}$ passed to the helper function untangleEx as the first argument, the third row shows its complex repack h using the sRVectorCPack function, $h = \text{sRVectorCPack}[\{u, v\}]$. In the fourth row, Fourier image $g = \text{Fourier}[h]$ of the complex repack h is displayed, and the following one shows the result after untangling g with sCVectorDFTUnpack. Finally, the direct Fourier$[u]$ and Fourier$[v]$ are in the last two rows for comparison. The last column displays lenghts of corresponding lists.

In[503]:=
```
untangleEx[{{1, 1}, {0, 1}}]
```

Out[503]//TableForm=

u	{1, 1}	2
v	{0, 1}	2
h = sRVectorCPack[{u, v}]	{1, 1 + i}	2
g = Fourier[h]	{2. + 1. i, 0. − 1. i}	2
sCVectorDFTUnpack[g]	{2., 0} {1., −1.}	2
Fourier[u]	{2., 0}	2
Fourier[v]	{1., −1.}	2

In[504]:=
```
untangleEx[{N@Cos[sCircularlySample[{-π, π}, 6]], {1, -1, 1, -1, 1, -1}},
  -1, True] // Style[#, 8] &
```

Out[504]=

u	{1., 0.5, −0.5, −1., −0.5, 0.5}	6
v	{1, −1, 1, −1, 1, −1}	6
h = sRVectorCPack[{u, v}]	{1. + 1. i, 0.5 − 1. i, −0.5 + 1. i, −1. − 1. i, −0.5 + 1. i, 0.5 − 1. i}	6
g = Fourier[h]	{0, 3., 0, 0. + 6. i, 0, 3.}	6
sCVectorDFTUnpack[g]	{0, 3., 0, 0, 0, 3.} {0, 0, 0, 6., 0, 0}	2
Fourier[u]	{0, 3., 0, 0, 0, 3.}	6
Fourier[v]	{0, 0, 0, 6., 0, 0}	6

Applications

This function can probably be applied, together with sRVectorCPack, in speeding up the DFT computation of a very long 1D real data using the "twofft"/"realft" untanglement described in details in Sec. 12.3 in [Press et al., 1992].

2.4 sRVectorCPack

In[505]:=
```
infoAbout@sRVectorCPack
```

sRVectorCPack[*u*] recasts the real vector *u* into a half-length complex vector.
sRVectorCPack[{*u*, *v*}] recasts two real vectors *u*, *v* into a single complex vector.

```
Attributes[sRVectorCPack] = {Protected, ReadProtected}

Options[sRVectorCPack] = {Padding → None}

SyntaxInformation[sRVectorCPack] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

Details

sRVectorCPack[*u*, *opts*] recasts the real vector

$$u = \{u_1,\ u_2,\ \ldots,\ u_n\}$$

of an arbitrary positive length *n* into a half-length complex vector

$$\{u_{2\,k-1} + i\,u_{2\,k}\},\ \ k = 1,\ \ldots,\ \lfloor n/2 \rfloor$$

- u_n is discarded if *n* is odd and the option Padding → None (default).
- The option setting Padding → *p* value (*p* must pass NumberQ) is used as a substitute for a missing paired element for an odd *n*, returning last element $u_n + i\,p$. This option has no effect for an even *n*.
- The option setting Padding → "Periodic" substitutes u_1 for a missing paired element (with an odd *n*) instead, returning last element $u_n + i\,u_1$. This option has no effect for an even *n*.

sRVectorCPack[{*u*, *v*}, *opts*] recasts two real vectors

$$u = \{u_1,\ u_2,\ \ldots,\ u_n\},\ \ v = \{v_1,\ v_2,\ \ldots,\ v_m\}$$

of respective lengths *n*, *m* into a single complex vector

$$\{u_k + i\,v_k\}$$

- If *n* == *m* (Padding is ignored is such case), we get a complex vector of length *n*.
- If $n \neq m$,
  - the option setting Padding → None (default) causes truncating the longer vector,
  - the option setting Padding → {p_1, …} causes the sequence p_1, … to be cyclically appended to the shorter vector until both lengths equal,

- the option setting Padding → "Periodic" cyclically repeats the shorter vector until both lengths equal.

Remarks valid for both syntax cases:

- All vector elements must pass NumericQ and all padding elements must pass NumberQ, however, only makes sense for use with integer, rational, and real values.
- Complex-valued input vectors are possible but inept (not checked for).
- Actually implements the "input entaglement part" of "twofft/realft," Sec. 12.3 in [Press et al., 1992].
- Refer to sCVectorDFTUnpack for typical usage.

Examples

Clearing global symbols:

```
In[506]:= Clear[u, v];
```

Repackig a single real vector

This will be our real test vector:

```
In[507]:= u = sGrandiSequence[8] Range[8]
Out[507]= {1, -2, 3, -4, 5, -6, 7, -8}
```

No truncation occurs unless the length of the vector is odd (the same result will be returned with the Padding→None option):

```
In[508]:= Column@{u, sRVectorCPack[u]}
Out[508]= {1, -2, 3, -4, 5, -6, 7, -8}
          {1 - 2 i, 3 - 4 i, 5 - 6 i, 7 - 8 i}
```

The unpaired end element will be discarded, however, and a note about truncation will be issued:

```
In[509]:= With[{u = Join[u, {9}]}, Column@{u, sRVectorCPack[u]}]
          sRVectorCPack: NOTE: odd-length vector, unpaired end element discarded.
Out[509]= {1, -2, 3, -4, 5, -6, 7, -8, 9}
          {1 - 2 i, 3 - 4 i, 5 - 6 i, 7 - 8 i}
```

Any specified padding will be ignored with an even number of elements:

```
In[510]:= Column@{u, sRVectorCPack[u, Padding → 10]}
Out[510]= {1, -2, 3, -4, 5, -6, 7, -8}
          {1 - 2 i, 3 - 4 i, 5 - 6 i, 7 - 8 i}
```

An explicitly specified pad value only has effect with an odd number of elements:

```
In[511]:= With[{u = Join[u, {9}]}, Column@{u, sRVectorCPack[u, Padding → 10]}]
```

sRVectorCPack: NOTE: odd-length vector, padding applied.

```
Out[511]= {1, -2, 3, -4, 5, -6, 7, -8, 9}
          {1 - 2 i, 3 - 4 i, 5 - 6 i, 7 - 8 i, 9 + 10 i}
```

With periodic padding, the 1st element will be substituted for the missing paired element:

```
In[512]:= With[{u = Join[u, {9}]}, Column@{u, sRVectorCPack[u, Padding → "Periodic"]}]
```

sRVectorCPack: NOTE: odd-length vector, padding applied.

```
Out[512]= {1, -2, 3, -4, 5, -6, 7, -8, 9}
          {1 - 2 i, 3 - 4 i, 5 - 6 i, 7 - 8 i, 9 + i}
```

The behavior for a trivial, one-element vector:

```
In[513]:= Column@{sRVectorCPack[{1}], sRVectorCPack[{1}, Padding → 10]}
```

sRVectorCPack: NOTE: odd-length vector, unpaired end element discarded.

sRVectorCPack: NOTE: odd-length vector, padding applied.

```
Out[513]= {}
          {1 + 10 i}
```

Repackig two real vectors

This will be our real test vectors:

```
In[514]:= u = sGrandiSequence[6] Range[6];
          v = sGrandiSequence[-6] Range[66, 11, -11];
          Column@{u, v}
```

```
Out[514]= {1, -2, 3, -4, 5, -6}
          {-66, 55, -44, 33, -22, 11}
```

No truncation occurs unless the lengths are equal (the same result will be returned with the Padding→None option):

```
In[515]:= Column@{u, v, sRVectorCPack[{u, v}]}
```

```
Out[515]= {1, -2, 3, -4, 5, -6}
          {-66, 55, -44, 33, -22, 11}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i}
```

End elements of any vector that exceed the length of the other vector will be discarded, however, and a note about truncation will be issued:

```
In[516]:= With[{u = Join[u, {7, -8, 9}]}, Column@{u, v, sRVectorCPack[{u, v}]}]
```

sRVectorCPack: NOTE: vector lengths differ, truncation applied.

```
Out[516]= {1, -2, 3, -4, 5, -6, 7, -8, 9}
          {-66, 55, -44, 33, -22, 11}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i}
```

```
In[517]:= With[{v = Join[v, {0, -11, 22}]}, Column@{u, v, sRVectorCPack[{u, v}]}]
```

sRVectorCPack: NOTE: vector lengths differ, truncation applied.

```
Out[517]= {1, -2, 3, -4, 5, -6}
          {-66, 55, -44, 33, -22, 11, 0, -11, 22}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i}
```

Padding one vector exactly up to the length of the other vector:

```
In[518]:= With[{u = Join[u, {7}]}, Column@{u, v, sRVectorCPack[{u, v}, Padding → {100}]}]
```

sRVectorCPack: NOTE: vector lengths differ, padding applied.

```
Out[518]= {1, -2, 3, -4, 5, -6, 7}
          {-66, 55, -44, 33, -22, 11}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i, 7 + 100 i}
```

```
In[519]:= With[{u = Join[u, {7, -8, 9}]},
           Column@{u, v, sRVectorCPack[{u, v}, Padding → {100, 200, 300}]}]
```

sRVectorCPack: NOTE: vector lengths differ, padding applied.

```
Out[519]= {1, -2, 3, -4, 5, -6, 7, -8, 9}
          {-66, 55, -44, 33, -22, 11}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i,
           5 - 22 i, -6 + 11 i, 7 + 100 i, -8 + 200 i, 9 + 300 i}
```

Adding more pads than necessary results in not using all pad elements:

```
In[520]:= With[{u = Join[u, {7, -8}]},
           Column@{u, v, sRVectorCPack[{u, v}, Padding → {100, 200, 300}]}]
```

sRVectorCPack: NOTE: vector lengths differ, padding applied.

```
Out[520]= {1, -2, 3, -4, 5, -6, 7, -8}
          {-66, 55, -44, 33, -22, 11}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i, 7 + 100 i, -8 + 200 i}
```

Adding less pads than necessary results in using the pads cyclically:

```
In[521]:= With[{v = Join[v, {0, -11, 22}]},
           Column@{u, v, sRVectorCPack[{u, v}, Padding → {100}]}]
```

sRVectorCPack: NOTE: vector lengths differ, padding applied.

```
Out[521]= {1, -2, 3, -4, 5, -6}
          {-66, 55, -44, 33, -22, 11, 0, -11, 22}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i, 100, 100 - 11 i, 100 + 22 i}
```

```
In[522]:= With[{v = Join[v, {0, -11, 22, -33, 44, -55, 66}]},
           Column@{u, v, sRVectorCPack[{u, v}, Padding → {100, 200, 300}]}]

          sRVectorCPack: NOTE: vector lengths differ, padding applied.

Out[522]= {1, -2, 3, -4, 5, -6}
          {-66, 55, -44, 33, -22, 11, 0, -11, 22, -33, 44, -55, 66}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i, 100,
           200 - 11 i, 300 + 22 i, 100 - 33 i, 200 + 44 i, 300 - 55 i, 100 + 66 i}
```

With periodic padding, the shorter vector will be cyclically repeated until the lengths equal:

```
In[523]:= With[{u = Join[u, {7}]},
           Column@{u, v, sRVectorCPack[{u, v}, Padding → "Periodic"]}]

          sRVectorCPack: NOTE: vector lengths differ, padding applied.

Out[523]= {1, -2, 3, -4, 5, -6, 7}
          {-66, 55, -44, 33, -22, 11}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i, 7 - 66 i}
```

```
In[524]:= With[{v = Join[v, {0, -11, 22}]},
           Column@{u, v, sRVectorCPack[{u, v}, Padding → "Periodic"]}]

          sRVectorCPack: NOTE: vector lengths differ, padding applied.

Out[524]= {1, -2, 3, -4, 5, -6}
          {-66, 55, -44, 33, -22, 11, 0, -11, 22}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i, 1, -2 - 11 i, 3 + 22 i}
```

Difference length equal to the shorter vector length:

```
In[525]:= With[{u = Join[u, {7, -8, 9, -10, 11, -12}]},
           Column@{u, v, sRVectorCPack[{u, v}, Padding → "Periodic"]}]

          sRVectorCPack: NOTE: vector lengths differ, padding applied.

Out[525]= {1, -2, 3, -4, 5, -6, 7, -8, 9, -10, 11, -12}
          {-66, 55, -44, 33, -22, 11}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i,
           7 - 66 i, -8 + 55 i, 9 - 44 i, -10 + 33 i, 11 - 22 i, -12 + 11 i}
```

Difference length exceeds the shorter vector length:

```
In[526]:= With[{u = Join[u, {7, -8, 9, -10, 11, -12, 13}]},
           Column@{u, v, sRVectorCPack[{u, v}, Padding → "Periodic"]}]

          sRVectorCPack: NOTE: vector lengths differ, padding applied.

Out[526]= {1, -2, 3, -4, 5, -6, 7, -8, 9, -10, 11, -12, 13}
          {-66, 55, -44, 33, -22, 11}
          {1 - 66 i, -2 + 55 i, 3 - 44 i, -4 + 33 i, 5 - 22 i, -6 + 11 i,
           7 - 66 i, -8 + 55 i, 9 - 44 i, -10 + 33 i, 11 - 22 i, -12 + 11 i, 13 - 66 i}
```

Applications

This function can probably be applied, together with sCVectorDFTUnpack, in speeding up the DFT computation of a very long 1D real data using the "twofft"/"realft" untanglement described in details in Sec. 12.3 in [Press et al., 1992].

2.5 sRealFourier

In[527]:=
```
infoAbout@sRealFourier
```

sRealFourier[*u*] gives one-sided DFT for an
even length real vector *u* using FourierParameters → {1, −1}.
sRealFourier[{*u*, *v*}] gives one-sided DFT for two real vectors *u*, *v* using FourierParameters → {1, −1}.

```
Attributes[sRealFourier] = {Protected, ReadProtected}
```

```
SyntaxInformation[sRealFourier] = {ArgumentsPattern → {_, _., _.}}
```

Details

sRealFourier[*u*, *b*, *full*] returns the DFT for an even-length real vector *u* using the Fourier built-in symbol with FourierParameters → {1, *b*}.

- Just a wrapper for the Fourier built-in symbol that implements the ideas of Sec. 12.3 in [Press et al., 1992].
- Only the nonnegative frequency half (one-sided spectrum) is returned unless *full* == True. First the one-sided spectrum is computed, only then the two-sided spectrum is additionally derived using symmetry, if required.
- Only makes sense for use with integer, rational, and real vectors; complex-valued vectors are possible but inept.
- Allowed values for the Fourier parameter *b* are −1 (default) and 1.
- Due to speed, just a minimal argument check.
- A close relative of sRVectorCPack, sCVectorDFTUnpack.
- Not so beneficial in terms of computing time compared to the usual Fourier built-in symbol, but can serve as an illustration for Sec. 12.3 in [Press et al., 1992].

sRealFourier[{*u*, *v*}, *b*, *full*] returns pair of the DFT's for two real vectors *u*, *v* (of arbitrary nonzero but equal lengths) using the Fourier built-in symbol with FourierParameters → {1, *b*}.

- Just a wrapper for the Fourier built-in symbol that implements the ideas of Sec. 12.3 in [Press et al., 1992].
- Only the nonnegative frequency halves (one-sided spectra) are returned unless *full* == True. First the two-sided spectra are computed, only then the one-sided spectra are additionally derived by cutting off, if required.
- Only makes sense for use with integer, rational, and real vectors; complex-valued vectors are possible but inept.

- Allowed values for the Fourier parameter *b* are −1 (default) and 1.
- Due to speed, just a minimal argument check.
- A close relative of sRVectorCPack, sCVectorDFTUnpack.
- Contrary to sRealFourier[*u*, *b*, *full*] , this case is beneficial in terms of computing time compared to the Fourier built-in symbol applied twice.

Examples

Clearing global symbols:

```
In[528]:= Clear[optsFourier, b, u, v, f, g, h, rf, num, tf, tim, num, tg, stats, ℋ, uvlen];
```

This will be our setup for both the Fourier built-in symbol and for sRealFourier symbol (allowing restoration of the options of the built-in symbol Fourier to the original state):

```
In[529]:= optsFourier = Options[Fourier];
          b = -1;
          SetOptions[Fourier, FourierParameters → {1, b}];
```

sRealFourier of a single real vector

This will be our real test vector:

```
In[531]:= u = UnitVector[8, 3]
Out[531]= {0, 0, 1, 0, 0, 0, 0, 0}
```

First we demonstrate behavior with the parameter *full* set to False (default, only the nonnegative frequency half is returned), then with *full* set to True (a complete Fourier image is returned). Finally, the norm of the difference of the built-in Fourier and sRealFourier is computed:

```
In[532]:= Column@{u, (f = Fourier[u]) // Chop, (rf = sRealFourier[u, b]) // Chop}
          If[rf =!= $Failed,
           Norm[rf - If[Length@rf < Length@f, Take[f, Length@f / 2 + 1], f]]]
          Column@{u, (f = Fourier[u]) // Chop, (rf = sRealFourier[u, b, True]) // Chop}
          If[rf =!= $Failed,
           Norm[rf - If[Length@rf < Length@f, Take[f, Length@f / 2 + 1], f]]]

          {0, 0, 1, 0, 0, 0, 0, 0}
Out[532]= {1., 0. - 1. i, -1., 0. + 1. i, 1., 0. - 1. i, -1., 0. + 1. i}
          {1., 0. - 1. i, -1., 0. + 1. i, 1.}

Out[533]= 0.

          {0, 0, 1, 0, 0, 0, 0, 0}
Out[534]= {1., 0. - 1. i, -1., 0. + 1. i, 1., 0. - 1. i, -1., 0. + 1. i}
          {1., 0. - 1. i, -1., 0. + 1. i, 1., 0. - 1. i, -1., 0. + 1. i}

Out[535]= 0.
```

sRealFourier of two real vectors

This will be our real test vectors of the same even lengths:

```
In[536]:= u = UnitVector[8, 1];
          v = N[(1 + Cos[#] + Cos[2 #] / 2 + Cos[3 #] / 4 + Cos[4 #] / 16) &@
              sCircularlySample[{-π, π}, 8]];
```

For both one-sided and two-sided spectra, the test vectors are shown, followed by respective outputs of the built-in Fourier symbol. Finally, in the penultimate two rows, the sRealFourier output is displayed (in columnized style), with the very last row showing the norm of the difference between the built-in Fourier and sRealFourier:

```
In[538]:= Column@{u, v, (f = Fourier[u]) // Chop, (g = Fourier[v]) // Chop,
            Column[(h = sRealFourier[{u, v}, b]) // Chop]}
          If[h =!= $Failed,
           Norm[Join @@ h - If[First[Length /@ h] < Length@f,
              Join[Take[f, Ceiling[(Length@f + 1) / 2]],
               Take[g, Ceiling[(Length@g + 1) / 2]]], Join[f, g]]]]
          Column@{u, v, (f = Fourier[u]) // Chop, (g = Fourier[v]) // Chop,
            Column[(h = sRealFourier[{u, v}, b, True]) // Chop]}
          If[h =!= $Failed,
           Norm[Join @@ h - If[First[Length /@ h] < Length@f,
              Join[Take[f, Ceiling[(Length@f + 1) / 2]],
               Take[g, Ceiling[(Length@g + 1) / 2]]], Join[f, g]]]]
```

```
Out[538]= {1, 0, 0, 0, 0, 0, 0, 0}
          {2.8125, 1.46783, 0.5625, 0.40717, 0.3125, 0.40717, 0.5625, 1.46783}
          {1., 1., 1., 1., 1., 1., 1., 1.}
          {8., 4., 2., 1., 0.5, 1., 2., 4.}
          {1., 1., 1., 1., 1.}
          {8., 4., 2., 1., 0.5}
```

```
Out[539]= 0.
```

```
Out[540]= {1, 0, 0, 0, 0, 0, 0, 0}
          {2.8125, 1.46783, 0.5625, 0.40717, 0.3125, 0.40717, 0.5625, 1.46783}
          {1., 1., 1., 1., 1., 1., 1., 1.}
          {8., 4., 2., 1., 0.5, 1., 2., 4.}
          {1., 1., 1., 1., 1., 1., 1., 1.}
          {8., 4., 2., 1., 0.5, 1., 2., 4.}
```

```
Out[541]= 0.
```

And now once again, but this time for test vectors of the same odd lengths:

In[542]:=
```
u = UnitVector[7, 1];
v = N[(1 + Cos[#] + Cos[2 #] / 2 + Cos[3 #] / 4 + Cos[4 #] / 16) &@
    sCircularlySample[{-π, π}, 7]];
Column@{u, v, (f = Fourier[u]) // Chop, (g = Fourier[v]) // Chop,
  Column[(h = sRealFourier[{u, v}, b]) // Chop]}
If[h =!= $Failed,
 Norm[Join @@ h - If[First[Length /@ h] < Length@f,
    Join[Take[f, Ceiling[(Length@f + 1) / 2]],
     Take[g, Ceiling[(Length@g + 1) / 2]]], Join[f, g]]]]
Column@{u, v, (f = Fourier[u]) // Chop, (g = Fourier[v]) // Chop,
  Column[(h = sRealFourier[{u, v}, b, True]) // Chop]}
If[h =!= $Failed,
 Norm[Join @@ h - If[First[Length /@ h] < Length@f,
    Join[Take[f, Ceiling[(Length@f + 1) / 2]],
     Take[g, Ceiling[(Length@g + 1) / 2]]], Join[f, g]]]]
```

Out[544]=
```
{1, 0, 0, 0, 0, 0, 0}
{2.8125, 1.23068, 0.521835, 0.341238, 0.341238, 0.521835, 1.23068}
{1., 1., 1., 1., 1., 1., 1.}
{7., 3.5, 1.75, 1.09375, 1.09375, 1.75, 3.5}
{1., 1., 1., 1.}
{7., 3.5, 1.75, 1.09375}
```

Out[545]=
```
0.
```

Out[546]=
```
{1, 0, 0, 0, 0, 0, 0}
{2.8125, 1.23068, 0.521835, 0.341238, 0.341238, 0.521835, 1.23068}
{1., 1., 1., 1., 1., 1., 1.}
{7., 3.5, 1.75, 1.09375, 1.09375, 1.75, 3.5}
{1., 1., 1., 1., 1., 1., 1.}
{7., 3.5, 1.75, 1.09375, 1.09375, 1.75, 3.5}
```

Out[547]=
```
0.
```

Applications

This function fuses functionality of sRVectorCPack, sCVectorDFTUnpack and the Fourier built-in symbol, covering area of applicability in speeding up the DFT computation of a very long 1D real data using the “twofft”/”realft” untanglement described in details in Sec. 12.3 in [Press et al., 1992].

Discussion

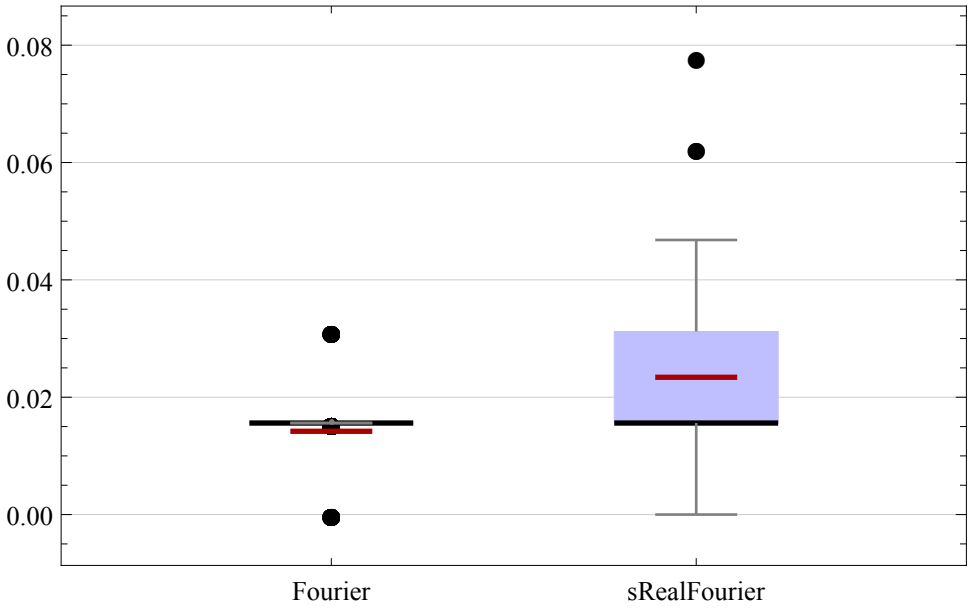
Performance test reveal that using sRealFourier for a single real vector is not beneficial. It was only implemented for completeness and as an illustration for the ideas outlined in Sec. 12.3 in [Press et al., 1992].

In[548]:=
```
num = 100; u = RandomReal[{-1, 1}, 2^20]; tim = Timing;
tf = Table[(Fourier[u]; // tim)[[1]], {num}];
tg = Table[(sRealFourier[u, b, False]; // tim)[[1]], {num}];
stats = {Mean, StandardDeviation, Median, MedianDeviation};
TableForm[Through[stats[#]] & /@ {tf, tg}, TableHeadings → {None, stats}]
BoxWhiskerChart[{tf, tg},
 {{"Outliers"}, {"FarOutliers"}, {"MedianMarker", 1, Directive[Thick, Black]},
  {"MeanMarker", 0.5, Directive[Thick, Darker@Red]}},
 ChartStyle → Lighter[Blue, 0.75], GridLines → {None, Automatic},
 ChartLabels → {"Fourier", "sRealFourier"}]
ℋ = LocationTest[{tf, tg}, 0, "HypothesisTestData",
   VerifyTestAssumptions → All];
ℋ["TestDataTable", All] // Magnify[#, 2 / 3] &
ℋ["TestConclusion", Automatic]
```

Out[552]//TableForm=

Mean	StandardDeviation	Median	MedianDeviation
0.0141961	0.00916768	0.0156001	0.
0.0234001	0.0139355	0.0156001	0.

Out[553]=



Out[555]=

	Statistic	P-Value
Mann-Whitney	3173.	4.13375×10^{-7}
Sign	14	7.06949×10^{-6}

Out[556]=
```
The null hypothesis that the mean difference is 0
 is rejected at the 5 percent level based on the Mann-Whitney test.
```

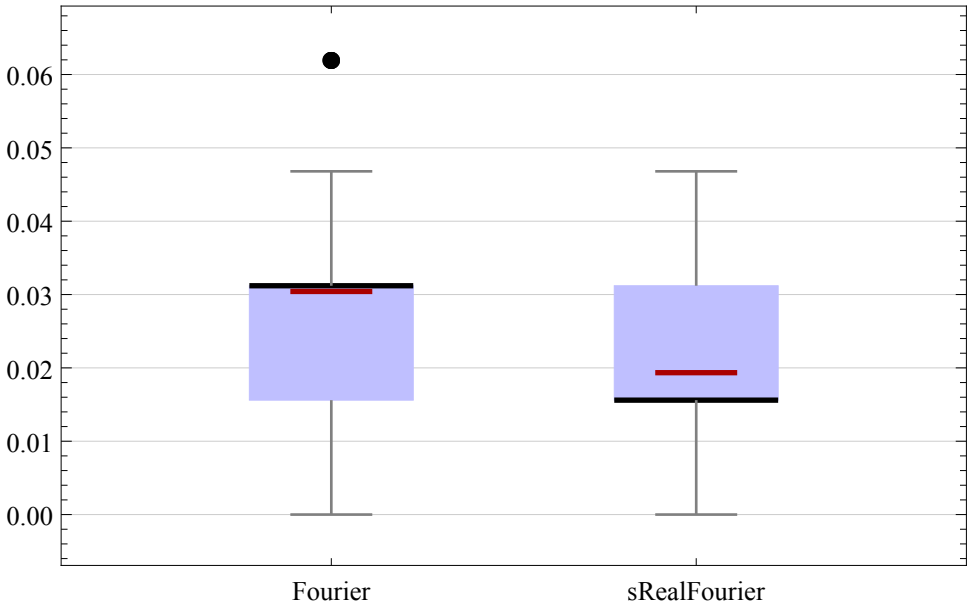
Performance test reveal that using sRealFourier on two real vector is slightly faster, but this improvement may not be statistically significant:

In[557]:=
```
num = 100;
uvlen = 2^20;
u = RandomReal[{-1, 1}, uvlen];
v = RandomReal[{-1, 1}, uvlen];
tim = Timing;
tf = Table[(Fourier[u]; // tim)[[1]] + (Fourier[v]; // tim)[[1]], {num}];
tg = Table[(sRealFourier[{u, v}, b, True]; // tim)[[1]], {num}];
stats = {Mean, StandardDeviation, Median, MedianDeviation};
TableForm[Through[stats[#]] & /@ {tf, tg}, TableHeadings → {None, stats}]
BoxWhiskerChart[{tf, tg},
 {{"Outliers"}, {"FarOutliers"}, {"MedianMarker", 1, Directive[Thick, Black]},
  {"MeanMarker", 0.5, Directive[Thick, Darker@Red]}},
 ChartStyle → Lighter[Blue, 0.75], GridLines → {None, Automatic},
 ChartLabels → {"Fourier", "sRealFourier"}]
ℋ = LocationTest[{tf, tg}, 0, "HypothesisTestData",
   VerifyTestAssumptions → All, AlternativeHypothesis → "Greater"];
ℋ["TestDataTable", All] // Magnify[#, 2 / 3] &
ℋ["TestConclusion", Automatic]
```

Out[562]//TableForm=

Mean	StandardDeviation	Median	MedianDeviation
0.0304202	0.0139135	0.0312002	0.0156001
0.0193441	0.0111131	0.0156001	0.

Out[563]=



Out[565]=

	Statistic	P-Value
Mann-Whitney	7246.	2.53463×10^{-9}
Sign	58	2.0466×10^{-7}

Out[566]=
```
The null hypothesis that the mean difference is less than or equal to 0
 is rejected at the 5 percent level based on the Mann-Whitney test.
```

Now we can restore the Fourier options setting to its original value:

In[567]:=
```
SetOptions[Fourier, Sequence @@ optsFourier];
```

2.6 sSavGolKernel1D

In[568]:=
```
infoAbout@sSavGolKernel1D
```

sSavGolKernel1D[*ord*, {*nl*, *nr*}] gives a Savitzky–Golay
1D kernel of order *ord* and *nl* (*nr*) elements on the left (right).

```
Attributes[sSavGolKernel1D] = {Protected, ReadProtected}

Options[sSavGolKernel1D] = {Derivs → {0}, Indexed → False}

SyntaxInformation[sSavGolKernel1D] =
 {ArgumentsPattern → {_, {_, _}, OptionsPattern[]}}
```

Details

sSavGolKernel1D[*ord*, $\{n_l, n_r\}$, *opts*] returns a Savitzky–Golay 1D rational-valued kernel [Savitzky & Golay, 1964; Press et al., 1992; Schafer, 2011]

$$\{k_{-n_l}, k_{-n_l+1}, \ldots, k_{-1}, k_0, k_1, \ldots, k_{n_r-1}, k_{n_r}\}$$

- Asymmetric kernels are possible unlike the SavitzkyGolayMatrix built-in function.
- *ord* is the order of the smoothing polynomial (and the highest conserved moment, usually 2 or 4); maximum order *ord* of the smoothing polynomial is $n_l + n_r$.
- $\{n_l, n_r\}$ gives the kernel wings lengths—the numbers of leftward (future) and rightward (past) data points, $n_l + n_r + 1$ being the kernel total size.
- Option "Derivs" → {drv$_1$, drv$_2$, …} specifies the list of derivative orders desired (default {0} computes smoothed function only). Maximum derivative order is the order *ord* of the smoothing polynomial.
- Option "Indexed" → False (default) returns a kernel (a list of kernels) if the "Derivs" list has one element (more than one elements). "Indexed" → True always returns an association <|drv$_1$ → ker$_1$, drv$_2$ → ker$_2$, …|>.

Examples

Clearing global symbols:

In[569]:=
```
Clear[b, s, i]
```

Basic usage

Creating symmetric smoothing kernel:

In[570]:=
```
sSavGolKernel1D[2, {2, 2}]
```

Out[570]=
$$\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}$$

Creating asymetric smoothing kernel:

In[571]:=
```
sSavGolKernel1D[2, {1, 3}]
```

Out[571]=
$$\left\{\frac{9}{35}, \frac{13}{35}, \frac{12}{35}, \frac{6}{35}, -\frac{1}{7}\right\}$$

Creating kernel for computing the 1st derivative:

In[572]:=
```
sSavGolKernel1D[2, {2, 2}, "Derivs" → {1}]
```

Out[572]=
$$\left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}$$

Creating list of kernels for smoothing and for computing 1st and 2nd derivatives:

In[573]:=
```
sSavGolKernel1D[2, {2, 2}, "Derivs" → {0, 1, 2}]
```

Out[573]=
$$\left\{\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}, \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}$$

Symmetric kernels, smoothing only

Maximum smoothing polynomial order is $n_l + n_r$. The integers representing the kernel orders are on the left side of the vertical bar, the corresponding kernels are on the rigt side of the vertical bar:

In[574]:=
```
With[{nl = 1, nr = 1}, sSavGolKernel1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
  TableForm[#, TableHeadings → {Range[0, nl + nr], None},
    TableSpacing → {2.5, 2}, TableAlignments → Center] &]
```

Out[574]//TableForm=

0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
2	0	1	0

```
In[575]:= With[{nl = 2, nr = 2}, sSavGolKernel1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
            TableForm[#, TableHeadings → {Range[0, nl + nr], None},
              TableSpacing → {2.5, 2}, TableAlignments → Center] &]
```

Out[575]//TableForm=

0	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
1	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
2	$-\frac{3}{35}$	$\frac{12}{35}$	$\frac{17}{35}$	$\frac{12}{35}$	$-\frac{3}{35}$
3	$-\frac{3}{35}$	$\frac{12}{35}$	$\frac{17}{35}$	$\frac{12}{35}$	$-\frac{3}{35}$
4	0	0	1	0	0

```
In[576]:= With[{nl = 4, nr = 4}, sSavGolKernel1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
            TableForm[#, TableHeadings → {Range[0, nl + nr], None},
              TableSpacing → {2.5, 2}, TableAlignments → Center] &]
```

Out[576]//TableForm=

0	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
1	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
2	$-\frac{1}{11}$	$\frac{2}{33}$	$\frac{13}{77}$	$\frac{18}{77}$	$\frac{59}{231}$	$\frac{18}{77}$	$\frac{13}{77}$	$\frac{2}{33}$	$-\frac{1}{11}$
3	$-\frac{1}{11}$	$\frac{2}{33}$	$\frac{13}{77}$	$\frac{18}{77}$	$\frac{59}{231}$	$\frac{18}{77}$	$\frac{13}{77}$	$\frac{2}{33}$	$-\frac{1}{11}$
4	$\frac{5}{143}$	$-\frac{5}{39}$	$\frac{10}{143}$	$\frac{45}{143}$	$\frac{179}{429}$	$\frac{45}{143}$	$\frac{10}{143}$	$-\frac{5}{39}$	$\frac{5}{143}$
5	$\frac{5}{143}$	$-\frac{5}{39}$	$\frac{10}{143}$	$\frac{45}{143}$	$\frac{179}{429}$	$\frac{45}{143}$	$\frac{10}{143}$	$-\frac{5}{39}$	$\frac{5}{143}$
6	$-\frac{7}{1287}$	$\frac{56}{1287}$	$-\frac{196}{1287}$	$\frac{392}{1287}$	$\frac{797}{1287}$	$\frac{392}{1287}$	$-\frac{196}{1287}$	$\frac{56}{1287}$	$-\frac{7}{1287}$
7	$-\frac{7}{1287}$	$\frac{56}{1287}$	$-\frac{196}{1287}$	$\frac{392}{1287}$	$\frac{797}{1287}$	$\frac{392}{1287}$	$-\frac{196}{1287}$	$\frac{56}{1287}$	$-\frac{7}{1287}$
8	0	0	0	0	1	0	0	0	0

Symmetric kernels, derivatives

Maximum derivative order $ord_{\max}$ is the smoothing polynomial order. Output rows represent the order 0, …, $ord_{\max}$, individual lists in the rows are kernels corresponding to 0th, …, *ord*-th derivative:

In[577]:=
```
With[{nl = 1, nr = 1}, sSavGolKernel1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
    Range[0, nl + nr]] // Column
```

Out[577]=

$$\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$$

$$\left\{\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}, \left\{\frac{1}{2}, 0, -\frac{1}{2}\right\}\right\}$$

$$\left\{\{0, 1, 0\}, \left\{\frac{1}{2}, 0, -\frac{1}{2}\right\}, \{1, -2, 1\}\right\}$$

In[578]:=
```
With[{nl = 2, nr = 2}, sSavGolKernel1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
    Range[0, nl + nr]] // Column // Style[#, 7] &
```

Out[578]=

$$\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}$$

$$\left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}, \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}$$

$$\left\{\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}, \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}$$

$$\left\{\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}, \left\{-\frac{1}{12}, \frac{2}{3}, 0, -\frac{2}{3}, \frac{1}{12}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}, \left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}\right\}$$

$$\left\{\{0, 0, 1, 0, 0\}, \left\{-\frac{1}{12}, \frac{2}{3}, 0, -\frac{2}{3}, \frac{1}{12}\right\}, \left\{-\frac{1}{12}, \frac{4}{3}, -\frac{5}{2}, \frac{4}{3}, -\frac{1}{12}\right\}, \left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}, \{1, -4, 6, -4, 1\}\right\}$$

In[579]:=
```
With[{nl = 3, nr = 3}, sSavGolKernel1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
    Range[0, nl + nr]] // Column // Style[#, 7] &
```

Out[579]=

$$\left\{\frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}\right\}$$

$$\left\{\left\{\frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}\right\}, \left\{\frac{3}{28}, \frac{1}{14}, \frac{1}{28}, 0, -\frac{1}{28}, -\frac{1}{14}, -\frac{3}{28}\right\}\right\}$$

$$\left\{\left\{-\frac{2}{21}, \frac{1}{7}, \frac{2}{7}, \frac{1}{3}, \frac{2}{7}, \frac{1}{7}, -\frac{2}{21}\right\}, \left\{\frac{3}{28}, \frac{1}{14}, \frac{1}{28}, 0, -\frac{1}{28}, -\frac{1}{14}, -\frac{3}{28}\right\}, \left\{\frac{5}{42}, 0, -\frac{1}{14}, -\frac{2}{21}, -\frac{1}{14}, 0, \frac{5}{42}\right\}\right\}$$

$$\left\{\left\{-\frac{2}{21}, \frac{1}{7}, \frac{2}{7}, \frac{1}{3}, \frac{2}{7}, \frac{1}{7}, -\frac{2}{21}\right\}, \left\{-\frac{11}{126}, \frac{67}{252}, \frac{29}{126}, 0, -\frac{29}{126}, -\frac{67}{252}, \frac{11}{126}\right\},\right.$$
$$\left.\left\{\frac{5}{42}, 0, -\frac{1}{14}, -\frac{2}{21}, -\frac{1}{14}, 0, \frac{5}{42}\right\}, \left\{\frac{1}{6}, -\frac{1}{6}, -\frac{1}{6}, 0, \frac{1}{6}, \frac{1}{6}, -\frac{1}{6}\right\}\right\}$$

$$\left\{\left\{\frac{5}{231}, -\frac{10}{77}, \frac{25}{77}, \frac{131}{231}, \frac{25}{77}, -\frac{10}{77}, \frac{5}{231}\right\}, \left\{-\frac{11}{126}, \frac{67}{252}, \frac{29}{126}, 0, -\frac{29}{126}, -\frac{67}{252}, \frac{11}{126}\right\},\right.$$
$$\left.\left\{-\frac{13}{132}, \frac{67}{132}, -\frac{19}{132}, -\frac{35}{66}, -\frac{19}{132}, \frac{67}{132}, -\frac{13}{132}\right\}, \left\{\frac{1}{6}, -\frac{1}{6}, -\frac{1}{6}, 0, \frac{1}{6}, \frac{1}{6}, -\frac{1}{6}\right\}, \left\{\frac{3}{11}, -\frac{7}{11}, \frac{1}{11}, \frac{6}{11}, \frac{1}{11}, -\frac{7}{11}, \frac{3}{11}\right\}\right\}$$

$$\left\{\left\{\frac{5}{231}, -\frac{10}{77}, \frac{25}{77}, \frac{131}{231}, \frac{25}{77}, -\frac{10}{77}, \frac{5}{231}\right\}, \left\{\frac{1}{60}, -\frac{3}{20}, \frac{3}{4}, 0, -\frac{3}{4}, \frac{3}{20}, -\frac{1}{60}\right\},\right.$$
$$\left.\left\{-\frac{13}{132}, \frac{67}{132}, -\frac{19}{132}, -\frac{35}{66}, -\frac{19}{132}, \frac{67}{132}, -\frac{13}{132}\right\}, \left\{-\frac{1}{8}, 1, -\frac{13}{8}, 0, \frac{13}{8}, -1, \frac{1}{8}\right\},\right.$$
$$\left.\left\{\frac{3}{11}, -\frac{7}{11}, \frac{1}{11}, \frac{6}{11}, \frac{1}{11}, -\frac{7}{11}, \frac{3}{11}\right\}, \left\{\frac{1}{2}, -2, \frac{5}{2}, 0, -\frac{5}{2}, 2, -\frac{1}{2}\right\}\right\}$$

$$\left\{\{0, 0, 0, 1, 0, 0, 0\}, \left\{\frac{1}{60}, -\frac{3}{20}, \frac{3}{4}, 0, -\frac{3}{4}, \frac{3}{20}, -\frac{1}{60}\right\},\right.$$
$$\left.\left\{\frac{1}{90}, -\frac{3}{20}, \frac{3}{2}, -\frac{49}{18}, \frac{3}{2}, -\frac{3}{20}, \frac{1}{90}\right\}, \left\{-\frac{1}{8}, 1, -\frac{13}{8}, 0, \frac{13}{8}, -1, \frac{1}{8}\right\},\right.$$
$$\left.\left\{-\frac{1}{6}, 2, -\frac{13}{2}, \frac{28}{3}, -\frac{13}{2}, 2, -\frac{1}{6}\right\}, \left\{\frac{1}{2}, -2, \frac{5}{2}, 0, -\frac{5}{2}, 2, -\frac{1}{2}\right\}, \{1, -6, 15, -20, 15, -6, 1\}\right\}$$

Asymmetric kernels, smoothing only

Maximum smoothing polynomial order is $n_l + n_r$. The integers representing the kernel orders are on the left side of the vertical bar, the corresponding kernels are on the rigt side of the vertical bar:

In[580]:=
```
With[{nl = 1, nr = 3}, sSavGolKernel1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
  TableForm[#, TableHeadings → {Range[0, nl + nr], None},
    TableSpacing → {2.5, 2}, TableAlignments → Center] &]
```

Out[580]//TableForm=

0	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
1	$\frac{2}{5}$	$\frac{3}{10}$	$\frac{1}{5}$	$\frac{1}{10}$	0
2	$\frac{9}{35}$	$\frac{13}{35}$	$\frac{12}{35}$	$\frac{6}{35}$	$-\frac{1}{7}$
3	$\frac{2}{35}$	$\frac{27}{35}$	$\frac{12}{35}$	$-\frac{8}{35}$	$\frac{2}{35}$
4	0	1	0	0	0

In[581]:=
```
With[{nl = 0, nr = 4}, sSavGolKernel1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
  TableForm[#, TableHeadings → {Range[0, nl + nr], None},
    TableSpacing → {2.5, 2}, TableAlignments → Center] &]
```

Out[581]//TableForm=

0	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
1	$\frac{3}{5}$	$\frac{2}{5}$	$\frac{1}{5}$	0	$-\frac{1}{5}$
2	$\frac{31}{35}$	$\frac{9}{35}$	$-\frac{3}{35}$	$-\frac{1}{7}$	$\frac{3}{35}$
3	$\frac{69}{70}$	$\frac{2}{35}$	$-\frac{3}{35}$	$\frac{2}{35}$	$-\frac{1}{70}$
4	1	0	0	0	0

In[582]:=
```
With[{nl = 4, nr = 0}, sSavGolKernel1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
  TableForm[#, TableHeadings → {Range[0, nl + nr], None},
    TableSpacing → {2.5, 2}, TableAlignments → Center] &]
```

Out[582]//TableForm=

0	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
1	$-\frac{1}{5}$	0	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{3}{5}$
2	$\frac{3}{35}$	$-\frac{1}{7}$	$-\frac{3}{35}$	$\frac{9}{35}$	$\frac{31}{35}$
3	$-\frac{1}{70}$	$\frac{2}{35}$	$-\frac{3}{35}$	$\frac{2}{35}$	$\frac{69}{70}$
4	0	0	0	0	1

Asymmetric kernels, derivatives

Maximum derivative order $ord_{\max}$ is the smoothing polynomial order. Output rows represent the order 0, …, $ord_{\max}$, individual lists in the rows are kernels corresponding to 0th, …, *ord*-th derivative:

In[583]:=
```
With[{nl = 1, nr = 3}, sSavGolKernel1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
    Range[0, nl + nr]] // Column // Style[#, 7] &
```

Out[583]=
$$\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}$$
$$\left\{\left\{\frac{2}{5}, \frac{3}{10}, \frac{1}{5}, \frac{1}{10}, 0\right\}, \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}$$
$$\left\{\left\{\frac{9}{35}, \frac{13}{35}, \frac{12}{35}, \frac{6}{35}, -\frac{1}{7}\right\}, \left\{\frac{17}{35}, -\frac{3}{70}, -\frac{2}{7}, -\frac{17}{70}, \frac{3}{35}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}$$
$$\left\{\left\{\frac{2}{35}, \frac{27}{35}, \frac{12}{35}, -\frac{8}{35}, \frac{2}{35}\right\}, \left\{\frac{19}{42}, \frac{1}{42}, -\frac{2}{7}, -\frac{13}{42}, \frac{5}{42}\right\}, \left\{\frac{11}{14}, -\frac{8}{7}, -\frac{2}{7}, \frac{6}{7}, -\frac{3}{14}\right\}, \left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}\right\}$$
$$\left\{\{0, 1, 0, 0, 0\}, \left\{\frac{1}{4}, \frac{5}{6}, -\frac{3}{2}, \frac{1}{2}, -\frac{1}{12}\right\}, \left\{\frac{11}{12}, -\frac{5}{3}, \frac{1}{2}, \frac{1}{3}, -\frac{1}{12}\right\}, \left\{\frac{3}{2}, -5, 6, -3, \frac{1}{2}\right\}, \{1, -4, 6, -4, 1\}\right\}$$

In[584]:=
```
With[{nl = 0, nr = 4}, sSavGolKernel1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
    Range[0, nl + nr]] // Column // Style[#, 7] &
```

Out[584]=
$$\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}$$
$$\left\{\left\{\frac{3}{5}, \frac{2}{5}, \frac{1}{5}, 0, -\frac{1}{5}\right\}, \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}$$
$$\left\{\left\{\frac{31}{35}, \frac{9}{35}, -\frac{3}{35}, -\frac{1}{7}, \frac{3}{35}\right\}, \left\{\frac{27}{35}, -\frac{13}{70}, -\frac{4}{7}, -\frac{27}{70}, \frac{13}{35}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}$$
$$\left\{\left\{\frac{69}{70}, \frac{2}{35}, -\frac{3}{35}, \frac{2}{35}, -\frac{1}{70}\right\}, \left\{\frac{125}{84}, -\frac{34}{21}, -\frac{4}{7}, \frac{22}{21}, -\frac{29}{84}\right\}, \left\{\frac{9}{7}, -\frac{15}{7}, -\frac{2}{7}, \frac{13}{7}, -\frac{5}{7}\right\}, \left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}\right\}$$
$$\left\{\{1, 0, 0, 0, 0\}, \left\{\frac{25}{12}, -4, 3, -\frac{4}{3}, \frac{1}{4}\right\}, \left\{\frac{35}{12}, -\frac{26}{3}, \frac{19}{2}, -\frac{14}{3}, \frac{11}{12}\right\}, \left\{\frac{5}{2}, -9, 12, -7, \frac{3}{2}\right\}, \{1, -4, 6, -4, 1\}\right\}$$

In[585]:=
```
With[{nl = 4, nr = 0}, sSavGolKernel1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
    Range[0, nl + nr]] // Column // Style[#, 7] &
```

Out[585]=
$$\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}$$
$$\left\{\left\{-\frac{1}{5}, 0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}\right\}, \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}$$
$$\left\{\left\{\frac{3}{35}, -\frac{1}{7}, -\frac{3}{35}, \frac{9}{35}, \frac{31}{35}\right\}, \left\{-\frac{13}{35}, \frac{27}{70}, \frac{4}{7}, \frac{13}{70}, -\frac{27}{35}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}$$
$$\left\{\left\{-\frac{1}{70}, \frac{2}{35}, -\frac{3}{35}, \frac{2}{35}, \frac{69}{70}\right\}, \left\{\frac{29}{84}, -\frac{22}{21}, \frac{4}{7}, \frac{34}{21}, -\frac{125}{84}\right\}, \left\{-\frac{5}{7}, \frac{13}{7}, -\frac{2}{7}, -\frac{15}{7}, \frac{9}{7}\right\}, \left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}\right\}$$
$$\left\{\{0, 0, 0, 0, 1\}, \left\{-\frac{1}{4}, \frac{4}{3}, -3, 4, -\frac{25}{12}\right\}, \left\{\frac{11}{12}, -\frac{14}{3}, \frac{19}{2}, -\frac{26}{3}, \frac{35}{12}\right\}, \left\{-\frac{3}{2}, 7, -12, 9, -\frac{5}{2}\right\}, \{1, -4, 6, -4, 1\}\right\}$$

Other features

Here is an example of “indexed output”:

In[586]:=
```
With[{nl = 1, nr = 3},
  sSavGolKernel1D[#, {nl, nr}, "Derivs" → Range[0, #], "Indexed" → True] & /@
   Range[0, nl + nr]] // Style[#, 8] &
```

Out[586]=
$$\left\{\left\langle\left|0 \to \left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right|\right\rangle, \left\langle\left|0 \to \left\{\frac{2}{5}, \frac{3}{10}, \frac{1}{5}, \frac{1}{10}, 0\right\}, 1 \to \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right|\right\rangle,\right.$$
$$\left\langle\left|0 \to \left\{\frac{9}{35}, \frac{13}{35}, \frac{12}{35}, \frac{6}{35}, -\frac{1}{7}\right\}, 1 \to \left\{\frac{17}{35}, -\frac{3}{70}, -\frac{2}{7}, -\frac{17}{70}, \frac{3}{35}\right\}, 2 \to \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right|\right\rangle,$$
$$\left\langle\left|0 \to \left\{\frac{2}{35}, \frac{27}{35}, \frac{12}{35}, -\frac{8}{35}, \frac{2}{35}\right\}, 1 \to \left\{\frac{19}{42}, \frac{1}{42}, -\frac{2}{7}, -\frac{13}{42}, \frac{5}{42}\right\}, 2 \to \left\{\frac{11}{14}, -\frac{8}{7}, -\frac{2}{7}, \frac{6}{7}, -\frac{3}{14}\right\},\right.\right.$$
$$\left.\left.3 \to \left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}\right|\right\rangle, \left\langle\left|0 \to \{0, 1, 0, 0, 0\}, 1 \to \left\{\frac{1}{4}, \frac{5}{6}, -\frac{3}{2}, \frac{1}{2}, -\frac{1}{12}\right\},\right.\right.$$
$$\left.\left.2 \to \left\{\frac{11}{12}, -\frac{5}{3}, \frac{1}{2}, \frac{1}{3}, -\frac{1}{12}\right\}, 3 \to \left\{\frac{3}{2}, -5, 6, -3, \frac{1}{2}\right\}, 4 \to \{1, -4, 6, -4, 1\}\right|\right\rangle\right\}$$

Derivatives are given exactly in the order specified:

```
In[587]:= sSavGolKernel1D[2, {2, 2}, "Derivs" → {2, 0, 1}]
```

$$\text{Out[587]}= \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}, \left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}, \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}$$

```
In[588]:= sSavGolKernel1D[2, {2, 2}, "Derivs" → {2, 0, 1}, "Indexed" → True] //
           Style[#, 8] &
```

$$\text{Out[588]}= \left\langle\left| 2 \to \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}, 0 \to \left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}, 1 \to \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right|\right\rangle$$

The Numerical Recipes Example

Here is the example taken from [Press et al., 1992] for comparison:

```
In[589]:= sSavGolKernel1D[4, {4, 4}] // N // NumberForm[#, {4, 3}] &
```

```
Out[589]//NumberForm=
{0.035, -0.128, 0.070, 0.315, 0.417, 0.315, 0.070, -0.128, 0.035}
```

```
In[590]:= sSavGolKernel1D[4, {5, 5}] // N // NumberForm[#, {4, 3}] &
```

```
Out[590]//NumberForm=
{0.042, -0.105, -0.023, 0.140, 0.280,
 0.333, 0.280, 0.140, -0.023, -0.105, 0.042}
```

Convolving

Here are some examples of convolution with both symmetric and asymetric kernels. All outputs are multiplied by 35 to remove the denominator to improve clarity. Refer to the ListConvolve built-in symbol for detailed explanation.

```
In[591]:= 35 sSavGolKernel1D[2, {2, 2}]
          Row[{ListConvolve[35 sSavGolKernel1D[2, {2, 2}], CharacterRange["a", "h"]] //
             Column,
          ListConvolve[35 sSavGolKernel1D[2, {2, 2}], CharacterRange["a", "h"],
              {-3, 3}] // Column}, Spacer[20]]
```

```
Out[591]= {-3, 12, 17, 12, -3}
```

$$\text{Out[592]}= \begin{array}{ll} & 17\,a + 12\,b - 3\,c - 3\,g + 12\,h \\ & 12\,a + 17\,b + 12\,c - 3\,d - 3\,h \\ -3\,a + 12\,b + 17\,c + 12\,d - 3\,e & -3\,a + 12\,b + 17\,c + 12\,d - 3\,e \\ -3\,b + 12\,c + 17\,d + 12\,e - 3\,f & -3\,b + 12\,c + 17\,d + 12\,e - 3\,f \\ -3\,c + 12\,d + 17\,e + 12\,f - 3\,g & -3\,c + 12\,d + 17\,e + 12\,f - 3\,g \\ -3\,d + 12\,e + 17\,f + 12\,g - 3\,h & -3\,d + 12\,e + 17\,f + 12\,g - 3\,h \\ & -3\,a - 3\,e + 12\,f + 17\,g + 12\,h \\ & 12\,a - 3\,b - 3\,f + 12\,g + 17\,h \end{array}$$

In[593]:=
```
35 sSavGolKernel1D[2, {1, 3}]
Row[{ListConvolve[35 sSavGolKernel1D[2, {1, 3}], CharacterRange["a", "h"]] //
    Column,
ListConvolve[35 sSavGolKernel1D[2, {1, 3}], CharacterRange["a", "h"],
    {-4, 2}] // Column}, Spacer[20]]
```

Out[593]=
$$\{9, 13, 12, 6, -5\}$$

Out[594]=

$-5a+6b+12c+13d+9e$	$13a+9b-5f+6g+12h$
$-5b+6c+12d+13e+9f$	$12a+13b+9c-5g+6h$
$-5c+6d+12e+13f+9g$	$6a+12b+13c+9d-5h$
$-5d+6e+12f+13g+9h$	$-5a+6b+12c+13d+9e$
	$-5b+6c+12d+13e+9f$
	$-5c+6d+12e+13f+9g$
	$-5d+6e+12f+13g+9h$
	$9a-5e+6f+12g+13h$

In[595]:=
```
35 sSavGolKernel1D[2, {4, 0}]
Row[{ListConvolve[35 sSavGolKernel1D[2, {4, 0}], CharacterRange["a", "h"]] //
    Column,
  ListConvolve[35 sSavGolKernel1D[2, {4, 0}], CharacterRange["a", "h"],
    {-1, 5}] // Column}, Spacer[20]]
```

Out[595]=
$$\{3, -5, -3, 9, 31\}$$

Out[596]=

$31a+9b-3c-5d+3e$	$31a+9b-3c-5d+3e$
$31b+9c-3d-5e+3f$	$31b+9c-3d-5e+3f$
$31c+9d-3e-5f+3g$	$31c+9d-3e-5f+3g$
$31d+9e-3f-5g+3h$	$31d+9e-3f-5g+3h$
	$3a+31e+9f-3g-5h$
	$-5a+3b+31f+9g-3h$
	$-3a-5b+3c+31g+9h$
	$9a-3b-5c+3d+31h$

Applications

Savitzky–Golay filtering has a wealth of applications in signal analysis. For further reading, see [Savitzky & Golay, 1964; Press et al., 1992; Schafer, 2011].

Discussion

Comparison to SavitzkyGolayMatrix

The built-in symbol SavitzkyGolayMatrix returns real output by defalut:

In[597]:=
```
SavitzkyGolayMatrix[{2}, 2]
sSavGolKernel1D[2, {2, 2}]
%% === %
```

Out[597]= $\{-0.0857143, 0.342857, 0.485714, 0.342857, -0.0857143\}$

Out[598]= $\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}$

Out[599]= False

Exact output can be forced using WorkingPrecision → Infinity:

In[600]:=
```
SavitzkyGolayMatrix[{2}, 2, WorkingPrecision → Infinity]
sSavGolKernel1D[2, {2, 2}]
%% === %
```

Out[600]= $\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}$

Out[601]= $\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}$

Out[602]= True

While the derivatives given by the built-in SavitzkyGolayMatrix and sSavGolKernel1D exactly coincide for odd-order derivatives (including the 0-th one),

In[603]:=
```
SavitzkyGolayMatrix[{5}, 2, WorkingPrecision → Infinity]
sSavGolKernel1D[2, {5, 5}]
%% === %
```

Out[603]= $\left\{-\frac{12}{143}, \frac{3}{143}, \frac{4}{39}, \frac{23}{143}, \frac{28}{143}, \frac{89}{429}, \frac{28}{143}, \frac{23}{143}, \frac{4}{39}, \frac{3}{143}, -\frac{12}{143}\right\}$

Out[604]= $\left\{-\frac{12}{143}, \frac{3}{143}, \frac{4}{39}, \frac{23}{143}, \frac{28}{143}, \frac{89}{429}, \frac{28}{143}, \frac{23}{143}, \frac{4}{39}, \frac{3}{143}, -\frac{12}{143}\right\}$

Out[605]= True

In[606]:=
```
SavitzkyGolayMatrix[{5}, 2, 2, WorkingPrecision → Infinity]
sSavGolKernel1D[2, {5, 5}, "Derivs" → {2}]
%% === %
```

Out[606]= $\left\{\frac{5}{143}, \frac{2}{143}, -\frac{1}{429}, -\frac{2}{143}, -\frac{3}{143}, -\frac{10}{429}, -\frac{3}{143}, -\frac{2}{143}, -\frac{1}{429}, \frac{2}{143}, \frac{5}{143}\right\}$

Out[607]= $\left\{\frac{5}{143}, \frac{2}{143}, -\frac{1}{429}, -\frac{2}{143}, -\frac{3}{143}, -\frac{10}{429}, -\frac{3}{143}, -\frac{2}{143}, -\frac{1}{429}, \frac{2}{143}, \frac{5}{143}\right\}$

Out[608]= True

the built-in SavitzkyGolayMatrix computes opposite sign for odd-order derivatives:

In[609]:=
```
-SavitzkyGolayMatrix[{2}, 2, 1, WorkingPrecision → Infinity]
sSavGolKernel1D[2, {2, 2}, "Derivs" → {1}]
%% === %
```

Out[609]= $\left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}$

Out[610]= $\left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}$

Out[611]= True

In[612]:=
```
-SavitzkyGolayMatrix[{5}, 2, 1, WorkingPrecision → Infinity]
sSavGolKernel1D[2, {5, 5}, "Derivs" → {1}]
%% === %
```

Out[612]= $\left\{\frac{1}{22}, \frac{2}{55}, \frac{3}{110}, \frac{1}{55}, \frac{1}{110}, 0, -\frac{1}{110}, -\frac{1}{55}, -\frac{3}{110}, -\frac{2}{55}, -\frac{1}{22}\right\}$

Out[613]= $\left\{\frac{1}{22}, \frac{2}{55}, \frac{3}{110}, \frac{1}{55}, \frac{1}{110}, 0, -\frac{1}{110}, -\frac{1}{55}, -\frac{3}{110}, -\frac{2}{55}, -\frac{1}{22}\right\}$

Out[614]= True

In[615]:=
```
-SavitzkyGolayMatrix[{5}, 4, 1, WorkingPrecision → Infinity]
sSavGolKernel1D[4, {5, 5}, "Derivs" → {1}]
%% === %
```

Out[615]= $\left\{-\frac{25}{429}, \frac{49}{858}, \frac{133}{1287}, \frac{503}{5148}, \frac{74}{1287}, 0, -\frac{74}{1287}, -\frac{503}{5148}, -\frac{133}{1287}, -\frac{49}{858}, \frac{25}{429}\right\}$

Out[616]= $\left\{-\frac{25}{429}, \frac{49}{858}, \frac{133}{1287}, \frac{503}{5148}, \frac{74}{1287}, 0, -\frac{74}{1287}, -\frac{503}{5148}, -\frac{133}{1287}, -\frac{49}{858}, \frac{25}{429}\right\}$

Out[617]= True

In the following code we compare the 33-tap, 6th order Savitzky–Golay 1D kernel to the SavitzkyGolayMatrix built-in function sharing the same parameters, incuding all possible derivatives corrected for the odd-order signs (up to order 6):

In[618]:=
```
b = Table[(-1)^i SavitzkyGolayMatrix[{16}, 6, i, WorkingPrecision → Infinity],
    {i, 0, 6}];
s = sSavGolKernel1D[6, {16, 16}, "Derivs" → Range[0, 6]];
b === s
```

Out[620]=
```
True
```

2.7 sSavGolElasticKernels1D

In[621]:=
```
infoAbout@sSavGolElasticKernels1D
```

sSavGolElasticKernels1D[*ord*, {*nl*, *nr*}] gives a bunch of Savitzky–Golay 1D kernels of order *ord* and type {*nl*, *nr*} for the central part of data.

```
Attributes[sSavGolElasticKernels1D] = {Protected, ReadProtected}

Options[sSavGolElasticKernels1D] = {Derivs → {0}, Debug → False}

SyntaxInformation[sSavGolElasticKernels1D] =
 {ArgumentsPattern → {_, {_, _}, OptionsPattern[]}}
```

Details

sSavGolElasticKernels1D[*ord*, {n_l, n_r}, *opts*] returns a bunch of Savitzky–Golay 1D rational-valued kernels [Savitzky & Golay, 1964; Press et al., 1992; Schafer, 2011] of the same order *ord* that consists of one {n_l, n_r}-type C-kernel that operates on the central "core" of data, and sequences of L-kernels and R-kernels (may be empty if either of n_l, n_r is zero) that treat the end effects by incrementing n_l at the expense of n_r (and vice versa) until one of them is zero.

- The arguments are identical to those of sSavGolKernel1D (refer to it for details).
- Provided option "Derivs" → {drv$_1$, drv$_2$, …} (default {0}) is given, the output is returned as ⟨|drv$_1$ → ⟨|"L" → ListOfLeftKernels, "C" → 1-ElemListCentralKernel, "R" → ListOfRightKernels|⟩, drv$_2$ → ⟨|"L" → ListOfLeftKernels, …|⟩, …|⟩
- and can directly be fed into sSavGolConvolve1D.

Examples

Basic usage

In[622]:=
```
sSavGolElasticKernels1D[1, {2, 2}]
```

Out[622]=
$$\langle|\, 0 \to \langle|\, L \to \left\{\left\{-\frac{1}{5}, 0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}\right\}\right\}, \left\{0, \frac{1}{10}, \frac{1}{5}, \frac{3}{10}, \frac{2}{5}\right\}\right\},$$
$$C \to \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}, R \to \left\{\left\{\frac{2}{5}, \frac{3}{10}, \frac{1}{5}, \frac{1}{10}, 0\right\}, \left\{\frac{3}{5}, \frac{2}{5}, \frac{1}{5}, 0, -\frac{1}{5}\right\}\right\}\,|\rangle\,|\rangle$$

In[623]:=
```
sSavGolElasticKernels1D[2, {3, 1}, "Derivs" → {0, 1, 2}] // Style[#, 7] &
```

Out[623]=
$$\langle|\, 0 \to \langle|\, L \to \left\{\left\{\frac{3}{35}, -\frac{1}{7}, -\frac{3}{35}, \frac{9}{35}, \frac{31}{35}\right\}\right\}, C \to \left\{\left\{-\frac{1}{7}, \frac{6}{35}, \frac{12}{35}, \frac{13}{35}, \frac{9}{35}\right\}\right\},$$
$$R \to \left\{\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}, \left\{\frac{9}{35}, \frac{13}{35}, \frac{12}{35}, \frac{6}{35}, -\frac{1}{7}\right\}, \left\{\frac{31}{35}, \frac{9}{35}, -\frac{3}{35}, -\frac{1}{7}, \frac{3}{35}\right\}\right\}\,|\rangle,$$
$$1 \to \langle|\, L \to \left\{\left\{-\frac{13}{35}, \frac{27}{70}, -\frac{4}{7}, \frac{13}{70}, -\frac{27}{35}\right\}\right\}, C \to \left\{\left\{-\frac{3}{35}, \frac{17}{70}, -\frac{2}{7}, \frac{3}{70}, -\frac{17}{35}\right\}\right\},$$
$$R \to \left\{\left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}, \left\{\frac{17}{35}, -\frac{3}{70}, -\frac{2}{7}, -\frac{17}{70}, \frac{3}{35}\right\}, \left\{\frac{27}{35}, -\frac{13}{70}, -\frac{4}{7}, -\frac{27}{70}, \frac{13}{35}\right\}\right\}\,|\rangle,$$
$$2 \to \langle|\, L \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}, C \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\},$$
$$R \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}\,|\rangle\,|\rangle$$

In[624]:=
```
sSavGolElasticKernels1D[2, {1, 1}, "Derivs" → {2, 0}]
% // Dataset // Magnify[#, 2 / 3] &
```

Out[624]=
$$\langle|\, 2 \to \langle|\, L \to \{\{1, -2, 1\}\}, C \to \{\{1, -2, 1\}\}, R \to \{\{1, -2, 1\}\}\,|\rangle,$$
$$0 \to \langle|\, L \to \{\{0, 0, 1\}\}, C \to \{\{0, 1, 0\}\}, R \to \{\{1, 0, 0\}\}\,|\rangle\,|\rangle$$

Out[625]=

	L			C			R		
2	1	−2	1	1	−2	1	1	−2	1
0	0	0	1	0	1	0	1	0	0

Symmetric kernels, smoothing only

Maximum smoothing polynomial order is $n_l + n_r$:

In[626]:=
```
With[{nl = 1, nr = 1},
 sSavGolElasticKernels1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
  TableForm[#, TableHeadings → {Range[0, nl + nr], None},
    TableSpacing → {2.5, 2}] &]
```

Out[626]//TableForm=

0	$\langle|\, 0 \to \langle|\, L \to \left\{\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}\right\}, C \to \left\{\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}\right\}, R \to \left\{\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}\right\}\,|\rangle\,|\rangle$
1	$\langle|\, 0 \to \langle|\, L \to \left\{\left\{-\frac{1}{6}, \frac{1}{3}, \frac{5}{6}\right\}\right\}, C \to \left\{\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}\right\}, R \to \left\{\left\{\frac{5}{6}, \frac{1}{3}, -\frac{1}{6}\right\}\right\}\,|\rangle\,|\rangle$
2	$\langle|\, 0 \to \langle|\, L \to \{\{0, 0, 1\}\}, C \to \{\{0, 1, 0\}\}, R \to \{\{1, 0, 0\}\}\,|\rangle\,|\rangle$

In[627]:=
```
With[{nl = 2, nr = 2},
  sSavGolElasticKernels1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
   TableForm[#, TableHeadings → {Range[0, nl + nr], None},
     TableSpacing → {2.5, 2}] &] // Style[#, 5] &
```

Out[627]=

$$\begin{array}{l}
0\ \left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}, C \to \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}, R \to \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}\right|\right\rangle, \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}\right|\right\rangle \\
1\ \left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{-\frac{1}{5}, 0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}\right\}\right\}, C \to \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}, R \to \left\{\left\{\frac{2}{5}, \frac{3}{10}, \frac{1}{5}, \frac{1}{10}, 0\right\}\right\}\right|\right\rangle, \left\{\left\{\frac{3}{5}, \frac{2}{5}, \frac{1}{5}, 0, -\frac{1}{5}\right\}\right\}\right|\right\rangle \\
2\ \left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{\frac{3}{35}, -\frac{1}{7}, -\frac{3}{35}, \frac{9}{35}, \frac{31}{35}\right\}\right\}, C \to \left\{\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}\right\}, R \to \left\{\left\{\frac{9}{35}, \frac{13}{35}, \frac{12}{35}, \frac{6}{35}, -\frac{1}{7}\right\}, \left\{\frac{31}{35}, \frac{9}{35}, -\frac{3}{35}, -\frac{1}{7}, \frac{3}{35}\right\}\right\}\right|\right\rangle\right|\right\rangle \\
3\ \left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{-\frac{1}{70}, \frac{2}{35}, -\frac{3}{35}, \frac{2}{35}, \frac{69}{70}\right\}, \left\{\frac{2}{35}, -\frac{8}{35}, \frac{12}{35}, \frac{27}{35}, \frac{2}{35}\right\}\right\}, C \to \left\{\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}\right\}, R \to \left\{\left\{\frac{2}{35}, \frac{27}{35}, \frac{12}{35}, -\frac{8}{35}, \frac{2}{35}\right\}, \left\{\frac{69}{70}, \frac{2}{35}, -\frac{3}{35}, \frac{2}{35}, -\frac{1}{70}\right\}\right\}\right|\right\rangle\right|\right\rangle \\
4\ \left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{0, 0, 0, 0, 1\right\}, \left\{0, 0, 0, 1, 0\right\}\right\}, C \to \left\{\left\{0, 0, 1, 0, 0\right\}\right\}, R \to \left\{\left\{0, 1, 0, 0, 0\right\}, \left\{1, 0, 0, 0, 0\right\}\right\}\right|\right\rangle\right|\right\rangle
\end{array}$$

Symmetric kernels, derivatives

Maximum derivative order is the smoothing polynomial order:

In[628]:=
```
With[{nl = 1, nr = 1},
  sSavGolElasticKernels1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
   Range[0, nl + nr]] // Column
```

Out[628]=

$$\begin{array}{l}
\left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}\right\}, C \to \left\{\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}\right\}, R \to \left\{\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}\right\}\right|\right\rangle\right|\right\rangle \\
\left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{-\frac{1}{6}, \frac{1}{3}, \frac{5}{6}\right\}\right\}, C \to \left\{\left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}\right\}, R \to \left\{\left\{\frac{5}{6}, \frac{1}{3}, -\frac{1}{6}\right\}\right\}\right|\right\rangle,\right. \\
\left.1 \to \left\langle\left|L \to \left\{\left\{\frac{1}{2}, 0, -\frac{1}{2}\right\}\right\}, C \to \left\{\left\{\frac{1}{2}, 0, -\frac{1}{2}\right\}\right\}, R \to \left\{\left\{\frac{1}{2}, 0, -\frac{1}{2}\right\}\right\}\right|\right\rangle\right|\right\rangle \\
\left\langle\left|0 \to \langle| L \to \{\{0, 0, 1\}\}, C \to \{\{0, 1, 0\}\}, R \to \{\{1, 0, 0\}\} |\rangle,\right.\right. \\
1 \to \left\langle\left|L \to \left\{\left\{-\frac{1}{2}, 2, -\frac{3}{2}\right\}\right\}, C \to \left\{\left\{\frac{1}{2}, 0, -\frac{1}{2}\right\}\right\}, R \to \left\{\left\{\frac{3}{2}, -2, \frac{1}{2}\right\}\right\}\right|\right\rangle, \\
\left.\left.2 \to \langle| L \to \{\{1, -2, 1\}\}, C \to \{\{1, -2, 1\}\}, R \to \{\{1, -2, 1\}\} |\rangle\right|\right\rangle
\end{array}$$

In[629]:=
```
With[{nl = 2, nr = 2},
  sSavGolElasticKernels1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
   Range[0, nl + nr]] // Column // Style[#, 5] &
```

Out[629]=

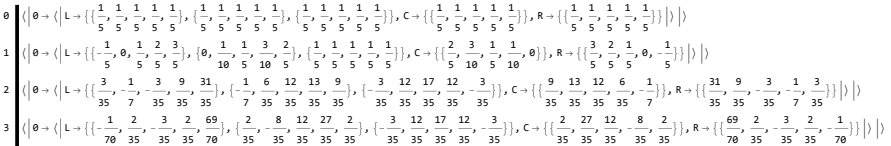
$$\begin{array}{l}
\left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}, C \to \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}, R \to \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}\right|\right\rangle, \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}\right|\right\rangle \\
\left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{-\frac{1}{5}, 0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}\right\}\right\}, C \to \left\{\left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}\right\}, R \to \left\{\left\{\frac{2}{5}, \frac{3}{10}, \frac{1}{5}, \frac{1}{10}, 0\right\}, \left\{\frac{3}{5}, \frac{2}{5}, \frac{1}{5}, 0, -\frac{1}{5}\right\}\right\}\right|\right\rangle,\right. \\
1 \to \left\langle\left|L \to \left\{\left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}, \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}, C \to \left\{\left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}, R \to \left\{\left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}, \left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}\right|\right\rangle\right|\right\rangle \\
\left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{\frac{3}{35}, -\frac{1}{7}, -\frac{3}{35}, \frac{9}{35}, \frac{31}{35}\right\}, \left\{-\frac{1}{7}, \frac{6}{35}, \frac{12}{35}, \frac{13}{35}, \frac{9}{35}\right\}\right\}, C \to \left\{\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}\right\}, R \to \left\{\left\{\frac{9}{35}, \frac{13}{35}, \frac{12}{35}, \frac{6}{35}, -\frac{1}{7}\right\}, \left\{\frac{31}{35}, \frac{9}{35}, -\frac{3}{35}, -\frac{1}{7}, \frac{3}{35}\right\}\right\}\right|\right\rangle,\right. \\
1 \to \left\langle\left|L \to \left\{\left\{-\frac{13}{35}, \frac{27}{70}, \frac{4}{7}, \frac{13}{70}, -\frac{27}{35}\right\}, \left\{-\frac{3}{35}, \frac{17}{70}, \frac{2}{7}, \frac{3}{70}, -\frac{17}{35}\right\}\right\}, C \to \left\{\left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}, R \to \left\{\left\{\frac{17}{35}, -\frac{3}{70}, -\frac{2}{7}, -\frac{17}{70}, \frac{3}{35}\right\}, \left\{\frac{27}{35}, -\frac{13}{70}, -\frac{4}{7}, -\frac{27}{70}, \frac{13}{35}\right\}\right\}\right|\right\rangle,\right. \\
2 \to \left\langle\left|L \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}, C \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}, R \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}\right|\right\rangle\right|\right\rangle \\
\left\langle\left|0 \to \left\langle\left|L \to \left\{\left\{-\frac{1}{70}, \frac{2}{35}, -\frac{3}{35}, \frac{2}{35}, \frac{69}{70}\right\}, \left\{\frac{2}{35}, -\frac{8}{35}, \frac{12}{35}, \frac{27}{35}, \frac{2}{35}\right\}\right\}, C \to \left\{\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}\right\}, R \to \left\{\left\{\frac{2}{35}, \frac{27}{35}, \frac{12}{35}, -\frac{8}{35}, \frac{2}{35}\right\}, \left\{\frac{69}{70}, \frac{2}{35}, -\frac{3}{35}, \frac{2}{35}, -\frac{1}{70}\right\}\right\}\right|\right\rangle,\right. \\
1 \to \left\langle\left|L \to \left\{\left\{\frac{29}{84}, -\frac{22}{21}, \frac{4}{7}, \frac{34}{21}, -\frac{125}{84}\right\}, \left\{-\frac{5}{42}, \frac{13}{42}, \frac{2}{7}, -\frac{1}{42}, -\frac{19}{42}\right\}\right\}, C \to \left\{\left\{-\frac{1}{12}, \frac{2}{3}, 0, -\frac{2}{3}, \frac{1}{12}\right\}\right\}, R \to \left\{\left\{\frac{19}{42}, \frac{1}{42}, -\frac{2}{7}, -\frac{13}{42}, \frac{5}{42}\right\}, \left\{\frac{125}{84}, -\frac{34}{21}, \frac{4}{7}, \frac{22}{21}, -\frac{29}{84}\right\}\right\}\right|\right\rangle,\right. \\
2 \to \left\langle\left|L \to \left\{\left\{-\frac{5}{7}, \frac{13}{7}, -\frac{2}{7}, -\frac{15}{7}, \frac{9}{7}\right\}, \left\{-\frac{3}{14}, \frac{6}{7}, -\frac{2}{7}, -\frac{8}{7}, \frac{11}{14}\right\}\right\}, C \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}, R \to \left\{\left\{\frac{11}{14}, -\frac{8}{7}, \frac{2}{7}, -\frac{6}{7}, \frac{3}{14}\right\}, \left\{\frac{9}{7}, -\frac{15}{7}, \frac{2}{7}, -\frac{13}{7}, \frac{5}{7}\right\}\right\}\right|\right\rangle,\right. \\
3 \to \left\langle\left|L \to \left\{\left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}, \left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}\right\}, C \to \left\{\left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}\right\}, R \to \left\{\left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}, \left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}\right\}\right|\right\rangle\right|\right\rangle \\
\left\langle\left|0 \to \langle| L \to \{\{0, 0, 0, 0, 1\}, \{0, 0, 0, 1, 0\}\}, C \to \{\{0, 0, 1, 0, 0\}\}, R \to \{\{0, 1, 0, 0, 0\}, \{1, 0, 0, 0, 0\}\} |\rangle,\right.\right. \\
1 \to \left\langle\left|L \to \left\{\left\{-\frac{1}{4}, \frac{4}{3}, -3, 4, -\frac{25}{12}\right\}, \left\{\frac{1}{12}, -\frac{1}{2}, \frac{3}{2}, -\frac{5}{6}, \frac{1}{4}\right\}\right\}, C \to \left\{\left\{-\frac{1}{12}, \frac{2}{3}, 0, -\frac{2}{3}, \frac{1}{12}\right\}\right\}, R \to \left\{\left\{\frac{1}{4}, \frac{5}{6}, -\frac{3}{2}, \frac{1}{2}, -\frac{1}{12}\right\}, \left\{\frac{25}{12}, -4, 3, -\frac{4}{3}, \frac{1}{4}\right\}\right\}\right|\right\rangle,\right. \\
2 \to \left\langle\left|L \to \left\{\left\{\frac{11}{12}, -\frac{14}{3}, \frac{19}{2}, -\frac{26}{3}, \frac{35}{12}\right\}, \left\{-\frac{1}{12}, \frac{1}{3}, \frac{1}{2}, -\frac{5}{3}, \frac{11}{12}\right\}\right\}, C \to \left\{\left\{-\frac{1}{12}, \frac{4}{3}, -\frac{5}{2}, \frac{4}{3}, -\frac{1}{12}\right\}\right\}, R \to \left\{\left\{\frac{11}{12}, -\frac{5}{3}, \frac{1}{2}, \frac{1}{3}, -\frac{1}{12}\right\}, \left\{\frac{35}{12}, -\frac{26}{3}, \frac{19}{2}, -\frac{14}{3}, \frac{11}{12}\right\}\right\}\right|\right\rangle,\right. \\
3 \to \left\langle\left|L \to \left\{\left\{-\frac{3}{2}, 7, -12, 9, -\frac{5}{2}\right\}, \left\{-\frac{1}{2}, 3, -6, 5, -\frac{3}{2}\right\}\right\}, C \to \left\{\left\{\frac{1}{2}, -1, 0, 1, -\frac{1}{2}\right\}\right\}, R \to \left\{\left\{\frac{3}{2}, -5, 6, -3, \frac{1}{2}\right\}, \left\{\frac{5}{2}, -9, 12, -7, \frac{3}{2}\right\}\right\}\right|\right\rangle,\right. \\
\left.\left.4 \to \langle| L \to \{\{1, -4, 6, -4, 1\}, \{1, -4, 6, -4, 1\}\}, C \to \{\{1, -4, 6, -4, 1\}\}, R \to \{\{1, -4, 6, -4, 1\}, \{1, -4, 6, -4, 1\}\} |\rangle\right|\right\rangle
\end{array}$$

Asymmetric kernels, smoothing only

Maximum smoothing polynomial order is $n_l + n_r$:

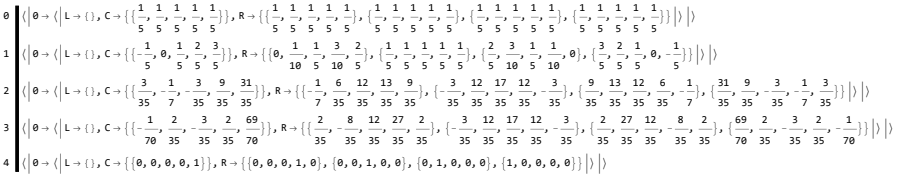
In[630]:=
```
With[{nl = 1, nr = 3},
  sSavGolElasticKernels1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
   TableForm[#, TableHeadings → {Range[0, nl + nr], None},
     TableSpacing → {2.5, 2}] &] // Style[#, 4] &
```

Out[630]=



In[631]:=
```
With[{nl = 4, nr = 0},
  sSavGolElasticKernels1D[#, {nl, nr}] & /@ Range[0, nl + nr] //
   TableForm[#, TableHeadings → {Range[0, nl + nr], None},
     TableSpacing → {2.5, 2}] &] // Style[#, 4] &
```

Out[631]=

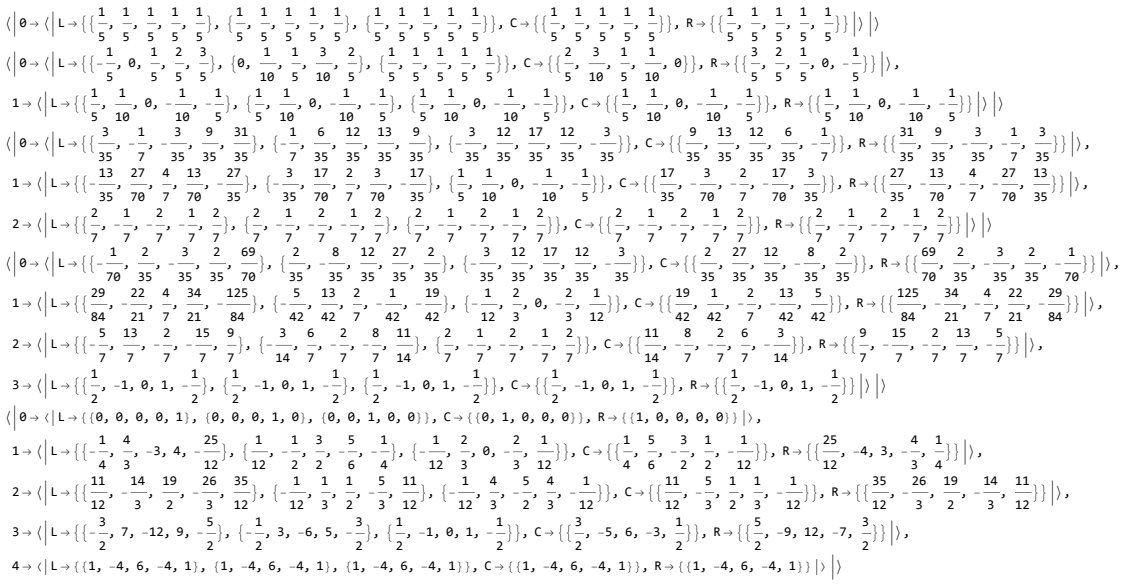


Asymmetric kernels, derivatives

Maximum derivative order is the smoothing polynomial order:

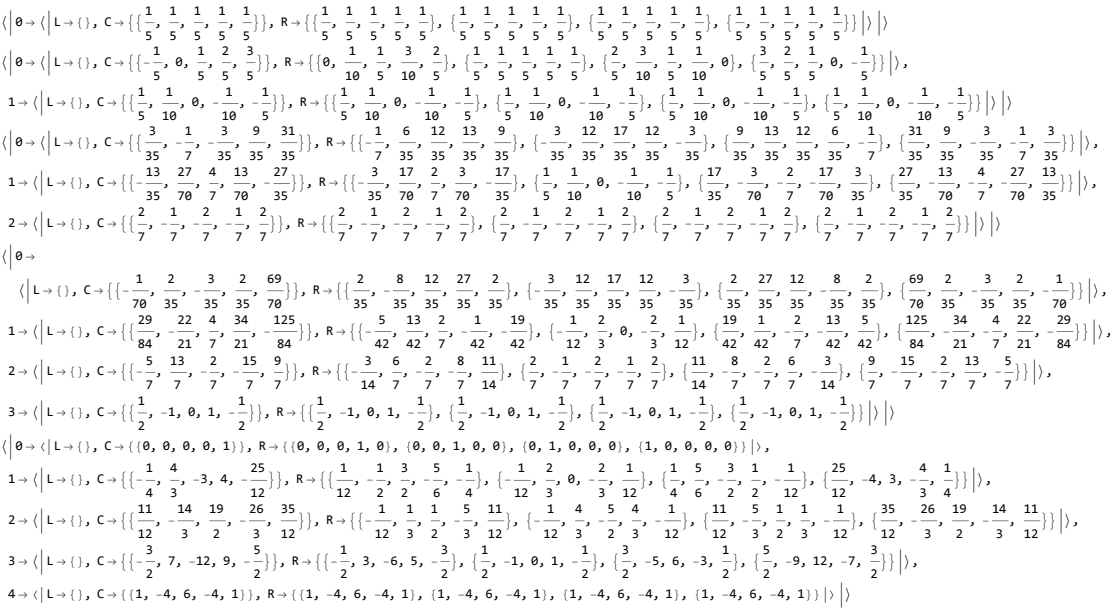
In[632]:=
```
With[{nl = 1, nr = 3},
  sSavGolElasticKernels1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
   Range[0, nl + nr]] // Column // Style[#, 5] &
```

Out[632]=



In[633]:=
```
With[{nl = 4, nr = 0},
    sSavGolElasticKernels1D[#, {nl, nr}, "Derivs" → Range[0, #]] & /@
     Range[0, nl + nr]] // Column // Style[#, 5] &
```

Out[633]=



Applications

Savitzky–Golay filtering has a wealth of applications in signal analysis. For further reading, see [Savitzky & Golay, 1964; Press et al., 1992; Schafer, 2011].

2.8 sSavGolBufferConvolve1D

In[634]:=
```
infoAbout@sSavGolBufferConvolve1D
```

sSavGolBufferConvolve1D[*kers*, *data*] convolves 1D
data with the bunch of kernels *kers* = sSavGolElasticKernels1D[…].

```
Attributes[sSavGolBufferConvolve1D] = {Protected, ReadProtected}

Options[sSavGolBufferConvolve1D] =
 {Debug → False, AspectRatio → False, Indexed → False}

SyntaxInformation[sSavGolBufferConvolve1D] =
 {ArgumentsPattern → {_, _, OptionsPattern[]}}
```

Details

sSavGolBufferConvolve1D[*kers*, *data*, *opts*] uses *kers* = sSavGolElasticKernels1D[…] to convolve relevant Savitzky–Golay 1D kernels [Savitzky & Golay, 1964; Press et al., 1992; Schafer, 2011] contained in the *kers* bunch with *data*, carrying out filtering and/or estimating derivatives

that treats end effects in a confined, "buffered" way.

- Multiply the output by 1/(sampling interval)^j to get true j-th derivative.
- Option AspectRatio → *asp* (surprisingly) tells whether to estimate arc curvature (a nonzero aspect ratio *asp*, or True that sets *asp* = 1/GoldenRatio) or not (False, default). A negative value forces scaling based on raw rather than smoothed data.
- Option "Indexed" → False (default) returns output data (list of output data) if the "Derivs" list has one element (more than one elements). "Indexed" → True always returns an association ⟨| drv$_1$ → outdat$_1$, … |⟩.
- If curvature is computed, −1 → outdat$_{curv}$ (the very last) is appended, and *kers* must contain kernels for derivative orders 0, 1, and 2 (with *asp* negative, 1 and 2 are sufficient).
- If extra order 3 is included, a zero-pass curvature extrema indicator is appended as -2 → outdat$_{zero}$ (as penultimate).

Examples

Clearing global symbols:

In[635]:=
```
Clear[kers, optsListPlot, colors, legends, legends2, par, hw, std,
  polord, noise, t, x, Δ, order, ord, data, k, out, optsPlot,
  optsListLinePlot, epspPlot, measurement, measurementPlot, nl, nr,
  c, c1, c2, filterOutput, filterOutputPlot, filterOutput1,
  filterOutput1Plot, filterOutput2, filterOutput2Plot, filterOutputAC,
  filterOutputACPlot, filterOutputZXPlot, funcPlot, func1Plot,
  func2Plot, func3Plot, func4Plot, func5Plot, noisydata, noisydataPlot,
  smoothed, smoothedPlot, sgFilter, drv1, drv1Plot, drv2, drv2Plot,
  drv3, drv3Plot, drv4, drv4Plot, drv5, drv5Plot, curv, curvPlot, zeroxPlot];
ClearAll[epsp, epsp1, epsp2, func, func1, func2, func3, func4, func5];
```

Basic usage

In[637]:=
```
(kers = sSavGolElasticKernels1D[2, {1, 1}, "Derivs" → Range[0, 2]]) // Normal //
 Column
sSavGolBufferConvolve1D[kers, CharacterRange["a", "h"], "Indexed" → False] //
 Column
```

Out[637]=
$$0 \to \langle| L \to \{\{0, 0, 1\}\}, C \to \{\{0, 1, 0\}\}, R \to \{\{1, 0, 0\}\} |\rangle$$
$$1 \to \left\langle\left| L \to \left\{\left\{-\frac{1}{2}, 2, -\frac{3}{2}\right\}\right\}, C \to \left\{\left\{\frac{1}{2}, 0, -\frac{1}{2}\right\}\right\}, R \to \left\{\left\{\frac{3}{2}, -2, \frac{1}{2}\right\}\right\} \right|\right\rangle$$
$$2 \to \langle| L \to \{\{1, -2, 1\}\}, C \to \{\{1, -2, 1\}\}, R \to \{\{1, -2, 1\}\} |\rangle$$

Out[638]=
$$\{a, b, c, d, e, f, g, h\}$$
$$\left\{-\frac{3a}{2} + 2b - \frac{c}{2}, -\frac{a}{2} + \frac{c}{2}, -\frac{b}{2} + \frac{d}{2}, -\frac{c}{2} + \frac{e}{2}, -\frac{d}{2} + \frac{f}{2}, -\frac{e}{2} + \frac{g}{2}, -\frac{f}{2} + \frac{h}{2}, \frac{f}{2} - 2g + \frac{3h}{2}\right\}$$
$$\{a - 2b + c, a - 2b + c, b - 2c + d,$$
$$c - 2d + e, d - 2e + f, e - 2f + g, f - 2g + h, f - 2g + h\}$$

In[639]:=
```
(kers = sSavGolElasticKernels1D[2, {2, 2}, "Derivs" → Range[0, 2]]) // Normal //
  Column // Style[#, 8] &
sSavGolBufferConvolve1D[kers, CharacterRange["a", "h"], "Indexed" → False] //
  Column // Style[#, 8] &
```

Out[639]=
$$0 \to \left\langle\left| L \to \left\{\left\{\frac{3}{35}, -\frac{1}{7}, -\frac{3}{35}, \frac{9}{35}, \frac{31}{35}\right\}, \left\{-\frac{1}{7}, \frac{6}{35}, \frac{12}{35}, \frac{13}{35}, \frac{9}{35}\right\}\right\},\right.\right.$$
$$\left.\left. C \to \left\{\left\{-\frac{3}{35}, \frac{12}{35}, \frac{17}{35}, \frac{12}{35}, -\frac{3}{35}\right\}\right\}, R \to \left\{\left\{\frac{9}{35}, \frac{13}{35}, \frac{12}{35}, \frac{6}{35}, -\frac{1}{7}\right\}, \left\{\frac{31}{35}, \frac{9}{35}, -\frac{3}{35}, -\frac{1}{7}, \frac{3}{35}\right\}\right\} \right|\right\rangle$$
$$1 \to \left\langle\left| L \to \left\{\left\{-\frac{13}{35}, \frac{27}{70}, \frac{4}{7}, \frac{13}{70}, -\frac{27}{35}\right\}, \left\{-\frac{3}{35}, \frac{17}{70}, \frac{2}{7}, \frac{3}{70}, -\frac{17}{35}\right\}\right\},\right.\right.$$
$$\left.\left. C \to \left\{\left\{\frac{1}{5}, \frac{1}{10}, 0, -\frac{1}{10}, -\frac{1}{5}\right\}\right\}, R \to \left\{\left\{\frac{17}{35}, -\frac{3}{70}, -\frac{2}{7}, -\frac{17}{70}, \frac{3}{35}\right\}, \left\{\frac{27}{35}, -\frac{13}{70}, -\frac{4}{7}, -\frac{27}{70}, \frac{13}{35}\right\}\right\} \right|\right\rangle$$
$$2 \to \left\langle\left| L \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\},\right.\right.$$
$$\left.\left. C \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\}, R \to \left\{\left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}, \left\{\frac{2}{7}, -\frac{1}{7}, -\frac{2}{7}, -\frac{1}{7}, \frac{2}{7}\right\}\right\} \right|\right\rangle$$

Out[640]=
$$\left\{\frac{31\,a}{35} + \frac{9\,b}{35} - \frac{3\,c}{35} - \frac{d}{7} + \frac{3\,e}{35}, \frac{9\,a}{35} + \frac{13\,b}{35} + \frac{12\,c}{35} + \frac{6\,d}{35} - \frac{e}{7},\right.$$
$$-\frac{3\,a}{35} + \frac{12\,b}{35} + \frac{17\,c}{35} + \frac{12\,d}{35} - \frac{3\,e}{35}, -\frac{3\,b}{35} + \frac{12\,c}{35} + \frac{17\,d}{35} + \frac{12\,e}{35} - \frac{3\,f}{35}, -\frac{3\,c}{35} + \frac{12\,d}{35} + \frac{17\,e}{35} + \frac{12\,f}{35} - \frac{3\,g}{35},$$
$$\left.-\frac{3\,d}{35} + \frac{12\,e}{35} + \frac{17\,f}{35} + \frac{12\,g}{35} - \frac{3\,h}{35}, -\frac{d}{7} + \frac{6\,e}{35} + \frac{12\,f}{35} + \frac{13\,g}{35} + \frac{9\,h}{35}, \frac{3\,d}{35} - \frac{e}{7} - \frac{3\,f}{35} + \frac{9\,g}{35} + \frac{31\,h}{35}\right\}$$
$$\left\{-\frac{27\,a}{35} + \frac{13\,b}{70} + \frac{4\,c}{7} + \frac{27\,d}{70} - \frac{13\,e}{35}, -\frac{17\,a}{35} + \frac{3\,b}{70} + \frac{2\,c}{7} + \frac{17\,d}{70} - \frac{3\,e}{35}, -\frac{a}{5} - \frac{b}{10} + \frac{d}{10} + \frac{e}{5}, -\frac{b}{5} - \frac{c}{10} + \frac{e}{10} + \frac{f}{5},\right.$$
$$\left.-\frac{c}{5} - \frac{d}{10} + \frac{f}{10} + \frac{g}{5}, -\frac{d}{5} - \frac{e}{10} + \frac{g}{10} + \frac{h}{5}, \frac{3\,d}{35} - \frac{17\,e}{70} - \frac{2\,f}{7} - \frac{3\,g}{70} + \frac{17\,h}{35}, \frac{13\,d}{35} - \frac{27\,e}{70} - \frac{4\,f}{7} - \frac{13\,g}{70} + \frac{27\,h}{35}\right\}$$
$$\left\{\frac{2\,a}{7} - \frac{b}{7} - \frac{2\,c}{7} - \frac{d}{7} + \frac{2\,e}{7}, \frac{2\,a}{7} - \frac{b}{7} - \frac{2\,c}{7} - \frac{d}{7} + \frac{2\,e}{7}, \frac{2\,a}{7} - \frac{b}{7} - \frac{2\,c}{7} - \frac{d}{7} + \frac{2\,e}{7}, \frac{2\,b}{7} - \frac{c}{7} - \frac{2\,d}{7} - \frac{e}{7} + \frac{2\,f}{7},\right.$$
$$\left.\frac{2\,c}{7} - \frac{d}{7} - \frac{2\,e}{7} - \frac{f}{7} + \frac{2\,g}{7}, \frac{2\,d}{7} - \frac{e}{7} - \frac{2\,f}{7} - \frac{g}{7} + \frac{2\,h}{7}, \frac{2\,d}{7} - \frac{e}{7} - \frac{2\,f}{7} - \frac{g}{7} + \frac{2\,h}{7}, \frac{2\,d}{7} - \frac{e}{7} - \frac{2\,f}{7} - \frac{g}{7} + \frac{2\,h}{7}\right\}$$

In[641]:=
```
(kers = sSavGolElasticKernels1D[2, {1, 1}, "Derivs" → Range[0, 2]]) // Normal //
  Column
sSavGolBufferConvolve1D[kers, CharacterRange["a", "h"], "Indexed" → True] //
  Normal // Column
```

Out[641]=
$$0 \to \langle| L \to \{\{0, 0, 1\}\}, C \to \{\{0, 1, 0\}\}, R \to \{\{1, 0, 0\}\} |\rangle$$
$$1 \to \left\langle\left| L \to \left\{\left\{-\frac{1}{2}, 2, -\frac{3}{2}\right\}\right\}, C \to \left\{\left\{\frac{1}{2}, 0, -\frac{1}{2}\right\}\right\}, R \to \left\{\left\{\frac{3}{2}, -2, \frac{1}{2}\right\}\right\} \right|\right\rangle$$
$$2 \to \langle| L \to \{\{1, -2, 1\}\}, C \to \{\{1, -2, 1\}\}, R \to \{\{1, -2, 1\}\} |\rangle$$

Out[642]=
$$0 \to \{a, b, c, d, e, f, g, h\}$$
$$1 \to \left\{-\frac{3\,a}{2} + 2\,b - \frac{c}{2}, -\frac{a}{2} + \frac{c}{2}, -\frac{b}{2} + \frac{d}{2}, -\frac{c}{2} + \frac{e}{2}, -\frac{d}{2} + \frac{f}{2}, -\frac{e}{2} + \frac{g}{2}, -\frac{f}{2} + \frac{h}{2}, \frac{f}{2} - 2\,g + \frac{3\,h}{2}\right\}$$
$$2 \to \{a - 2\,b + c, a - 2\,b + c, b - 2\,c + d,$$
$$c - 2\,d + e, d - 2\,e + f, e - 2\,f + g, f - 2\,g + h, f - 2\,g + h\}$$

Noisy data smoothing and derivatives estimation

This allows restoring the options of the built-in symbol ListPlot to the original state:

In[643]:=
```
optsListPlot = Options[ListPlot];
```

First we define synthetic data in the form of the ChebyshevT polynomial $T_4(x)$ of the 1st kind polluted with additive Gaussian noise, and the Savitzky–Golay 1D filter parameters.

In[644]:=
```
SetOptions[ListPlot, Axes → False, Frame → True, PlotRange → Full,
  GridLines → Automatic, Filling → None, Joined → True, Mesh → All,
  MeshStyle → Directive[PointSize[0.01], Black], AspectRatio → 1 / GoldenRatio,
  ImageSize → 320];
colors = {Directive[Thickness[0.025], LightGray], Black, Red, Green,
   Blue, Brown};
legends = {"original", "smoothed", "1st deriv", "2nd deriv", "3rd deriv",
   "4th deriv"};
legends2 = {"original", "smoothed", "curvature", "curv. extrema"};
par = Sequence[3, {4, 4}];
hw = 25; std = 0.001; polord = 4; noise = 1; Clear[x];
Δ = {1, hw, hw^2, hw^3, hw^4}; order = First@List@par;
data = With[{hw = hw, std = std},
   Table[ChebyshevT[polord, k/hw], {k, -hw, hw, 1}] +
    noise RandomVariate[NormalDistribution[0, std], 2 hw + 1]];
Table[D[ChebyshevT[polord, x], {x, k}], {k, 0, order}] //
 TableForm[#, TableHeadings → {Range[0, order], None}] &
```

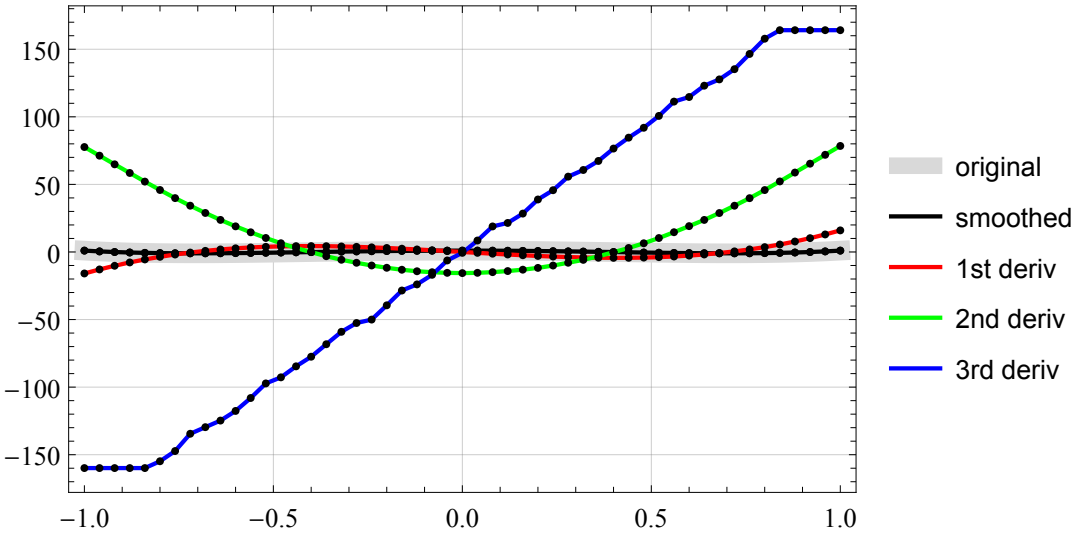
Out[652]//TableForm=

0	$1 - 8\,x^2 + 8\,x^4$
1	$-16\,x + 32\,x^3$
2	$-16 + 96\,x^2$
3	$192\,x$

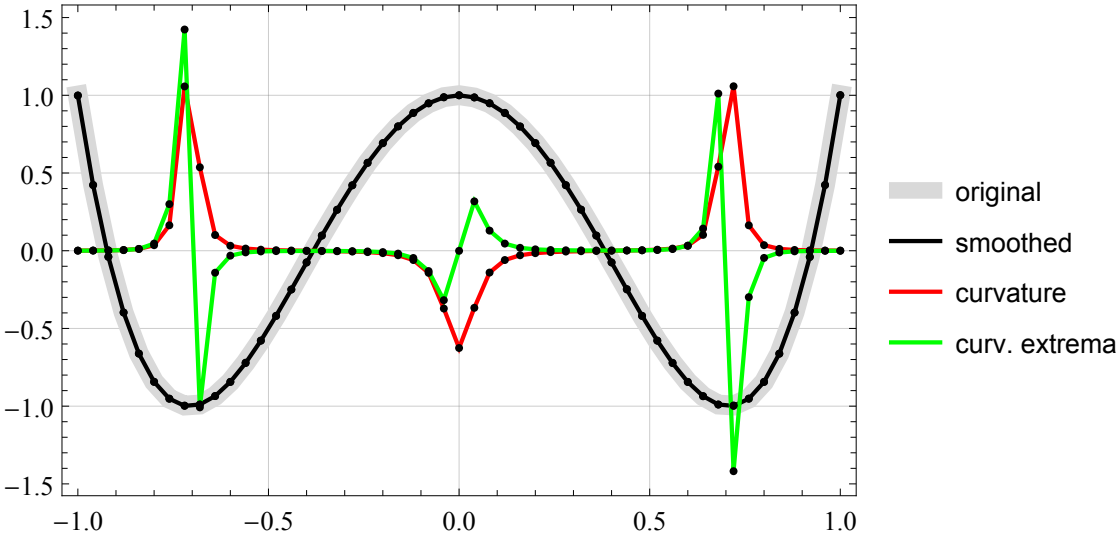
The we calculate the bunch of “elastic” kernels and perform the Savitzky–Golay 1D filtering, including derivatives of the data:

In[653]:=
```
kers = sSavGolElasticKernels1D[par, "Derivs" → Range[0, order]];
out = If[order > 1,
    sSavGolBufferConvolve1D[kers, data, AspectRatio → 1],
    sSavGolBufferConvolve1D[kers, data]];
If[order > 0,
 ListPlot[{data, Sequence @@ (Take[Δ, order + 1] Take[out, order + 1])},
  DataRange → {-1, 1}, PlotStyle → Take[colors, order + 2],
  PlotLegends → Take[legends, order + 2]],
 ListPlot[{data, out}, DataRange → {-1, 1}, PlotStyle → Take[colors, order + 2],
  PlotLegends → Take[legends, order + 2]]]
If[order > 2,
 ListPlot[{data, First@out, Last@out, out[[-2]]}, DataRange → {-1, 1},
  PlotStyle → Take[colors, order + 2], PlotLegends → legends2]]
```

Out[655]=



Out[656]=



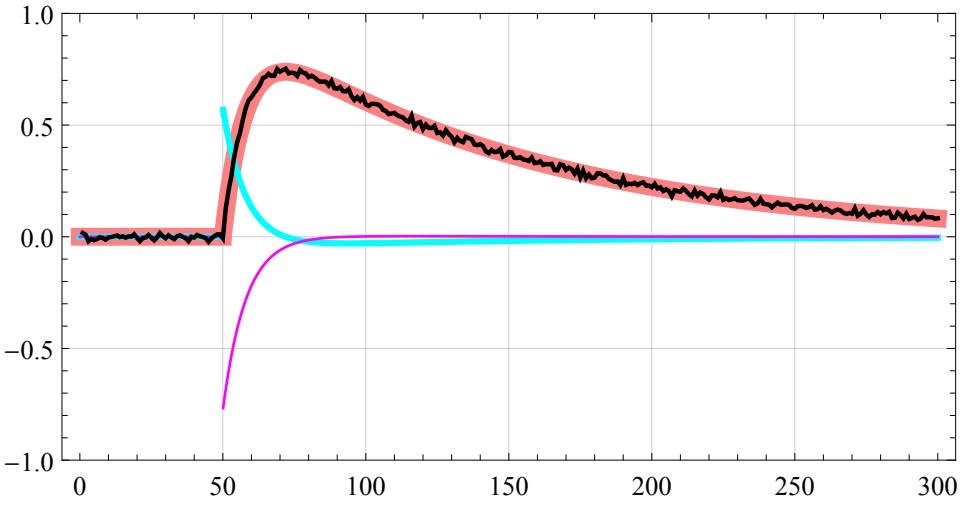
Applications

Signal filtering: Example A

Using sSavGolBufferConvolve1D[*kers*, *data*, *opts*] (with kernels k e r s returned by sSavGolElasticKernels1D) to apply the Savitzky–Golay filtering to a data vector. We define an "analytically" given function in the form of a piecewise function as depicted in the upper panel in red color, including its first (scaled up 5×) and second (scaled up 50×) derivatives. Consequently, we sample the function and form the data by adding additive Gaussian noise, as shown in the bottom panel. (The first line of the code allows restoring the options of the built-in symbols Plot and ListLinePlot to the original state).

In[657]:=
```
optsPlot = Options[Plot];
optsListLinePlot = Options[ListLinePlot];
SetOptions[Plot, Axes → False, Frame → True, ImageSize → Medium,
  GridLines → Automatic, AspectRatio → 1 / 2];
SetOptions[ListLinePlot, Axes → False, Frame → True, ImageSize → Medium,
  GridLines → Automatic, AspectRatio → 1 / 2];
epsp[t_] := Piecewise[{{-E^(-(t - 50.) / 8.) + E^(-(t - 50.) / 100.), t > 50.}},
  0.]
epsp1[t_] := D[epsp[t], t]; epsp2[t_] := D[epsp[t], {t, 2}]
epspPlot = Plot[{epsp[t], Evaluate[5 epsp1[t]], Evaluate[50 epsp2[t]]},
   {t, 0, 300}, PlotRange → {-1, 1},
   PlotStyle → {Directive[Pink, Thickness[0.02]],
     Directive[Cyan, Thickness[0.007]],
     Directive[Magenta, Thickness[0.003]]}];
measurement = Table[epsp[t] + RandomVariate[NormalDistribution[0, 0.01]],
   {t, 1, 300}];
measurementPlot = ListLinePlot[measurement, PlotRange → {-1, 1},
   PlotStyle → Directive[Black, Thickness[0.005]]];
Show[epspPlot, measurementPlot]
```

Out[665]=



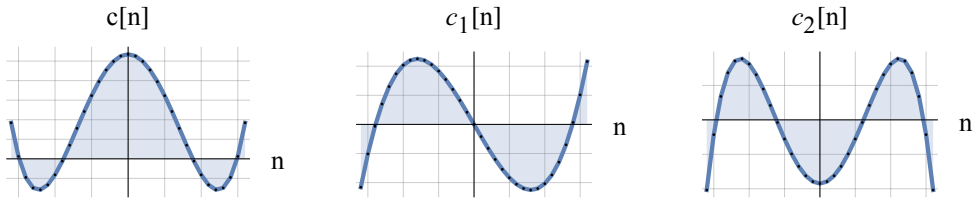
Symmetric kernel

Here we define a 33-tap, $n_l = n_r = 16$, symmetric Savitzky–Golay smoothing kernel of 4th order,

including kernels for first and second derivatives:

In[666]:=
```
ord = 4; nl = 16; nr = 16;
{c, c1, c2} = sSavGolKernel1D[ord, {nl, nr}, "Derivs" → Range[0, 2]];
Total /@ {c, c1, c2};
{c, c1, c2} = N[{c, c1, c2}];
SetOptions[ListPlot, Axes → True, Frame → False];
GraphicsRow[
 {ListPlot[c, AxesLabel → {"n", "c[n]"}, Ticks → None, DataRange → {-nl, nr},
   Filling → Axis, PlotStyle → Directive[PointSize[Medium]]],
  ListPlot[c1, AxesLabel → {"n", "c1[n]"}, Ticks → None, DataRange → {-nl, nr},
   Filling → Axis, PlotStyle → Directive[PointSize[Medium]]],
  ListPlot[c2, AxesLabel → {"n", "c2[n]"}, Ticks → None, DataRange → {-nl, nr},
   Filling → Axis, PlotStyle → Directive[PointSize[Medium]]]},
 ImageSize → 380]
```

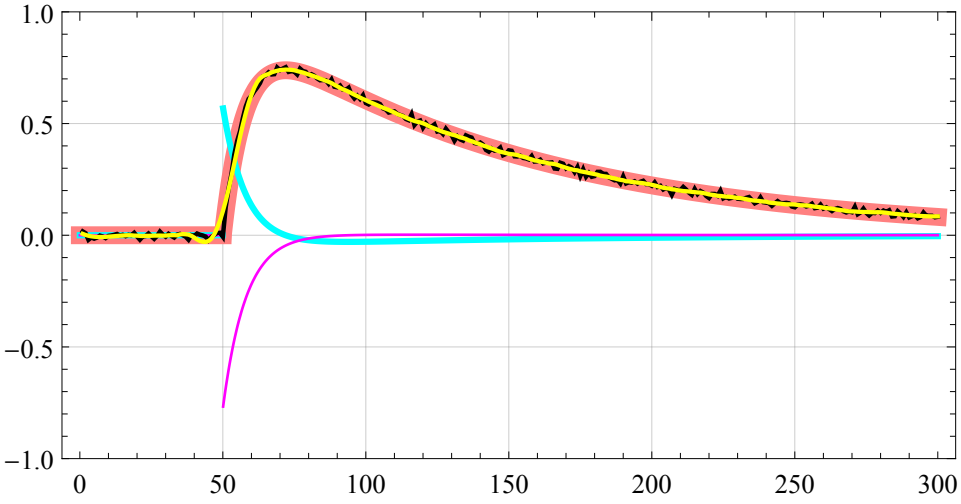
Out[669]=



Now, we can observe the effect of the smoothing Savitzky–Golay filtering (yellow vs. red curves),

In[670]:=
```
SetOptions[ListPlot, Axes → False, Frame → True];
filterOutput = sSavGolBufferConvolve1D[
   sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0}], measurement];
filterOutputPlot = ListLinePlot[filterOutput, PlotRange → {-1, 1},
   PlotStyle → Directive[Yellow]];
Show[{epspPlot, measurementPlot, filterOutputPlot}]
```

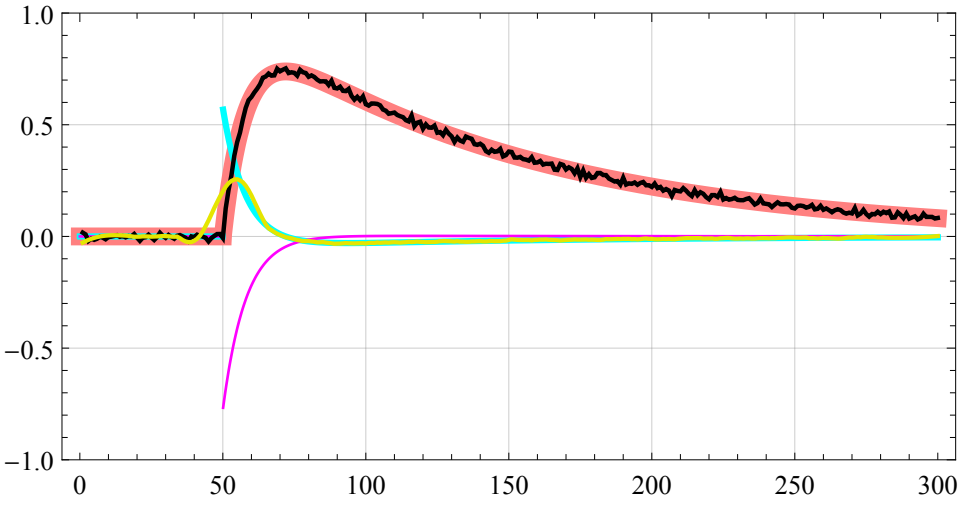
Out[673]=



the result of the first derivative computation (yellow vs. cyan curves),

In[674]:=
```
filterOutput1 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {1}], measurement];
filterOutput1Plot = ListLinePlot[5 filterOutput1, PlotRange → {-1, 1},
    PlotStyle → Directive[Darker[Yellow, 0.1]]];
Show[{epspPlot, measurementPlot, filterOutput1Plot}]
```

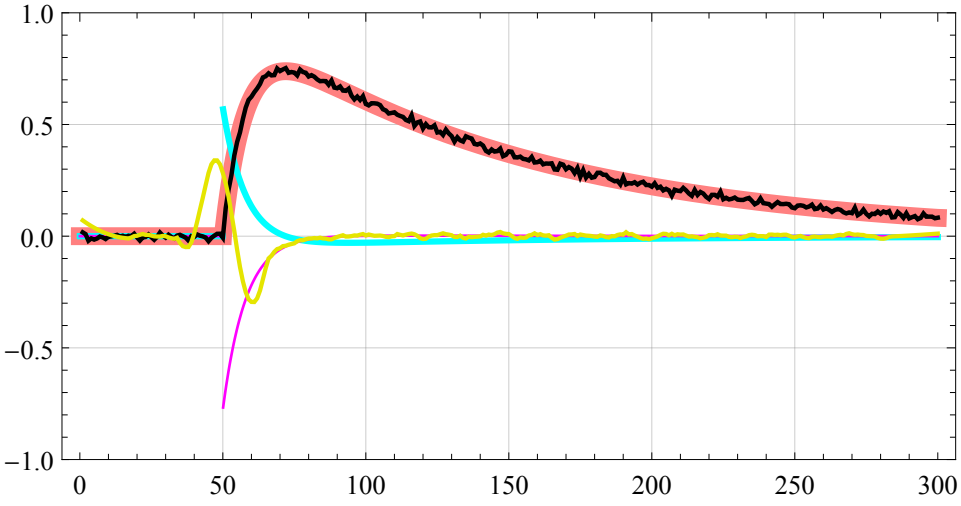
Out[676]=



the result of the second derivative computation (yellow vs. violet curves),

In[677]:=
```
filterOutput2 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {2}], measurement];
filterOutput2Plot = ListLinePlot[50 filterOutput2, PlotRange → {-1, 1},
    PlotStyle → Directive[Darker[Yellow, 0.1]]];
Show[{epspPlot, measurementPlot, filterOutput2Plot}]
```

Out[679]=

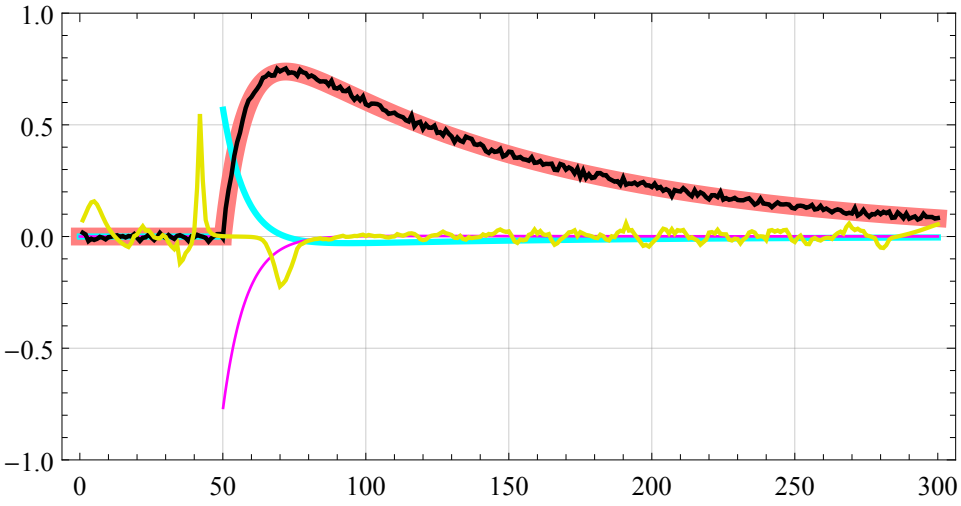


the result of the curvature computation (yellow curve),

In[680]:=
```
Last@
  (filterOutputAC = sSavGolBufferConvolve1D[
     sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0, 1, 2}],
     measurement, AspectRatio → True]);
filterOutputACPlot = ListLinePlot[Last@filterOutputAC, PlotRange → {-1, 1},
   PlotStyle → Directive[Darker[Yellow, 0.1]]];
Show[{epspPlot, measurementPlot, filterOutputACPlot}]
```

sSavGolBufferConvolve1D: NOTE: default aspect ratio $\frac{1}{\text{GoldenRatio}}$ used.

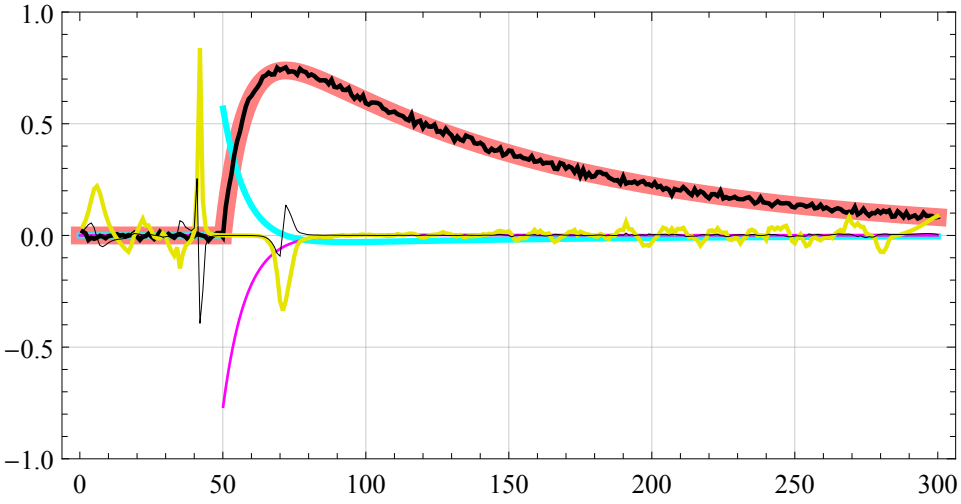
Out[682]=



and finally, the result of the curvature and the curvature first derivative computation (yellow and black curves, respectively).

In[683]:=
```
filterOutputAC = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0, 1, 2, 3}],
    measurement, AspectRatio → 1];
filterOutputACPlot = ListLinePlot[filterOutputAC[[-1]], PlotRange → {-1, 1},
    PlotStyle → Directive[Darker[Yellow, 0.1]]];
filterOutputZXPlot = ListLinePlot[filterOutputAC[[-2]], PlotRange → {-1, 1},
    PlotStyle → Directive[Thin, Black]];
Show[{epspPlot, measurementPlot, filterOutputACPlot, filterOutputZXPlot}]
```

Out[686]=

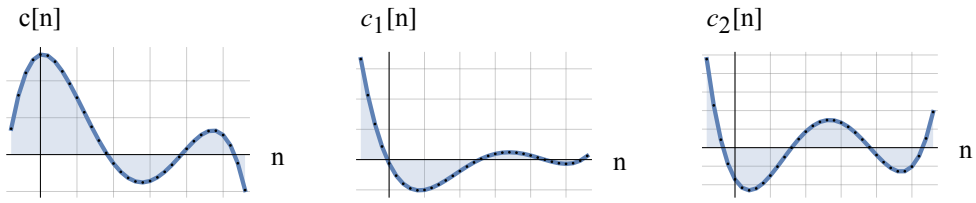


Asymmetric kernel

Here we define a 33-tap, $n_l = 4$, $n_r = 28$, asymmetric Savitzky–Golay smoothing kernel of 4th order, including kernels for first and second derivatives.

In[687]:=
```
ord = 4;
nl = 4;
nr = 28;
{c, c1, c2} = sSavGolKernel1D[ord, {nl, nr}, "Derivs" → Range[0, 2]];
Total /@ {c, c1, c2};
{c, c1, c2} = N[{c, c1, c2}];
SetOptions[ListPlot, Axes → True, Frame → False];
GraphicsRow[
 {ListPlot[c, AxesLabel → {"n", "c[n]"}, DataRange → {-nl, nr},
   Ticks → None, Filling → Axis, PlotStyle → Directive[PointSize[Medium]]],
  ListPlot[c1, AxesLabel → {"n", "c1[n]"}, DataRange → {-nl, nr},
   Ticks → None, Filling → Axis, PlotStyle → Directive[PointSize[Medium]]],
  ListPlot[c2, AxesLabel → {"n", "c2[n]"}, DataRange → {-nl, nr},
   Ticks → None, Filling → Axis, PlotStyle → Directive[PointSize[Medium]]]},
 ImageSize → 380]
```

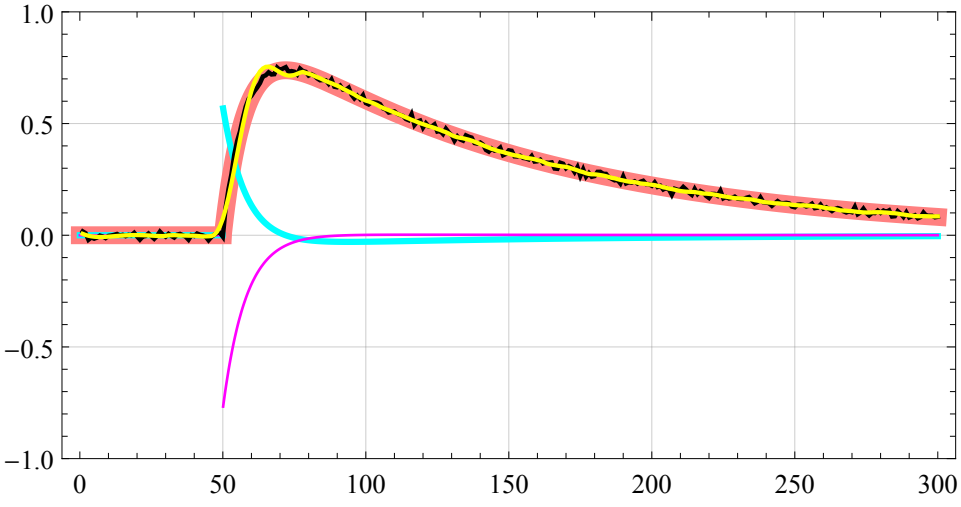
Out[689]=



Now, we can observe the effect of the smoothing Savitzky–Golay filtering (yellow vs. red curves),

In[690]:=
```
filterOutput = sSavGolBufferConvolve1D[
   sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0}], measurement];
filterOutputPlot = ListLinePlot[filterOutput, PlotRange → {-1, 1},
   PlotStyle → Directive[Yellow]];
Show[{epspPlot, measurementPlot, filterOutputPlot}]
```

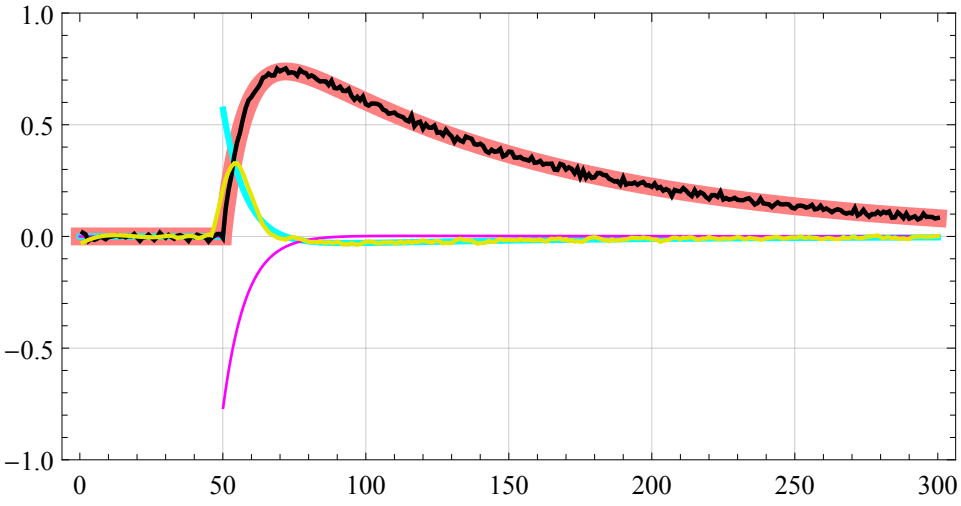
Out[692]=



the result of the first derivative computation (yellow vs. cyan curves),

In[693]:=
```
filterOutput1 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {1}], measurement];
filterOutput1Plot = ListLinePlot[5 filterOutput1, PlotRange → {-1, 1},
    PlotStyle → Directive[Darker[Yellow, 0.1]]];
Show[{epspPlot, measurementPlot, filterOutput1Plot}]
```

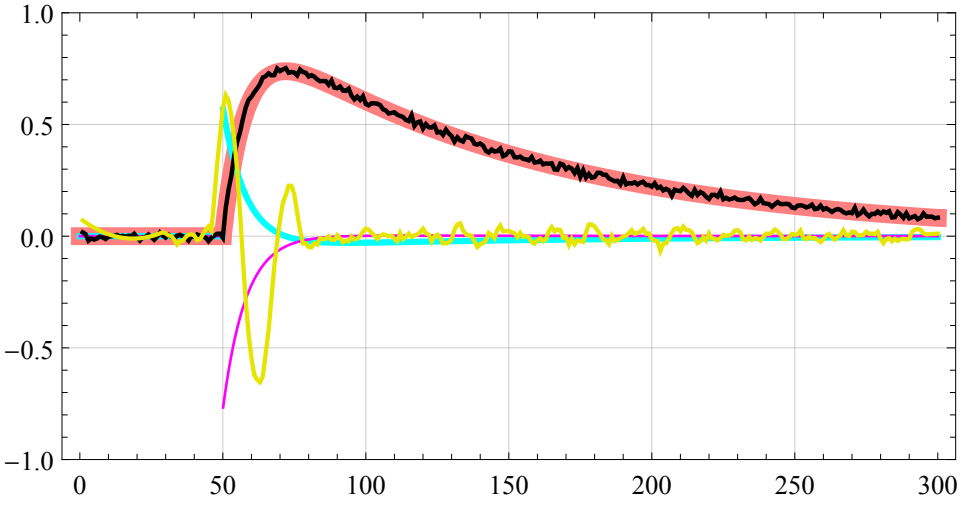
Out[695]=



the result of the second derivative computation (yellow vs. violet curves),

In[696]:=
```
filterOutput2 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {2}], measurement];
filterOutput2Plot = ListLinePlot[50 filterOutput2, PlotRange → {-1, 1},
    PlotStyle → Directive[Darker[Yellow, 0.1]]];
Show[{epspPlot, measurementPlot, filterOutput2Plot}]
```

Out[698]=

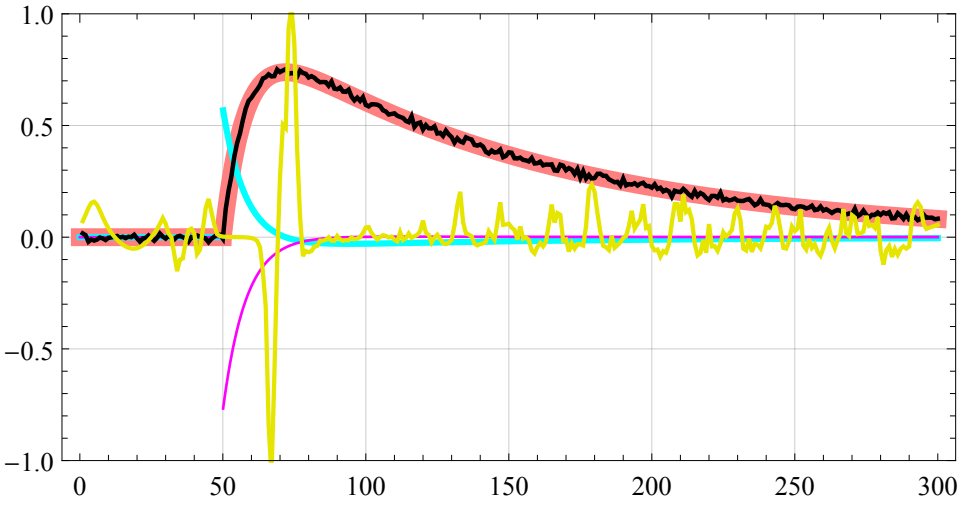


the result of the curvature computation (yellow curve),

In[699]:=
```
filterOutputAC = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0, 1, 2}],
    measurement, AspectRatio → True];
filterOutputACPlot = ListLinePlot[Last@filterOutputAC, PlotRange → {-1, 1},
    PlotStyle → Directive[Darker[Yellow, 0.1]]];
Show[{epspPlot, measurementPlot, filterOutputACPlot}]
```

sSavGolBufferConvolve1D: NOTE: default aspect ratio $\frac{1}{\text{GoldenRatio}}$ used.

Out[701]=

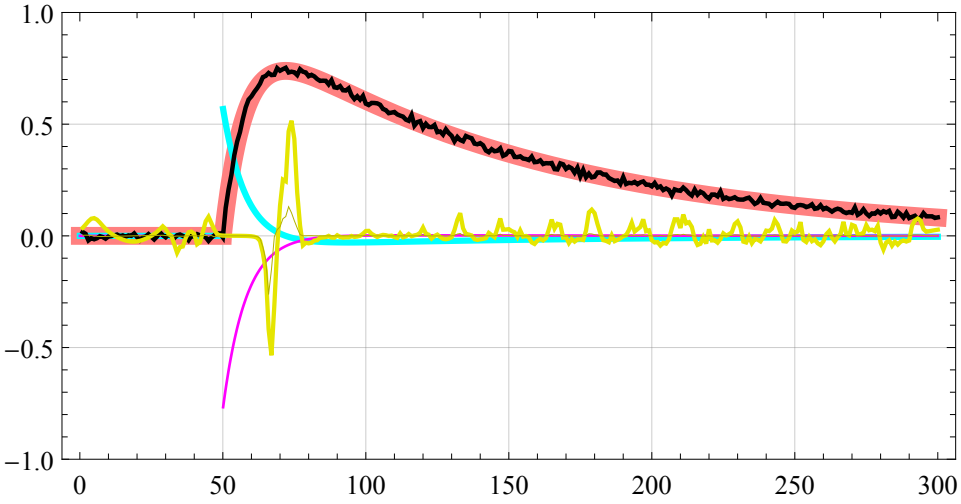


and finally, the result of the curvature and the curvature first derivative computation (yellow and black curves, respectively) for the default aspect ratio,

In[702]:=
```
filterOutputAC = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0, 1, 2, 3}],
    measurement, AspectRatio → True];
filterOutputACPlot = ListLinePlot[0.5 filterOutputAC[[-1]],
    PlotRange → {-1, 1}, PlotStyle → Directive[Darker[Yellow, 0.1]]];
filterOutputZXPlot = ListLinePlot[0.1 filterOutputAC[[-2]],
    PlotRange → {-1, 1}, PlotStyle → Directive[Thin, Darker[Yellow, 0.3]]];
Show[{epspPlot, measurementPlot, filterOutputACPlot, filterOutputZXPlot}]
```

sSavGolBufferConvolve1D: NOTE: default aspect ratio $\frac{1}{\text{GoldenRatio}}$ used.

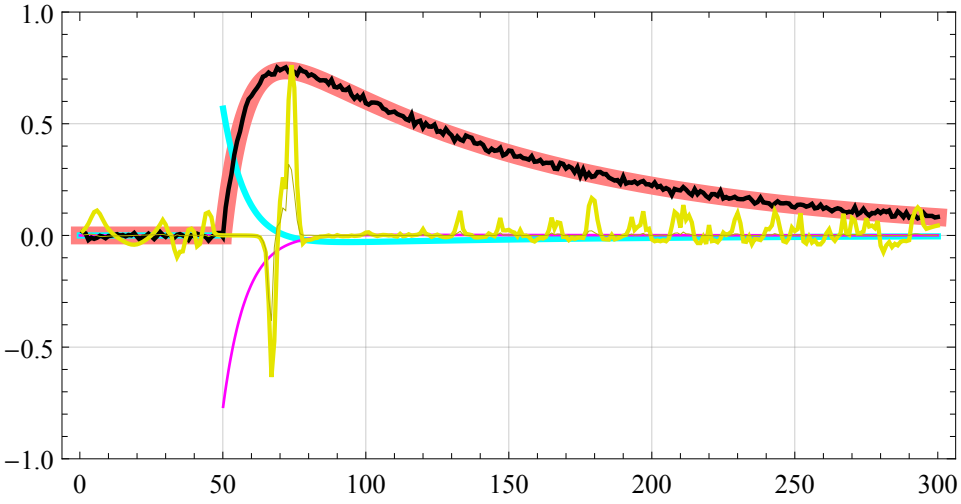
Out[705]=



and for aspect ratio set to 1.

In[706]:=
```
filterOutputAC = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0, 1, 2, 3}],
    measurement, AspectRatio → 1];
filterOutputACPlot = ListLinePlot[0.5 filterOutputAC[[-1]],
    PlotRange → {-1, 1}, PlotStyle → Directive[Darker[Yellow, 0.1]]];
filterOutputZXPlot = ListLinePlot[0.1 filterOutputAC[[-2]],
    PlotRange → {-1, 1}, PlotStyle → Directive[Thin, Darker[Yellow, 0.3]]];
Show[{epspPlot, measurementPlot, filterOutputACPlot, filterOutputZXPlot}]
```

Out[709]=

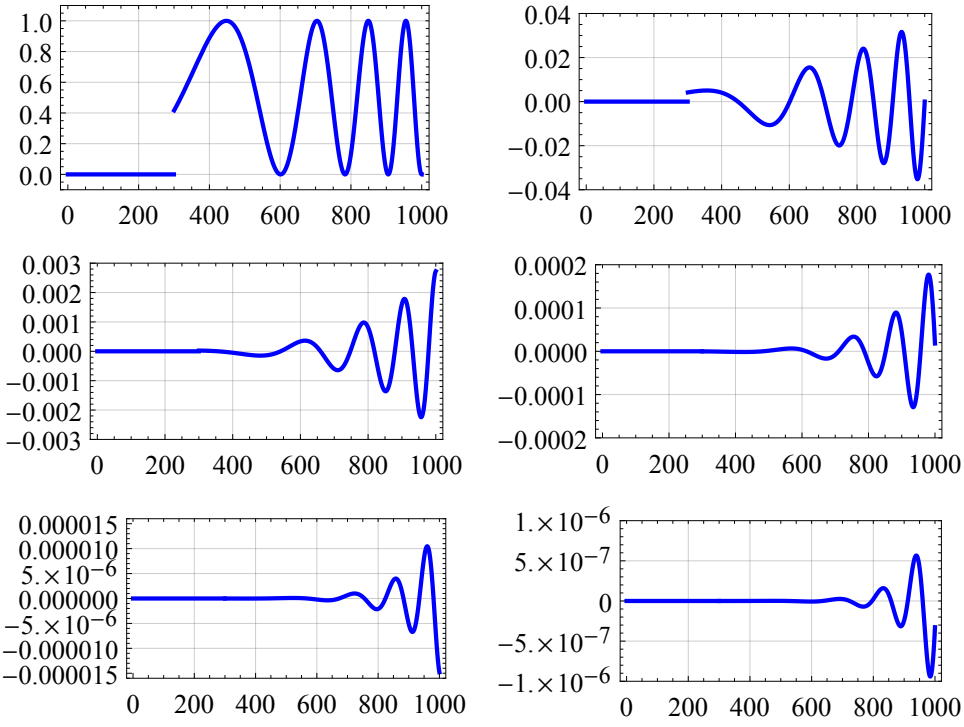


Signal filtering: Example B

We define an “analytically” given piecewise function as depicted in the panels, including its derivatives up to 5th order. Consequently, we sample the function and form the data by adding additive Gaussian noise, as shown in the bottom panel.

In[710]:=
```
func[t_] := Piecewise[{{Sin[((4 π)^(1/4) - 0.5) t/1000. + 0.5)^4]^2, t ≥ 300}}, 0.0]
func1[t_] := D[func[t], t];
func2[t_] := D[func[t], {t, 2}];
func3[t_] := D[func[t], {t, 3}];
func4[t_] := D[func[t], {t, 4}]; func5[t_] := D[func[t], {t, 5}]
GraphicsGrid[
 {{funcPlot = Plot[func[t], {t, 0, 1000}, PlotRange → {-0.1, 1.1},
     PlotStyle → {Blue}],
   func1Plot = Plot[Evaluate@func1[t], {t, 0, 1000},
     PlotRange → {-0.04, 0.04}, PlotStyle → {Blue}]},
  {func2Plot = Plot[Evaluate@func2[t], {t, 0, 1000},
     PlotRange → {-0.003, 0.003}, PlotStyle → {Blue}],
   func3Plot = Plot[Evaluate@func3[t], {t, 0, 1000},
     PlotRange → {-0.0002, 0.0002}, PlotStyle → {Blue}]},
  {func4Plot = Plot[Evaluate@func4[t], {t, 0, 1000},
     PlotRange → {-0.000016, 0.000016}, PlotStyle → {Blue}],
   func5Plot = Plot[Evaluate@func5[t], {t, 0, 1000},
     PlotRange → {-0.000001, 0.000001}, PlotStyle → {Blue}]}},
 ImageSize → 360]
```

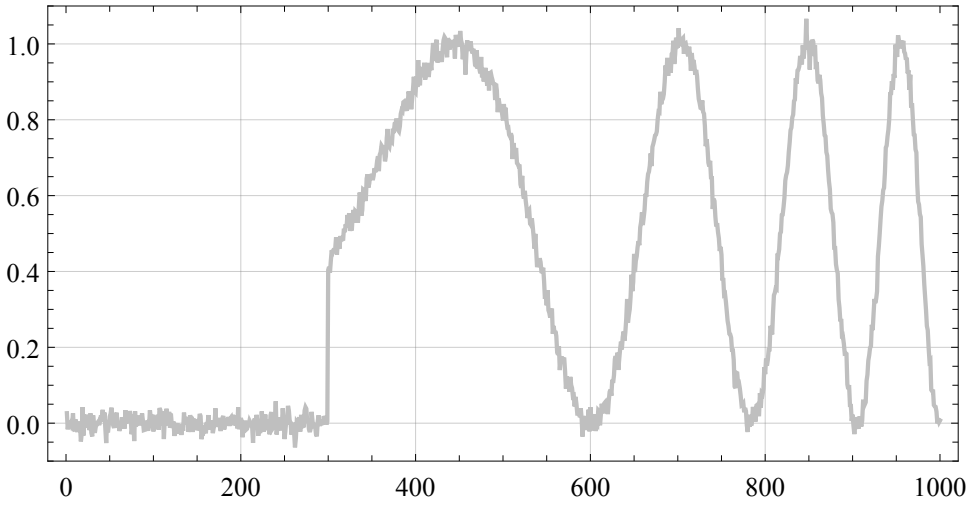
Out[713]=



Then we admix some additive Gaussian noise:

In[714]:=
```
noisydata = Table[func[t] + RandomVariate[NormalDistribution[0, 0.02]],
    {t, 1, 1000}];
noisydataPlot = ListLinePlot[noisydata, PlotRange → {-0.1, 1.1},
  PlotStyle → {GrayLevel[0.75]}]
```

Out[715]=

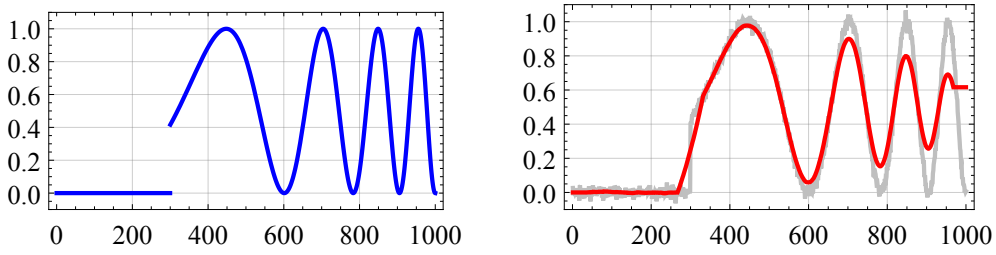


In the following subsections we can observe the effect of the Savitzky–Golay smoothing and derivative computations from the noised data, and their comparison with the original functions.

Order 0, symmetric kernel

In[716]:=
```
ord = 0; nl = 32; nr = 32;
smoothed = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0}], noisydata];
smoothedPlot = ListLinePlot[smoothed, PlotRange → {-0.1, 1.1},
    PlotStyle → {Red}];
sgFilter = GraphicsRow[{funcPlot, Show[{noisydataPlot, smoothedPlot}]},
  ImageSize → 380]
```

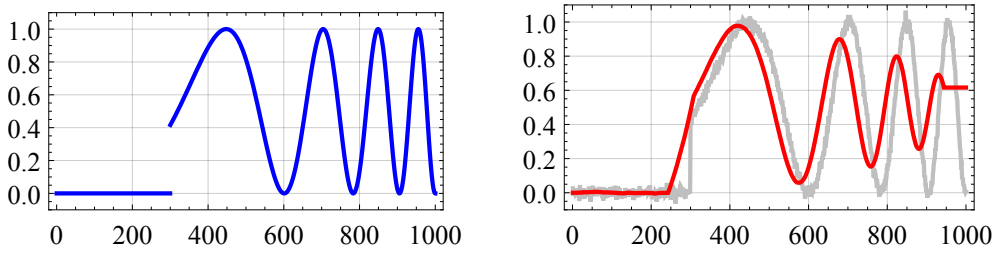
Out[719]=



Order 0, asymmetric kernel

In[720]:=
```
ord = 0; nl = 56; nr = 8;
smoothed = sSavGolBufferConvolve1D[
   sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0}], noisydata];
smoothedPlot = ListLinePlot[smoothed, PlotRange → {-0.1, 1.1},
   PlotStyle → {Red}];
sgFilter = GraphicsRow[{funcPlot, Show[{noisydataPlot, smoothedPlot}]},
  ImageSize → 380]
```

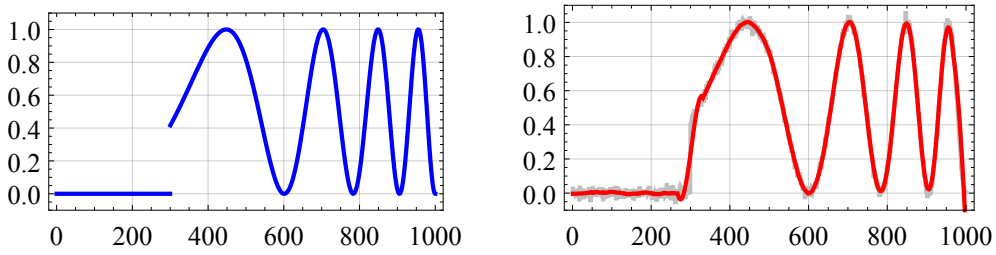
Out[723]=



Order 2, symmetric kernel

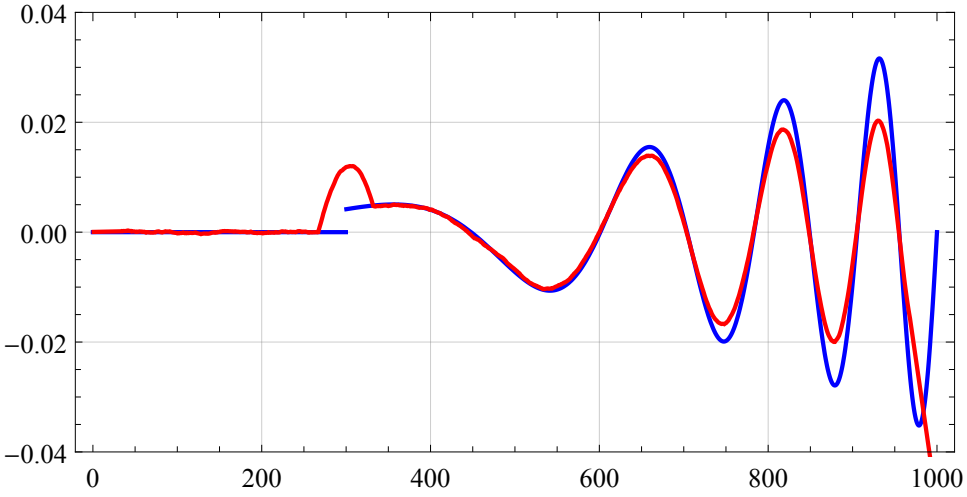
In[724]:=
```
ord = 2; nl = 32; nr = 32;
smoothed = sSavGolBufferConvolve1D[
   sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0}], noisydata];
smoothedPlot = ListLinePlot[smoothed, PlotRange → {-0.1, 1.1},
   PlotStyle → {Red}];
sgFilter = GraphicsRow[{funcPlot, Show[{noisydataPlot, smoothedPlot}]},
  ImageSize → 380]
```

Out[727]=



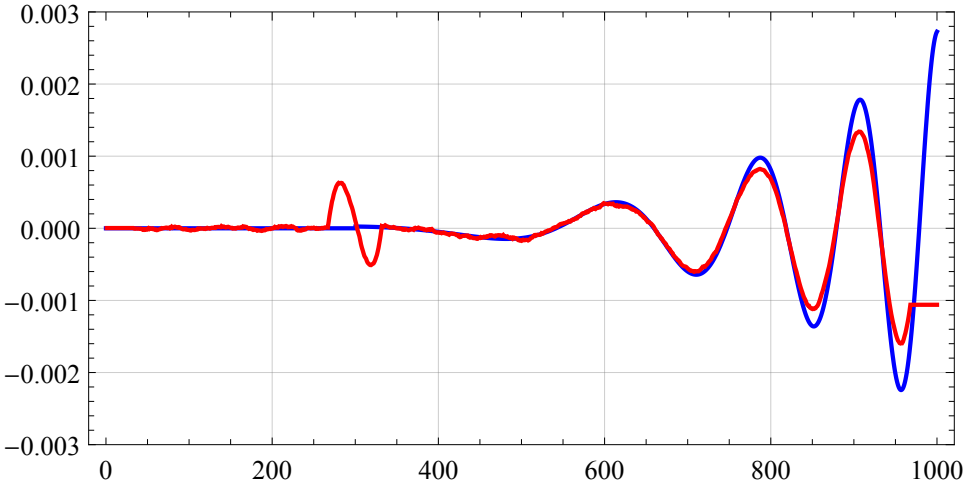
In[728]:=
```
drv1 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {1}], noisydata];
drv1Plot = ListLinePlot[drv1, PlotStyle → {Red}];
Show[{func1Plot, drv1Plot}]
```

Out[729]=



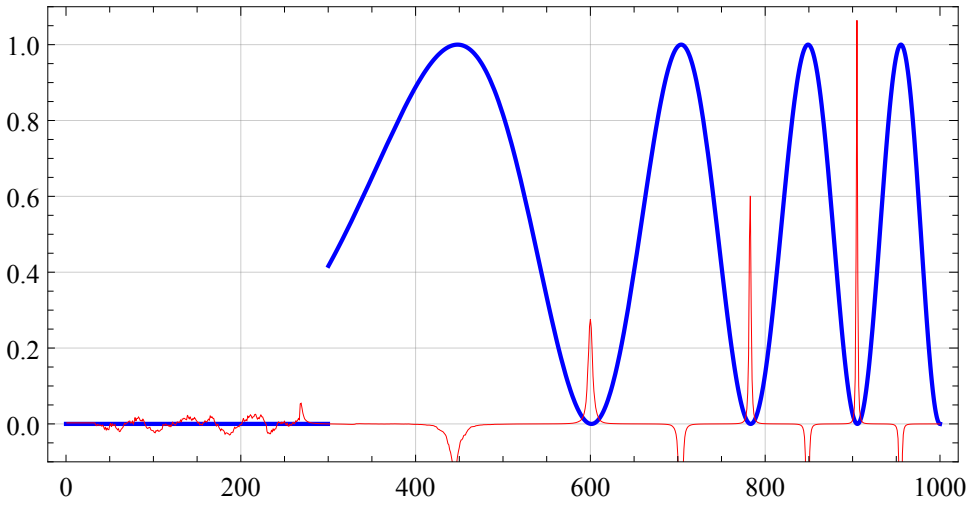
In[730]:=
```
drv2 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {2}], noisydata];
drv2Plot = ListLinePlot[drv2, PlotStyle → {Red}];
Show[{func2Plot, drv2Plot}]
```

Out[731]=



In[732]:=
```
Last@
  (curv = sSavGolBufferConvolve1D[
     sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0, 1, 2}],
     noisydata, AspectRatio → 1]);
curvPlot = ListLinePlot[Last@curv, PlotStyle → Directive[Thin, Red],
   PlotRange → All];
Show[{funcPlot, curvPlot}]
```

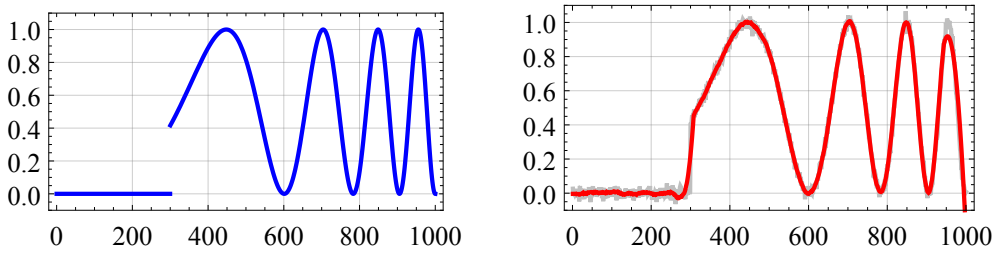
Out[734]=



Order 2, asymmetric kernel

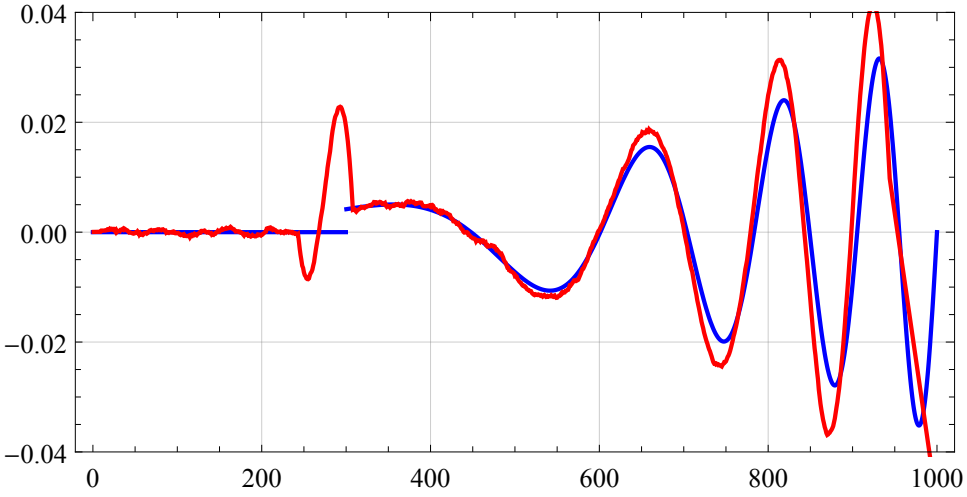
In[735]:=
```
ord = 2; nl = 56; nr = 8;
smoothed = sSavGolBufferConvolve1D[
   sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0}], noisydata];
smoothedPlot = ListLinePlot[smoothed, PlotRange → {-0.1, 1.1},
   PlotStyle → {Red}];
sgFilter = GraphicsRow[{funcPlot, Show[{noisydataPlot, smoothedPlot}]},
  ImageSize → 380]
```

Out[738]=



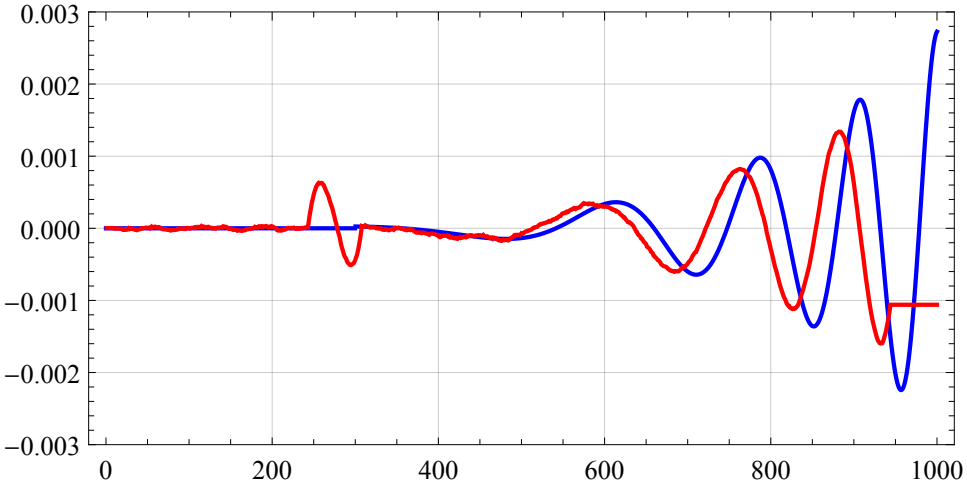
In[739]:=
```
drv1 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {1}], noisydata];
drv1Plot = ListLinePlot[drv1, PlotStyle → {Red}];
Show[{func1Plot, drv1Plot}]
```

Out[741]=



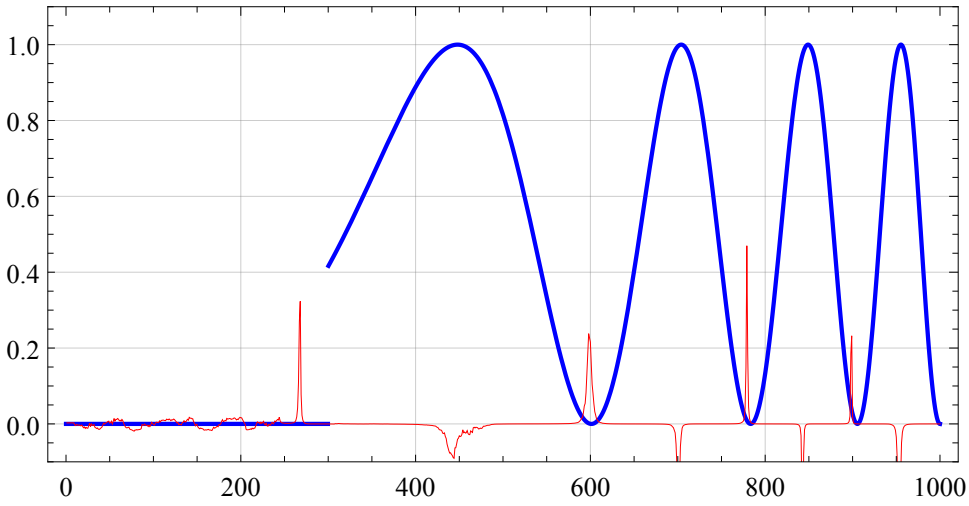
In[742]:=
```
drv2 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {2}], noisydata];
drv2Plot = ListLinePlot[drv2, PlotStyle → {Red}];
Show[{func2Plot, drv2Plot}]
```

Out[744]=



In[745]:=
```
Last@
  (curv = sSavGolBufferConvolve1D[
     sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0, 1, 2}],
     noisydata, AspectRatio → 1]);
curvPlot = ListLinePlot[Last@curv, PlotStyle → Directive[Thin, Red],
   PlotRange → Full];
Show[{funcPlot, curvPlot}]
```

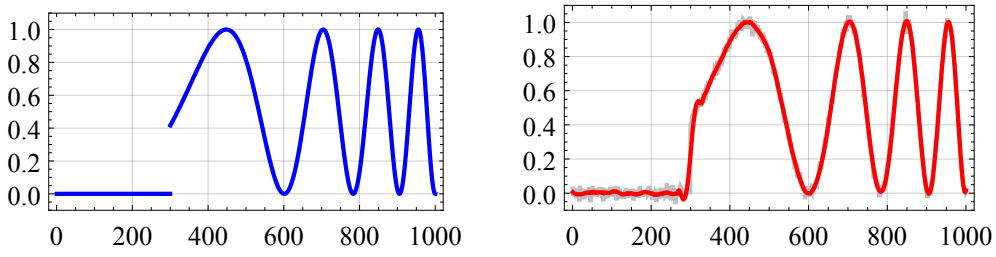
Out[747]=



Order 4, symmetric kernel

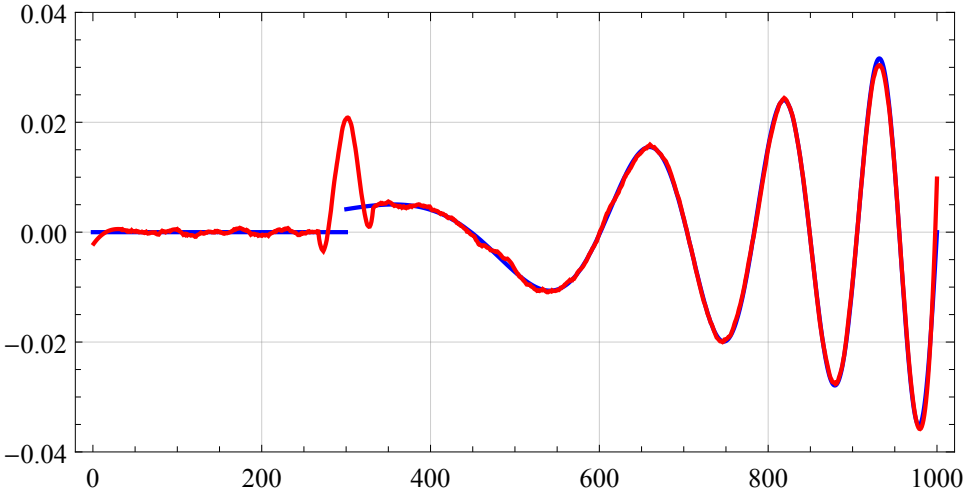
In[748]:=
```
ord = 4; nl = 32; nr = 32;
smoothed = sSavGolBufferConvolve1D[
   sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0}], noisydata];
smoothedPlot = ListLinePlot[smoothed, PlotRange → {-0.1, 1.1},
   PlotStyle → {Red}];
sgFilter = GraphicsRow[{funcPlot, Show[{noisydataPlot, smoothedPlot}]},
  ImageSize → 380]
```

Out[751]=



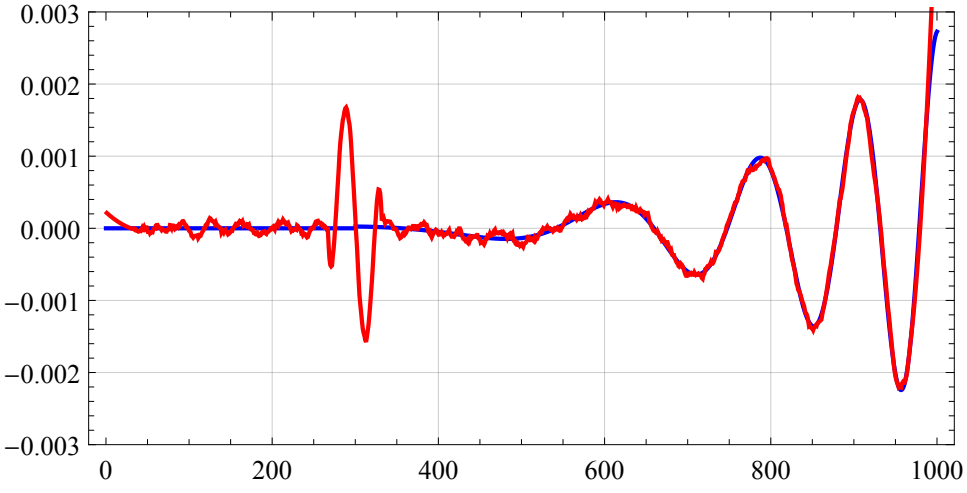
In[752]:=
```
drv1 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {1}], noisydata];
drv1Plot = ListLinePlot[drv1, PlotStyle → {Red}, PlotRange → Full];
Show[{func1Plot, drv1Plot}]
```

Out[754]=



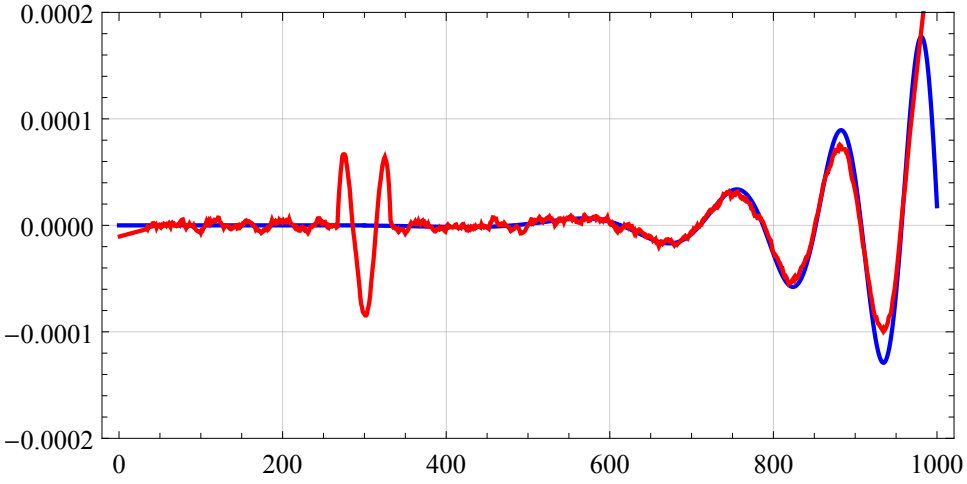
In[755]:=
```
drv2 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {2}], noisydata];
drv2Plot = ListLinePlot[drv2, PlotStyle → {Red}, PlotRange → Full];
Show[{func2Plot, drv2Plot}]
```

Out[757]=



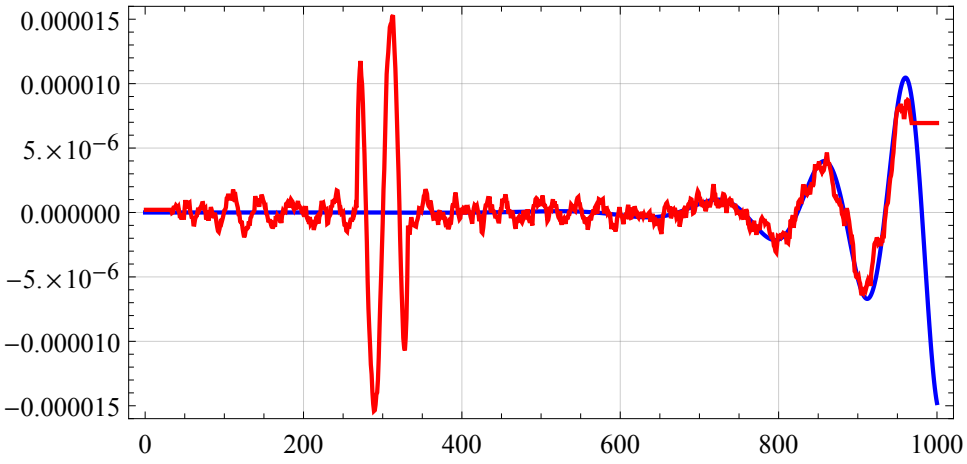
In[758]:=
```
drv3 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {3}], noisydata];
drv3Plot = ListLinePlot[drv3, PlotStyle → {Red}, PlotRange → Full];
Show[{func3Plot, drv3Plot}]
```

Out[760]=



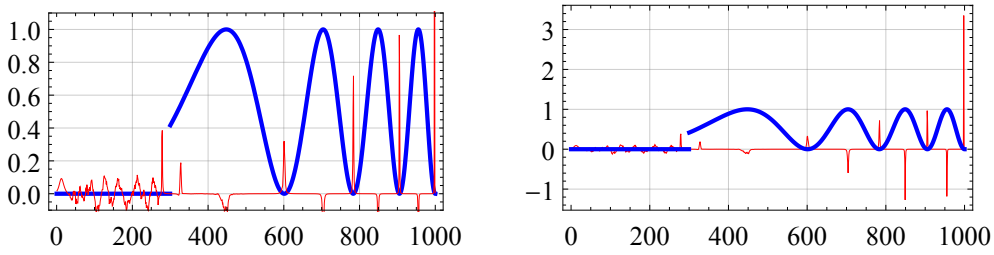
In[761]:=
```
drv4 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {4}], noisydata];
drv4Plot = ListLinePlot[drv4, PlotStyle → {Red}, PlotRange → Full];
Show[{func4Plot, drv4Plot}]
```

Out[763]=



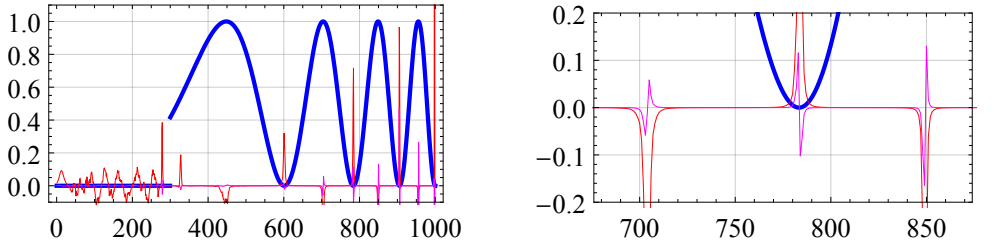
In[764]:=
```
Last@
  (curv = sSavGolBufferConvolve1D[
     sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0, 1, 2}],
     noisydata, AspectRatio → 1]);
curvPlot = ListLinePlot[Last@curv, PlotStyle → Directive[Thin, Red],
   PlotRange → Full];
GraphicsRow[{Show[{funcPlot, curvPlot}], Show[{curvPlot, funcPlot}]},
 ImageSize → 380]
```

Out[766]=



In[767]:=
```
curv = sSavGolBufferConvolve1D[
   sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → Range[0, 3]],
   noisydata, AspectRatio → 1];
curvPlot = ListLinePlot[curv[[-1]], PlotStyle → Directive[Thin, Red],
   PlotRange → Full];
zeroxPlot = ListLinePlot[0.2 curv[[-2]], PlotStyle → Directive[Thin, Magenta],
   PlotRange → Full];
GraphicsRow[{Show[{funcPlot, curvPlot, zeroxPlot}],
  (*Show[{curvPlot,zeroxPlot,funcPlot}],*)
  Show[{funcPlot, curvPlot, zeroxPlot},
   PlotRange → {{680, 870}, {-0.2, 0.2}}]}, ImageSize → 380]
```

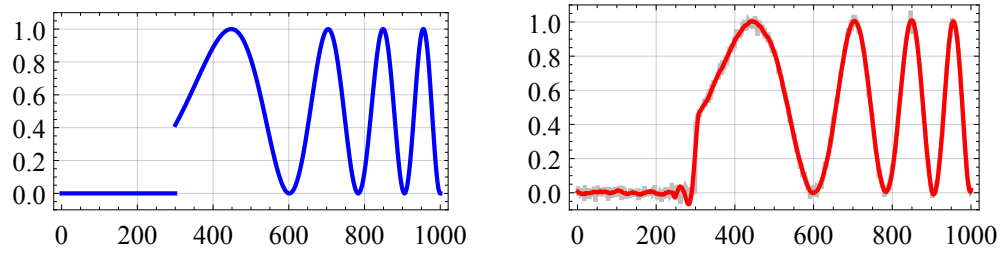
Out[770]=



Order 4, asymmetric kernel

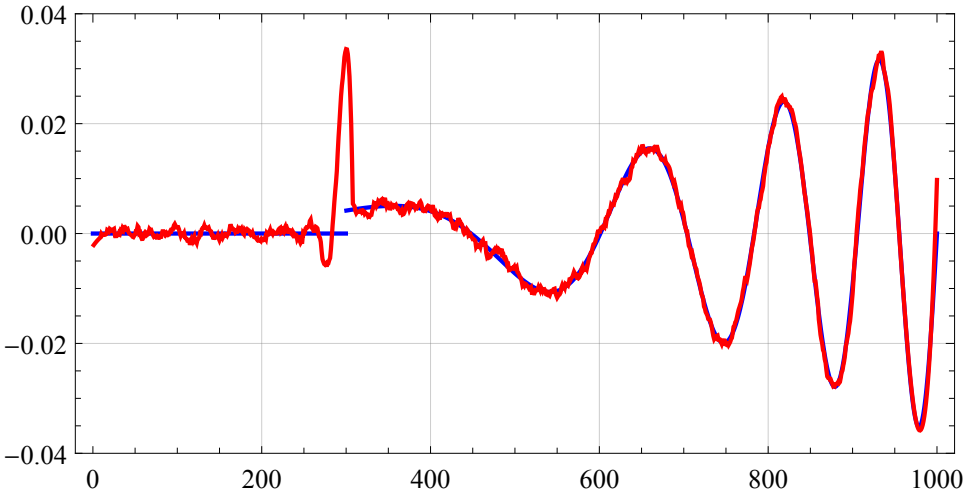
In[771]:=
```
ord = 4; nl = 56; nr = 8;
smoothed = sSavGolBufferConvolve1D[
   sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0}], noisydata];
smoothedPlot = ListLinePlot[smoothed, PlotRange → {-0.1, 1.1},
   PlotStyle → {Red}];
sgFilter = GraphicsRow[{funcPlot, Show[{noisydataPlot, smoothedPlot}]},
  ImageSize → 380]
```

Out[774]=



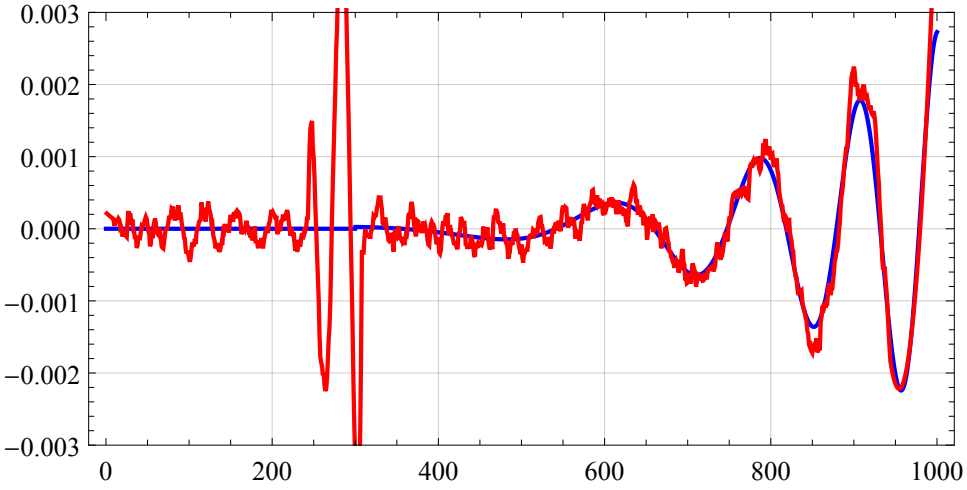
In[775]:=
```
drv1 = sSavGolBufferConvolve1D[
   sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {1}], noisydata];
drv1Plot = ListLinePlot[drv1, PlotStyle → {Red}, PlotRange → Full];
Show[{func1Plot, drv1Plot}]
```

Out[777]=



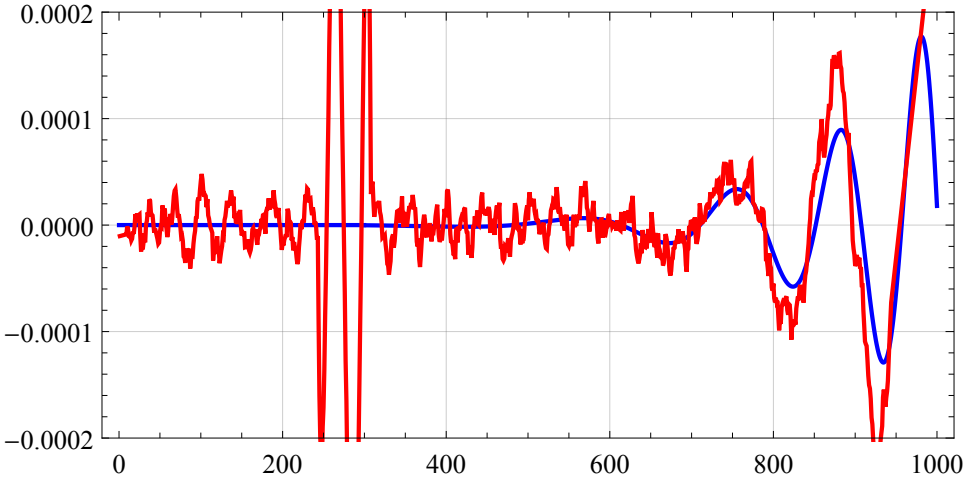
In[778]:=
```
drv2 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {2}], noisydata];
drv2Plot = ListLinePlot[drv2, PlotStyle → {Red}, PlotRange → Full];
Show[{func2Plot, drv2Plot}]
```

Out[780]=



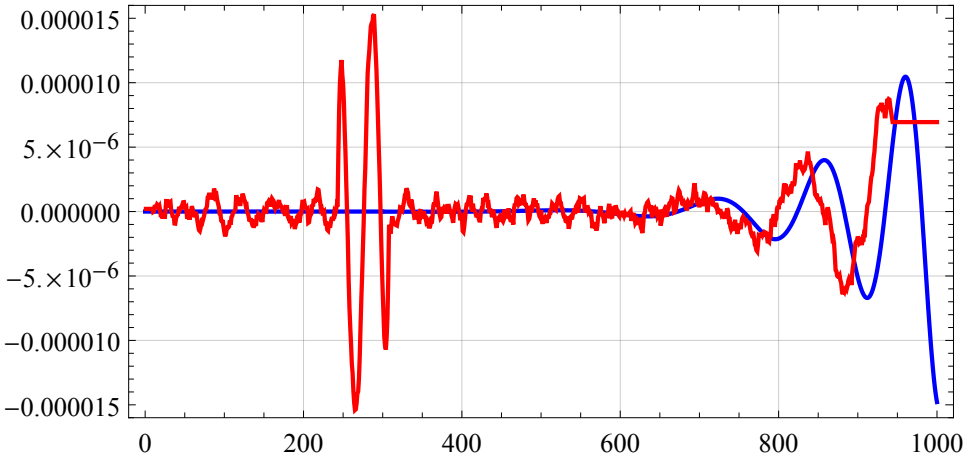
In[781]:=
```
drv3 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {3}], noisydata];
drv3Plot = ListLinePlot[drv3, PlotStyle → {Red}, PlotRange → Full];
Show[{func3Plot, drv3Plot}]
```

Out[783]=



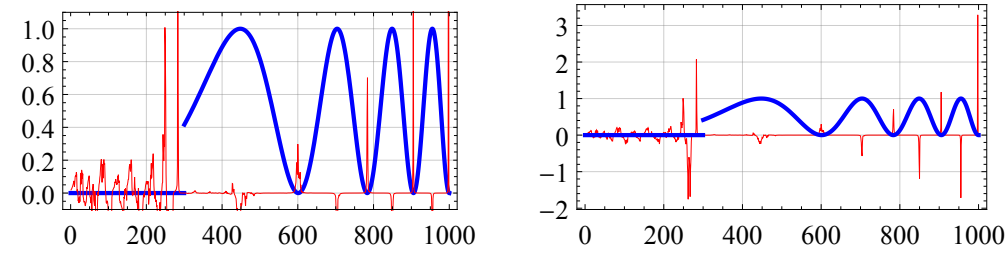
In[784]:=
```
drv4 = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {4}], noisydata];
drv4Plot = ListLinePlot[drv4, PlotStyle → {Red}, PlotRange → Full];
Show[{func4Plot, drv4Plot}]
```

Out[786]=



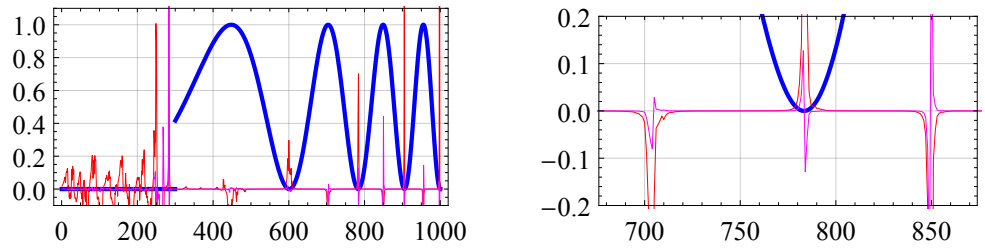
In[787]:=
```
Last@
  (curv = sSavGolBufferConvolve1D[
     sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → {0, 1, 2}],
     noisydata, AspectRatio → 1]);
curvPlot = ListLinePlot[Last@curv, PlotStyle → Directive[Thin, Red],
   PlotRange → Full];
GraphicsRow[{Show[{funcPlot, curvPlot}], Show[{curvPlot, funcPlot}]},
 ImageSize → 380]
```

Out[789]=



In[790]:=
```
curv = sSavGolBufferConvolve1D[
    sSavGolElasticKernels1D[ord, {nl, nr}, "Derivs" → Range[0, 3]],
    noisydata, AspectRatio → 1];
curvPlot = ListLinePlot[curv[[-1]], PlotStyle → Directive[Thin, Red],
    PlotRange → Full];
zeroxPlot = ListLinePlot[0.2 curv[[-2]], PlotStyle → Directive[Thin, Magenta],
    PlotRange → Full];
GraphicsRow[{Show[{funcPlot, curvPlot, zeroxPlot}],
  (*Show[{curvPlot,zeroxPlot,funcPlot}],*)
  Show[{funcPlot, curvPlot, zeroxPlot},
   PlotRange → {{680, 870}, {-0.2, 0.2}}]}, ImageSize → 380]
```

Out[793]=



Now we can restore the ListPlot, Plot, and ListLinePlot options setting to its original value:

In[794]:=
```
SetOptions[ListPlot, Sequence @@ optsListPlot];
SetOptions[Plot, Sequence @@ optsPlot];
SetOptions[ListLinePlot, Sequence @@ optsListLinePlot];
```

2.9 sGrandiSequence

In[795]:=
```
infoAbout@sGrandiSequence
```

sGrandiSequence[n] returns $|n|$-elemental vector {1, −1, …} if integer $n > 0$ and {−1, 1, …} if $n < 0$.

```
Attributes[sGrandiSequence] = {Protected, ReadProtected}
```

```
SyntaxInformation[sGrandiSequence] = {ArgumentsPattern → {_}}
```

Details

sGrandiSequence[n] returns $|n|$-elemental vector $\{1, -1, \ldots\}$ if integer $n > 0$ and $\{-1, 1, \ldots\}$ if $n < 0$. Returns {} if n is zero.

- At first glance, sGrandiSequence appears superfluous, unnecessary, simple, even almost primitive. It can however prove be quite useful, e.g. in converting a series into alternating series, and in multitude of signal analysis applications. The name is derived from the term Grandi's series [Rila, 2014] which is customarily used for the series

$$1 - 1 + 1 - 1 + 1 - 1 + \cdots$$

Examples

Clearing global symbols:

```
In[796]:= Clear[n, k, l, pwr, cos, mod, arr, pad, grd, st, th, optsBoxWhiskerChart, ℋ];
```

Behavior with positive, negative and zero argument:

```
In[797]:= sGrandiSequence /@ {10, -10, 0} // Column

          {1, -1, 1, -1, 1, -1, 1, -1, 1, -1}
Out[797]= {-1, 1, -1, 1, -1, 1, -1, 1, -1, 1}
          {}
```

The Grandi sequence of length 200:

```
In[798]:= sGrandiSequence[200] // Pane[Style[#, 8], 400] &

          {1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1,
           1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1,
           -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1,
Out[798]=  1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1,
           -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1,
           1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1,
           -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1,
           1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1}
```

Applications

We generate the harmonic sequence $1/k,\ k = 1, \ldots, n$:

```
In[799]:= n = 10; Range[n]^-1
```

Out[799]= $\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9}, \frac{1}{10}\right\}$

Now convert it into alternating-sign sequence $(-1)^{k-1}/k,\ k = 1, \ldots, n$, and determine its total:

```
In[800]:= sGrandiSequence[n] Range[n]^-1
          Total[%]
```

Out[800]= $\left\{1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \frac{1}{5}, -\frac{1}{6}, \frac{1}{7}, -\frac{1}{8}, \frac{1}{9}, -\frac{1}{10}\right\}$

Out[801]= $\frac{1627}{2520}$

Directly summing up using the dot product:

```
In[802]:= sGrandiSequence[n].Range[n]^-1
```

$$\text{Out[802]=}\ \frac{1627}{2520}$$

The same for $n = 1, \ldots, 15$:

```
In[803]:= Table[sGrandiSequence[n].Range[n]^-1, {n, 1, 15}] // Style[#, 8] &
```

$$\text{Out[803]=}\ \left\{1, \frac{1}{2}, \frac{5}{6}, \frac{7}{12}, \frac{47}{60}, \frac{37}{60}, \frac{319}{420}, \frac{533}{840}, \frac{1879}{2520}, \frac{1627}{2520}, \frac{20\,417}{27\,720}, \frac{18\,107}{27\,720}, \frac{263\,111}{360\,360}, \frac{237\,371}{360\,360}, \frac{52\,279}{72\,072}\right\}$$

Sum up a large number of terms of the harmonic series and alternating harmonic series:

```
In[804]:= N[Block[{n = 10^5, r}, r = Range[n]^-1;
            {Total[r], sGrandiSequence[n].r}], 10]
```

```
Out[804]= {12.09014613, 0.6931421806}
```

Make sure about the convergence of the alternating harmonic series to natural logarithm of 2:

```
In[805]:= Reap[Sow[N[Log[2], 10]] - %[[2]]]
```

$$\text{Out[805]=}\ \left\{5.000 \times 10^{-6}, \{\{0.6931471806\}\}\right\}$$

Discussion

There is with no doubts a plethora of ways how to achieve the same result as sGrandiSequence provides. I believe you'll immediately come up with many amazing solutions; here are six methods based on built-in symbols Power, Cos, Mod, Array, PadRight and Riffle that recently came to my mind:

```
In[806]:= n = 10;
          {Table[Power[-1, k], {k, 0, n - 1}], Table[Cos[k π], {k, 0, n - 1}],
           Mod[Range[n], 2] /. {0 → -1}, Array[If[OddQ[#], 1, -1] &, n],
           PadRight[{}, n, {1, -1}],
           Riffle[Sequence @@ (ConstantArray[#, n / 2] & /@ {1, -1})˄]} // Column
```

```
Out[807]= {1, -1, 1, -1, 1, -1, 1, -1, 1, -1}
          {1, -1, 1, -1, 1, -1, 1, -1, 1, -1}
          {1, -1, 1, -1, 1, -1, 1, -1, 1, -1}
          {1, -1, 1, -1, 1, -1, 1, -1, 1, -1}
          {1, -1, 1, -1, 1, -1, 1, -1, 1, -1}
          {1, -1, 1, -1, 1, -1, 1, -1, 1, -1}
```

Now we may pose a question about the speed of these solutions. Such question can be answered correctly by performance comparison only. Let each code plus the sGrandiSequence generate alternating sequence of length 10^6, repeating this operation fifty times. Subsequently, the resulting mean times can be statistically compared easily. (Keep calm and be patient, the following code may take some time. The method based on Riffle is omitted since this is the method sGrandiSequence is based on.)

In[808]:=
```
l = 10^6; r = 50; t = Timing;
pwr = Table[(Table[Power[-1, k], {k, 0, l - 1}]; // t)[[1]], {r}];
cos = Table[(Table[Cos[k π], {k, 0, l - 1}]; // t)[[1]], {r}];
mod = Table[((Mod[Range[l], 2] /. {0 → -1}); // t)[[1]], {r}];
arr = Table[(Array[If[OddQ[#], 1, -1] &, l]; // t)[[1]], {r}];
pad = Table[(PadRight[{}, l, {1, -1}]; // t)[[1]], {r}];
grd = Table[(sGrandiSequence[l]; // t)[[1]], {r}];
st = {Mean, Median, StandardDeviation, Min, Max};
TableForm[Through[st[#]] & /@ {pwr, cos, mod, arr, pad, grd},
 TableHeadings → {th = {"Power", "Cos", "Mod", "Array", "PadRight", "Grandi"},
   st /. StandardDeviation → "StdDev"}]
```

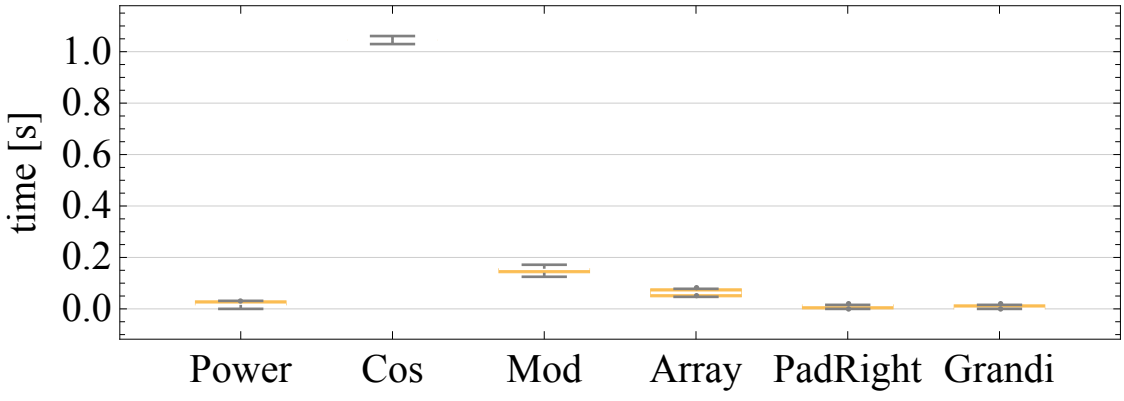
Out[816]//TableForm=

	Mean	Median	StdDev	Min	Max
Power	0.0193441	0.0156001	0.0111696	0.	0.0312002
Cos	1.04801	1.04521	0.00751797	1.02961	1.06081
Mod	0.149761	0.156001	0.00996654	0.124801	0.171601
Array	0.0620884	0.0624004	0.0119972	0.0468003	0.0780005
PadRight	0.00998406	0.0156001	0.00756407	0.	0.0156001
Grandi	0.00436803	0.	0.00707554	0.	0.0156001

Graphical representation in the form of box-and-whisker chart may be helpful in performance assessment:

In[817]:=
```
optsBoxWhiskerChart = Options[BoxWhiskerChart];
SetOptions[BoxWhiskerChart, FrameLabel → {None, "time [s]"},
  GridLines → {None, Automatic}, FrameStyle → Directive[Black, 15],
  AspectRatio → 1 / 3, ImageSize → 420];
BoxWhiskerChart[{pwr, cos, mod, arr, pad, grd}, ChartLabels → th]
```

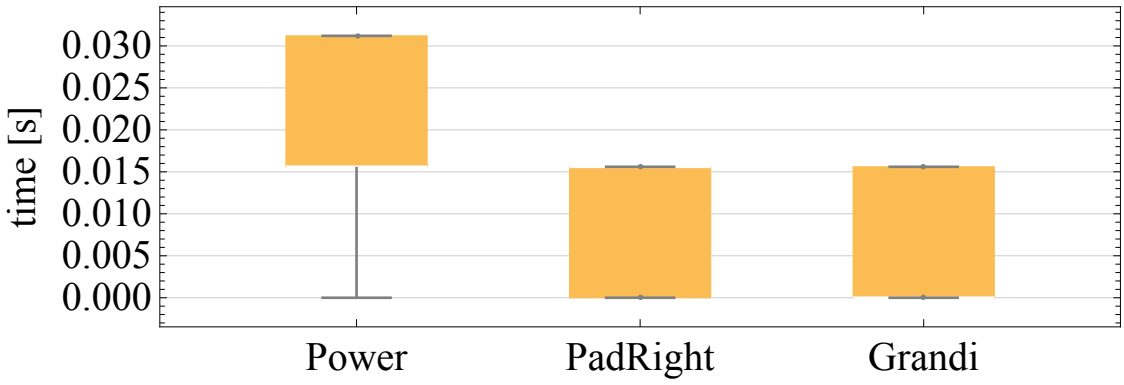
Out[819]=



sGrandiSequence appears to be the winner, while Cos is hopelessly the slowest. A close-up of the three fastest methods, Power, PadRight and sGrandiSequence looks like follows:

```
In[820]:= BoxWhiskerChart[{pwr, pad, grd}, ChartLabels → {"Power", "PadRight", "Grandi"}]
```

Out[820]=



It looks like PadRight and sGrandiSequence are both a bit faster than Power. The speed relation between PadRight and sGrandiSequence is hard to distinguish, so here is the statistical test. The null hypothesis that the PadRight's execution times median is less or equal than the sGrandiSequence's execution times median.

```
In[821]:= ℋ = LocationTest[{pad, grd}, 0, "HypothesisTestData",
              VerifyTestAssumptions → All, AlternativeHypothesis → "Greater"];
          Row[{ℋ["TestDataTable", All] // Magnify[#, 2 / 3] &, Spacer[40],
            Style[Pane[Column[ℋ["TestConclusion", All]], 360], 8]}]
```

Out[822]=

	Statistic	P-Value
Mann-Whitney	1700.	0.000165669
Sign	23	0.000456117
Signed-Rank	333.5	0.000351192

```
The null hypothesis that the median difference is less than or equal to 0
 is rejected at the 5 percent level based on the Mann-Whitney test.
The null hypothesis that the median difference is less than or equal to 0
 is rejected at the 5 percent level based on the Sign test.
The null hypothesis that the median difference is less than or equal to 0
 is rejected at the 5 percent level based on the Signed-Rank test.
```

The null hypothesis is to be rejected (pValue < 0.05). The sGrandiSequence is slightly faster than PadRight, but PadRight being a close runner-up (at least on my laptop). Before implementing even a very simple algorithm, consider its speed. The basic idea of sGrandiSequence is using Riffle:

```
In[823]:= n = 10;
          Riffle[Sequence @@ (ConstantArray[#, n / 2] & /@ {1, -1})^]
```

```
Out[824]= {1, -1, 1, -1, 1, -1, 1, -1, 1, -1}
```

Anybody who delivers even better solution than sGrandiSequence and leaves me a connection will be rewarded with a box of chocolates of his/her own choice.

```
In[825]:= SetOptions[BoxWhiskerChart, Sequence @@ optsBoxWhiskerChart];
```

3

Files, import, export

3.1 sFileSelect

In[826]:= `infoAbout@sFileSelect`

sFileSelect[] returns a list of file(s) selected in dialog.
sFileSelect["DirFiles"] returns a list of files in the directory selected in dialog.
sFileSelect["Paths"] returns a list of subdirs in the directory selected in dialog.
sFileSelect["Resources"] returns a list of all stuff in the directory selected in dialog.

```
Attributes[sFileSelect] = {Protected, ReadProtected}

Options[sFileSelect] = {Filter → {All files → {*}}}

SyntaxInformation[sFileSelect] =
 {ArgumentsPattern → {_., _., OptionsPattern[]}}
```

Details

sFileSelect[*type*, *dep*, *opts*] brings up an interactive system dialog, returns a list of local files/directories selected in the dialog (depending on *type*).

- The dialog starts in the current evaluation notebook directory, or in $HomeDirectory if unsaved.
- Valid *type*'s (and the list they return): "Files" (selected file or files), "DirFiles" (files in a selected directory), "Paths" (subdirectories in a selected directory), "Resources" (all stuff in a selected directory).
- The selected directory is recursively traversed to the depth given by integer *dep* if *dep* > 0 (default 1), or to infinite depth if *dep* == 0 (dep does not affect "Files").

Option "Filter" → {desc$_1$ → {patt$_{11}$, patt$_{12}$, …}, desc$_2$ → {patt$_{21}$, …}, …} is a list of rules. If *type* == "Files", a corresponding drop-down filter selection menu having items desc$_1$, desc$_2$, … appears in the dialog window bottom right corner. For remaining *type*'s, a flat union of all rule values {patt$_{11}$, patt$_{12}$, …, patt$_{21}$, …} is applied as a FileNames filter without displaying. Defaults to {"All files" → {"*"}}.

- If the dialog window is discarded with the Cancel button, $Canceled is returned.

Examples

The sFileSelect function is difficult to demonstrate because the interactive dialogue triggered by its evaluation cannot be included in the output which is in the form of local files/directories selected in the dialog. The only way is inserting a printscreen of the dialog window and explicitly specifying corresponding output returned by the function, which is done for one example case below.

To revive the examples go to the Batch evaluation section, and check the checkbox in the code output therein.

Selecting a file or more files (type "Files")

No arguments

Same as sFileSelect["Files"]:

In[827]:=
```
If[runX, sFileSelect[]]
```

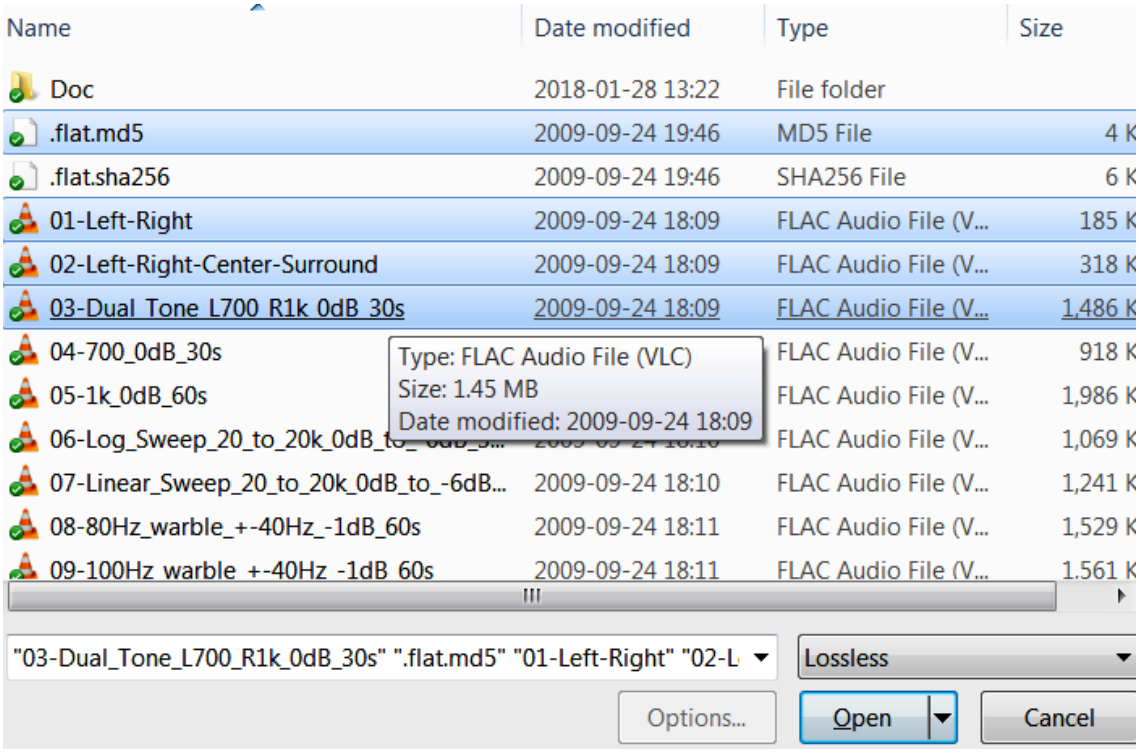
File filter applied:

In[828]:=
```
If[runX, sFileSelect["Filter" → {"Audio (WAV)" → {"*.wav"}}]]
```

More complex file filters applied in the example below.

In[829]:=
```
If[runX,
 sFileSelect[
  "Filter" → {"Lossy" → {"*.mp3", "*.ogg"},
    "Lossless" → {"*.wav", "*.fla*"}}]]
```

The dialog window looks like this:



We clicked through into the appropriate directory, selected "Lossless" in the drop-down filter menu in the bottom right corner of the dialog window, and (using the Ctrl key) selected three FLAC files and one other (checksum) file. After clicking the Open button, the following list was returned:

```
{C:\Users\hledik\Documents\Dropbox\LongTerm\Instruction\DataScience\Audio\Know
lesMichaelBink\.flat.md5,C:\Users\hledik\Documents\Dropbox\LongTerm\Instructio
n\DataScience\Audio\KnowlesMichaelBink\01-Left-
Right.flac,C:\Users\hledik\Documents\Dropbox\LongTerm\Instruction\DataScience\
Audio\KnowlesMichaelBink\02-Left-Right-Center-
Surround.flac,C:\Users\hledik\Documents\Dropbox\LongTerm\Instruction\DataScien
ce\Audio\KnowlesMichaelBink\03-Dual_Tone_L700_R1k_0dB_30s.flac}
```

Note that in addition to FLAC files, also .flat.md5 and .flat.sha256 checksum files pass through the filter "Filter"→{"Lossy"→{"*.mp3","*.ogg"},"Lossless"→{"*.wav","*.fla*"}}. This is due to that the asterisk in the "*.fla*" regular pattern matches any string including the empty one.

Another example using file filter is, e.g.,

```
In[830]:= If[runX,
           sFileSelect[
            "Filter" → {"Mathematica notebook" → {"*.nb"},
              "Mathematica M file" → {"*.m"}}]]
```

In the following examples, neither output returned nor dialog windows will be demonstrated; the inputs will be left unevaluated. Evaluating them and playing with them is left to the reader.

First argument only

Same as sFileSelect[]:

```
In[831]:= If[runX, sFileSelect["Files"]]

In[832]:= If[runX, sFileSelect["f"]]
```

Same as sFileSelect["Filter"→{"Audio (WAV)"→{"*.wav"}}]:

```
In[833]:= If[runX, sFileSelect["f", "Filter" → {"Audio (WAV)" → {"*.wav"}}]]
```

Second argument only

Since the first argument defaults to "Files", giving the second argument only is sufficient. The 2nd argument has no effect with type "Files".

Same as sFileSelect[]:

```
In[834]:= If[runX, sFileSelect[2]]
```

A wrong depth is ignored with "Files", same as sFileSelect[]

```
In[835]:= If[runX, sFileSelect[-1]]
```

Both arguments

The second argument has no effect with type "Files". Same as sFileSelect[“Filter”→{“Audio (WAV)”→{“*.wav”}}]:

```
In[836]:= If[runX, sFileSelect["f", 2, "Filter" → {"Audio (WAV)" → {"*.wav"}}]]
```

Listing file(s) in a selected directory (type "DirFiles")

First argument only

Default depth 1:

```
In[837]:= If[runX, sFileSelect["DirFiles"]]

In[838]:= If[runX, sFileSelect["d"]]
```

Same as sFileSelect["DirFiles"] but filter applied:

```
In[839]:= If[runX, sFileSelect["d", "Filter" → {"Audio (WAV)" → {"*.wav"}}]]
```

Same as sFileSelect["DirFiles"] but more complex filters applied:

```
In[840]:= If[runX, sFileSelect["dir",
            "Filter" → {"Lossy" → {"*.mp3", "*.ogg"},
              "Lossless" → {"*.wav", "*.fla*"}}]]
```

Both arguments

Same as sFileSelect["DirFiles"] but scans down to depth of 2:

```
In[841]:= If[runX, sFileSelect["d", 2]]
```

Same as previous but more complex filters applied:

```
In[842]:= If[runX, sFileSelect["dir", 4,
            "Filter" → {"Lossy" → {"*.mp3", "*.ogg"},
              "Lossless" → {"*.wav", "*.fla*"}}]]
```

Listing subdirs in a selected directory (type "Paths")

First argument only

Default depth 1:

```
In[843]:= If[runX, sFileSelect["Paths"]]
```

```
In[844]:= If[runX, sFileSelect["p"]]
```

Both arguments

Same as sFileSelect["Paths"], but to infinite depth

```
In[845]:= If[runX, sFileSelect["p", 0]]
```

Same as previous but filtered:

```
In[846]:= If[runX, sFileSelect["p", 0, "Filter" → {"Dot directory" → {"*.*"}}]]
```

Listing all stuff in a selected directory (type "Resources")

First argument only

Default depth 1:

```
In[847]:= If[runX, sFileSelect["Resources"]]
```

```
In[848]:= If[runX, sFileSelect["r"]]
```

Same as previous but filtered:

```
In[849]:= If[runX, sFileSelect["res", "Filter" → {"Dot directory" → {"*.*"}}]]
```

```
In[850]:= If[runX, sFileSelect["resources", "Filter" → {"Dot directory" → {"*.*"}}]]
```

```
In[851]:= If[runX, sFileSelect["r", "Filter" → {"Audio (WAV)" → {"*.wav"}}]]
```

```
In[852]:= If[runX, sFileSelect["r", "Filter" → {"Test" → {".*"}}]]
```

Both arguments

Same as sFileSelect["Resources"], but to infinite depth

```
In[853]:= If[runX, sFileSelect["r", 0]]
```

Same as previous but filtered:

```
In[854]:= If[runX, sFileSelect["res", 0, "Filter" → {"Dot directory" → {"*.*"}}]]
```

Same as previous, but to depth 1 only, second argument may be omitted

In[855]:=
```
If[runX, sFileSelect["resources", 1, "Filter" → {"Dot directory" → {"*.*"}}]]
```

Applications

sFileSelect proved to be a very useful tool in data processing practice, when a lot of files is to be interactively selected for import.

3.2 sDirectoryDigest

In[856]:=
```
infoAbout@sDirectoryDigest
```

sDirectoryDigest[*dir*] saves hexadecimal MD5 hash codes for
files in the *dir* tree to file .deep.md5 down to infinite recursion depth.

```
Attributes[sDirectoryDigest] = {Protected, ReadProtected}

Options[sDirectoryDigest] = {Basename → .deep.,
  Mode → Binary, Range → All, Format → HexString, Debug → False}

SyntaxInformation[sDirectoryDigest] =
 {ArgumentsPattern → {_., _., _., OptionsPattern[]}}
```

Details

sDirectoryDigest[*dir*, *dep*, *opts*] saves hexadecimal MD5 hash codes (using the "md5sum" format) for files in the directory *dir* tree to file .deep.md5 down to recursion depth *dep*: default 0 means true depth ∞, 1 only scans dir files, 2 adds files in subdirs of dir etc.

- Directory *dir* must be specified by an absolute path, it is beneficial using

  FileNameJoin[{NotebookDirectory[], "somedir"}]

  or simply

  NotebookDirectory[]

  Unless *dir* is given, an interactive dialog pops out. The dialog starts in the current evaluation notebook directory, or in $HomeDirectory if unsaved.
- If the dialog window is discarded with the Cancel button, $Canceled is returned.

sDirectoryDigest[*dir*, *dep*, {$type_1$, …}, *opts*] same as above but uses typen hash codes, saving them to respective files .deep.$type_n$ (default {"MD5"}).

- See FileHash built-in function for supported hash types.
- Options:
  - "Basename"→string (default ".deep." for $dep \neq 1$, ".flat." otherwise)
  - "Mode"→"Binary" (default) or "Text"

- Options "Range" and "Format" correspond to the 3rd and 4th argument of the FileHash system function (All and "HexString" by default).
- Option IgnoreCase (Automatic by default) can be passed on to FileNames system function.

- If the dialog window is discarded with the Cancel button, $Canceled is returned.

Examples

To revive the examples go to the Batch evaluation section, and check the checkbox in the code output therein.

No directory is given, an interactive dialog will pop out:

```
In[857]:= If[runX, sDirectoryDigest[]]
```

This will create a .deep.md5 hash file of this notebook directory down to recursion depth infinity:

```
In[858]:= If[runX, sDirectoryDigest[NotebookDirectory[]]]
```

This will create a .deep.md5 hash file of a directory specified in the dialog down to recursion depth 1. Three hash files will be created with the FILES basename:

```
In[859]:= If[runX, sDirectoryDigest[1, {"MD5", "SHA", "SHA512"}, "Basename" → "FILES."]]
```

The following command uses all implemented options (ignore the unrecognized option name coloring for IgnoreCase):

```
In[860]:= If[runX,
           sDirectoryDigest[FileNameJoin[{NotebookDirectory[], "sDirectoryDigest"}],
            0(*1*), {"MD5", "SHA512"}, "Basename" → "FILES.", "Mode" → "Binary",
            "Range" → All, "Format" → "HexString", IgnoreCase → Automatic,
            "Debug" → True]]
```

Applications

sDirectoryDigest comes handy when one needs quickly create hash file(s) in the "md5sum" format. It is advantageous to save the following code (or something customized to your taste, but omit all the If[runX, …] conditions) in a file, say, .deep.nb, and use it interactively if necessary.

Directory digest maker

```
In[861]:= If[runX, Needs["Snapdragon`"]]
```

MD5 and SHA512, infinite depth

Results in .deep.md5, .deep.sha512

```
In[862]:= If[runX, sDirectoryDigest[{"MD5", "SHA512"}]]
```

MD5 and SHA512, files in a specified directory only

Results in .flat.md5, .flat.sha512

In[863]:=
```
If[runX, sDirectoryDigest[1, {"MD5", "SHA512"}]]
```

MD5, infinite depth

Results in .deep.md5

In[864]:=
```
If[runX, sDirectoryDigest[]]
```

MD5, files in a specified directory only

Results in .flat.md5

In[865]:=
```
If[runX, sDirectoryDigest[1]]
```

3.3 sBulkExport

In[866]:=
```
infoAbout@sBulkExport
```

sBulkExport[{s_1, ...}, *opts*] exports expressions assigned to symbols
s_i to files $s_i.e_j$ with e_j given by "Extensions" → {e_1, ...} (default {"png"}).
sBulkExport[] returns table of all formats and extensions supported.

```
Attributes[sBulkExport] = {HoldFirst, Protected, ReadProtected}

Options[sBulkExport] =
 {Extensions → {png}, Directory → ., Elements → None, Debug → False}

SyntaxInformation[sBulkExport] = {ArgumentsPattern → {OptionsPattern[]}}
```

Details

sBulkExport[{s_1, s_2, ...}, *opts*] exports expressions assigned to symbols s_i to files $s_i.e_j$ with strings e_j given by the option "Extensions" → {e_1, e_2, ...} (default {"png"}).

- The export directory can be specified relative to the notebook directory by a string value of the option "Directory" (default "." which correspond to the notebook directory).
- For an unsaved notebook, export goes to $TemporaryDirectory/sBulkExport/sUniqueName[].
- Export elements and/or explicit export format can be specified by the option "Elements" (default None; no syntax check, see reference pages of the respective formats).
- Any further options are passed to the Export function, however, their pertinence to the formats being exported is not checked.

sBulkExport["x"|"f"|"e"] yields sorted list of: formats and extensions, formats, extensions supported, respectively (case insensitive).

sBulkExport[] returns full table of all formats and extensions supported (including links to the *Wolfram Language* & System Documentation Center, and other information).

sBulkExport[*op*] returns full table filtered with a selection operator *op* (based on operator form of system functions Select, Cases, …), e.g.

Select[#["Format"] == "EPS" &]

Select[MemberQ[#["Ext"], "pdf"] &]

Examples

Clearing global symbols:

```
In[867]:= Clear[ox, of, ux, ue, i1, i2, optssBulkExport, hello, sine, sineModulated];
```

To revive the examples go to the Batch evaluation section, and check the checkbox in the code output therein.

Formats and extensions available

In order to save space, Magnify[#,0.5]& will be applied to those examples that provide tabular output. First, tabulate all formats and extensions (this command is left unevaluated to save a lot of space):

```
In[868]:= sBulkExport[] // Magnify[#, 1 / 6] &
```

Out[868]=

Format	Ext	Purpose	MIME type	Options	Limitations	Note
AIFF	aif aifc aiff	Sampled Audio	audio/aiff audio/x-aiff audio/x-aifc	"AudioChannels" "AudioEncoding" IncludeMetaInformation SampleRate	—	Audio Interchange File since 1988 by Apple an Uncompressed mono or Binary format
AU	au	Sampled Audio	audio/basic	"AudioChannels" "AudioEncoding" IncludeMetaInformation SampleRate	—	By Sun Microsystems in Identical to the SND for Various sample rates, a Binary format
AVI	avi	Video	video/avi	Background ImageSize "FrameRate"	Generates uncompressed RGB frames with color resolution of 8b per channel	Audio Video Interleave Microsoft AVI format (1 Binary format
BMP	bmp dib	Sampled Image	image/bmp	ImageSize "ImageCompression" "ImageTopOrientation"	Up to 256 8b RGB color palettes	Windows Bitmap format Supports transparency Uncompressed or RL en Different bit depths in c Binary format
CDF	cdf	Wolfram System	application/vnd.wolfram.cdf application/vnd.wolfram.cdf.text	—	—	Wolfram Computable D Text, typeset exprs, gra ASCII format based on
CSV	csv	Text Data	text/comma-separated-values text/csv	Alignment CharacterEncoding "EmptyField" "FillRows" "TableHeadings" "TextDelimiters"	—	Comma-Separated Valu Records of numerical & Spreadsheet apps as ar Similar to TSV Plain text format
DICOM	dcm dcm.gz dic dic.gz	Sampled Image Data	application/dicom	ImageSize "ImageTopOrientation" "Smoothing"	—	Digital Imaging & Comm Multiple raster images Conforms to Sec. PS 3. "*.gz" GZIPped form Binary format
EMF	emf	Vector Image Sampled Image Text	—	—	—	Microsoft Enhanced Met Successor of the 16b W Binary format
EPS	eps eps.gz eps.bz2 epsf epsf.gz epsf.bz2	Vector Image Sampled Image Text	application/postscript application/eps application/x-eps image/eps image/x-eps	Background ImageSize ImageResolution "AllowRasterization" "EmbeddedFonts" "PreviewFormat"	—	Encapsulated PostScript Uses PostScript languag PS lev. 2, incl. certain l Wolfram & other nonsto "*.gz", "*.bz2" compres Plain text format
FITS	fit fits	Sampled Image Data	application/fits image/fits	DataReversed "Append" "CompressionMethod"	—	Flexible Image Transpo FITS scientific image ar Endorsed by NASA and Multiple HDUs (header- each an image, ASCII o
FLAC	flac	Sampled Audio	audio/x-flac	"AudioChannels" "AudioEncoding" IncludeMetaInformation SampleRate	Up to 8 channels	Free Lossless Audio Coc Open standard for audi Linear prediction and ru Binary format
FLV	flv	Video Sampled Animation	video/x-flv	Background ImageSize "CompressionMethod" "VideoEncoding" "BlockSize" "AnimationDuration" "ControlAppearance" "FrameRate" "RepeatAnimation" "Scalable" "HTMLFile" "ThumbnailFile"	—	Adobe/Macromedia Flas Supports Adobe Flash F Binary format
GIF	gif	Sampled Image Sampled Animation	image/gif	"AnimationRepetitions" "ColorMapLength" "ControlAppearance" "DisplayDurations" Dithering ImageFormattingWidth ImageSize "ImageTopOrientation" "Interlaced" "QuantizationMethod" "TransparentColor"	—	Graphics Interchange F Supports the GIF89a St LZW lossless after restr Binary format
HTML	html htm	Web Pages Sampled Image	text/html	"Content" "ConversionRules" "ConvertClosed" "ConvertLinkedNotebooks" "ConvertReverseClosed" "CSS" "FullDocument" "Graphics3DOutput" "GraphicsOutput" "HeadAttributes" "HeadElements" "ManipulateOutput" "MathOutput"	—	Hypertext Markup Lang Fully supports HTML 4.0 Output conforms to the Since 1996 by the Worl Predecessor of XHTML Plain text format
ICNS	icns	Sampled Image	—	—	Image & Graphics std dims 512, 256, 128, 48, 32, 16 Otherwise scaled its longest dim fits the closest std	Macintosh icons format Supports alpha channel icons at different color r Binary format
ICO	ico	Sampled Image	image/vnd.microsoft.icon	—	Up to 256x256 images, larger will be scaled down	Microsoft Windows icon Similar to CUR format Multiple icons at diff col Supports alpha channel Binary format
JPEG	jpeg jpg	Sampled Image	image/jpeg	ImageResolution ImageSize IncludeMetaInformation "ImageTopOrientation" "ColorSpace" "CompressionLevel" "Progressive" "Smoothing" ImageFormattingWidth	8b per color channel	Joint Photographic Expe Lossy 8x8-block DCT co Grayscale, RGB, CMYK Exif 2.3, IPTC 4.2, XMP Binary format
JPEG2000	jp2 j2k	Sampled Image	image/jp2	ImageSize "ImageTopOrientation" "BitDepth" "CompressionLevel" "ImageTopOrientation" "TileSize" "ColorSpace" "ImageEncoding" "TileDimensions"	—	Joint Photographic Expe Part 1 of the ISO 15444 Lossy/lossless compress Variety of color resoluti Binary format
M4A	m4a mp4	Sampled Audio	audio/mp4M4A	"AudioChannels" "BitRate" IncludeMetaInformation SampleRate	Sample rate 8-96kHz Up to 48 channels	ISO/IEC 13818-7 stand A perception-based los Binary format
MGF	mgf	Sampled Image Wolfram System	—	ImageSize	A single image as RGB raster at 8b per color	Mathematica Graphics F Wolfram System MGF b WS user i'face for storin Binary format

showing 1–20 of 54

Tabulate selected formats (multiple extensions may be pertinent to a single format):

```
In[869]:= sBulkExport[Select[#["Format"] == "JPEG" &]] // Magnify[#, 1 / 4] &
```

Out[869]=

Format	Ext	Purpose	MIME type	Options	Limitations	Note
JPEG	jpeg jpg	Sampled Image	image/jpeg	ImageResolution ImageSize IncludeMetaInformation "ImageTopOrientation" "ColorSpace" "CompressionLevel" "Progressive" "Smoothing" ImageFormattingWidth	8b per color channel	Joint Photogra Lossy 8x8-blo Grayscale, RG Exif 2.3, IPTC Binary format

```
In[870]:= sBulkExport[Select[StringStartsQ[#["Format"], "JPEG"] &]] //
            Magnify[#, 1 / 4] &
```

Out[870]=

Format	Ext	Purpose	MIME type	Options	Limitations	Note
JPEG	jpeg jpg	Sampled Image	image/jpeg	ImageResolution ImageSize IncludeMetaInformation "ImageTopOrientation" "ColorSpace" "CompressionLevel" "Progressive" "Smoothing" ImageFormattingWidth	8b per color channel	Joint Photogra Lossy 8x8–blo Grayscale, RG Exif 2.3, IPTC Binary format
JPEG2000	jp2 j2k	Sampled Image	image/jp2	ImageSize "ImageTopOrientation" "BitDepth" "CompressionLevel" "ImageTopOrientation" "TileSize" "ColorSpace" "ImageEncoding" "TileDimensions"	—	Joint Photogra Part 1 of the I: Lossy/lossless Variety of colo Binary format

Tabulate selected extensions:

```
In[871]:= sBulkExport[Select[MemberQ[#["Ext"], "pdf"] &]] // Magnify[#, 1 / 4] &
```

Out[871]=

Format	Ext	Purpose	MIME type	Options	Limitations	Note
PDF	pdf pdf.gz pdf.bz2	Vector Image Sampled Image Text	application/pdf	ImageSize ImageResolution "AllowRasterization"	PDF Version 1.5 and earlier	Adobe Acrobat Text, fonts, im in a device-/re Can store emb Multiple lossy/l "*.gz", "*.bz2" Binary format

Tabulate selected purpose:

```
In[872]:= sBulkExport[Select[MemberQ[#["Purpose"], "Wolfram System"] &]] //
            Magnify[#, 1 / 6] &
```

Out[872]=

Format	Ext	Purpose	MIME type	Options	Limitations	Note
CDF	cdf	Wolfram System	application/vnd.wolfram.cdf application/vnd.wolfram.cdf.text	—	—	Wolfram Computable Docu Text, typeset exprs, graph ASCII format based on the
MGF	mgf	Sampled Image Wolfram System	—	ImageSize	A single image as RGB raster at 8b per color	Mathematica Graphics Forr Wolfram System MGF bitm WS user i'face for storing r Binary format
MX	mx	Wolfram System	—	—	Cannot be exchanged between OSs that differ in $SystemWordLength ".mx" by newer versions of Wolfram Sys may not be usable by olders	Wolfram System serialized Designed for distribution o Arb. WL exprs in serialized Adds "$SystemID." before Can be created with Dump Binary format
NB	nb	Wolfram System	application/mathematica application/vnd.wolfram.mathematica	—	—	Wolfram System notebook Text, typeset exprs, graph Supports interactive eleme ASCII format based on the
Package	wl m	Wolfram System	application/vnd.wolfram.wl application/vnd.wolfram.mathematica.package	PageWidth "Comments"	—	Wolfram System package s Program code, num & text images, 3D geometries, so Plain ASCII text format
WDX	wdx	Wolfram System Sampled Image	—	—	—	Wolfram Data Exchange, 2 Storing and exchanging ex Arbitrary WL exprs in a ser Binary format

List and count all formats, including their relevant extensions:

```
In[873]:= sBulkExport["x"]
          Length@%

Out[873]= {{AIFF, {aif, aifc, aiff}}, {AU, {au}}, {AVI, {avi}}, {BMP, {bmp, dib}},
           {CDF, {cdf}}, {CSV, {csv}}, {DICOM, {dcm, dcm.gz, dic, dic.gz}},
           {EMF, {emf}}, {EPS, {eps, eps.gz, eps.bz2, epsf, epsf.gz, epsf.bz2}},
           {FITS, {fit, fits}}, {FLAC, {flac}}, {FLV, {flv}}, {GIF, {gif}},
           {HTML, {html, htm}}, {ICNS, {icns}}, {ICO, {ico}}, {JPEG, {jpeg, jpg}},
           {JPEG2000, {jp2, j2k}}, {M4A, {m4a, mp4}}, {MGF, {mgf}}, {MIDI, {mid}},
           {MP3, {mp3}}, {MX, {mx}}, {NB, {nb}}, {OggVorbis, {oga, ogg}},
           {Package, {wl, m}}, {PBM, {pbm}}, {PCX, {pcx}}, {PDF, {pdf, pdf.gz, pdf.bz2}},
           {PGM, {pgm}}, {PICT, {pct, pic, pict}}, {PNG, {png}}, {PNM, {pnm}},
           {PPM, {ppm}}, {PXR, {pxr}}, {QuickTime, {mov, qt}}, {RTF, {rtf}},
           {SCT, {sct}}, {SND, {snd}}, {SVG, {svg, svgz}}, {SWF, {swf}}, {TeX, {tex}},
           {TeXFragment, {tex}}, {TGA, {tga}}, {TIFF, {tiff, tif}}, {TSV, {tsv}},
           {WAV, {wav}}, {Wave64, {w64}}, {WDX, {wdx}}, {WMF, {wmf}}, {XBM, {xbm}},
           {XHTML, {xht, xhtml, xml}}, {XLS, {xls}}, {XLSX, {xlsx, xlsm}}}

Out[874]= 54
```

The "x" specification (as well as the "f", "e") is case insensitive:

In[875]:=
```
sBulkExport["X"]
```

Out[875]=
```
{{AIFF, {aif, aifc, aiff}}, {AU, {au}}, {AVI, {avi}}, {BMP, {bmp, dib}},
 {CDF, {cdf}}, {CSV, {csv}}, {DICOM, {dcm, dcm.gz, dic, dic.gz}},
 {EMF, {emf}}, {EPS, {eps, eps.gz, eps.bz2, epsf, epsf.gz, epsf.bz2}},
 {FITS, {fit, fits}}, {FLAC, {flac}}, {FLV, {flv}}, {GIF, {gif}},
 {HTML, {html, htm}}, {ICNS, {icns}}, {ICO, {ico}}, {JPEG, {jpeg, jpg}},
 {JPEG2000, {jp2, j2k}}, {M4A, {m4a, mp4}}, {MGF, {mgf}}, {MIDI, {mid}},
 {MP3, {mp3}}, {MX, {mx}}, {NB, {nb}}, {OggVorbis, {oga, ogg}},
 {Package, {wl, m}}, {PBM, {pbm}}, {PCX, {pcx}}, {PDF, {pdf, pdf.gz, pdf.bz2}},
 {PGM, {pgm}}, {PICT, {pct, pic, pict}}, {PNG, {png}}, {PNM, {pnm}},
 {PPM, {ppm}}, {PXR, {pxr}}, {QuickTime, {mov, qt}}, {RTF, {rtf}},
 {SCT, {sct}}, {SND, {snd}}, {SVG, {svg, svgz}}, {SWF, {swf}}, {TeX, {tex}},
 {TeXFragment, {tex}}, {TGA, {tga}}, {TIFF, {tiff, tif}}, {TSV, {tsv}},
 {WAV, {wav}}, {Wave64, {w64}}, {WDX, {wdx}}, {WMF, {wmf}}, {XBM, {xbm}},
 {XHTML, {xht, xhtml, xml}}, {XLS, {xls}}, {XLSX, {xlsx, xlsm}}}
```

List and count all formats, without their relevant extensions:

In[876]:=
```
sBulkExport["f"]
Length@%
```

Out[876]=
```
{AIFF, AU, AVI, BMP, CDF, CSV, DICOM, EMF, EPS, FITS, FLAC, FLV, GIF, HTML, ICNS,
 ICO, JPEG, JPEG2000, M4A, MGF, MIDI, MP3, MX, NB, OggVorbis, Package, PBM, PCX,
 PDF, PGM, PICT, PNG, PNM, PPM, PXR, QuickTime, RTF, SCT, SND, SVG, SWF, TeX,
 TeXFragment, TGA, TIFF, TSV, WAV, Wave64, WDX, WMF, XBM, XHTML, XLS, XLSX}
```

Out[877]=
```
54
```

List distinct extensions only:

In[878]:=
```
sBulkExport["e"]
Length@%
```

Out[878]=
```
{aif, aifc, aiff, au, avi, bmp, cdf, csv, dcm, dcm.gz, dib, dic, dic.gz, emf,
 eps, eps.bz2, epsf, epsf.bz2, epsf.gz, eps.gz, fit, fits, flac, flv,
 gif, htm, html, icns, ico, j2k, jp2, jpeg, jpg, m, m4a, mgf, mid, mov,
 mp3, mp4, mx, nb, oga, ogg, pbm, pct, pcx, pdf, pdf.bz2, pdf.gz, pgm, pic,
 pict, png, pnm, ppm, pxr, qt, rtf, sct, snd, svg, svgz, swf, tex, tga, tif,
 tiff, tsv, w64, wav, wdx, wl, wmf, xbm, xht, xhtml, xls, xlsm, xlsx, xml}
```

Out[879]=
```
81
```

Compare the number of formats from "x" and "f", they are the same:

```
In[880]:= ox = sBulkExport["x"];
          of = sBulkExport["f"];
          {Length@ox, Length@of}
          of === First /@ ox

Out[880]= {54, 54}

Out[881]= True
```

Compare the number of extensions from "x" and "e", they are the same:

```
In[882]:= ux = Union[Flatten[Last /@ sBulkExport["x"]]];
          ue = sBulkExport["e"];
          {Length@ux, Length@ue}
          ue === ux

Out[882]= {81, 81}

Out[883]= True
```

Applications

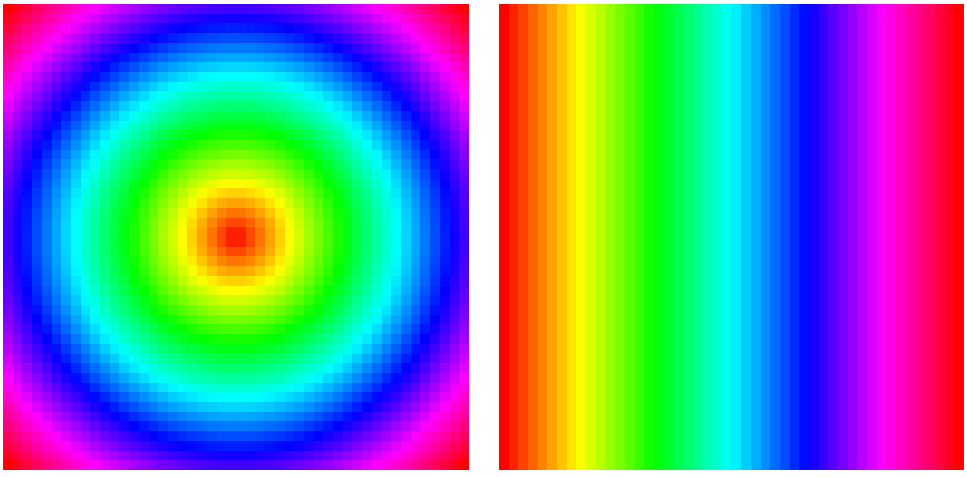
sBulkExport proved to be a very useful function. Imagine you have a large notebook with many complex graphic, plots, etc. You just save each of them into a suitable valid symbol (like i1, i2, sine, sineModulated in the application examples below), and at the end of the notebook (or any appropriate place within it) use the sBulkExport, specify the output directory, formats and maybe compression level, and you are done.

Image export

First of all, we define some tiny images to be exported:

```
In[884]:= ({i1, i2} = Through[{RadialGradientImage[##] &, LinearGradientImage[##] &}[
              Hue, {48, 48}]]) // GraphicsRow
```

Out[884]=



In the following examples, the inputs will be left unevaluated. Evaluating them and playing with them is left to the reader. The export goes to the sBulkExport subdirectory of the directory this

notebook is saved in (will be created if no sBulkExport subdirectory exists). To avoid specifying the output directory for each command particularly, we set

```
In[885]:= optssBulkExport = Options[sBulkExport];
          SetOptions[sBulkExport, "Directory" → "sBulkExport"]

Out[886]= {Extensions → {png}, Directory → sBulkExport, Elements → None, Debug → False}
```

Exporting to the PNG image format (default):

```
In[887]:= If[runX, sBulkExport[{i1, i2}]]
```

For usage with LaTeX, {"eps", "pdf"} is recommended:

```
In[888]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"eps", "pdf"}]]
```

Image content can even be exported to the Excel or general tabular format, but do not expect any reasonable result:

```
In[889]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"xlsx"}]]

In[890]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"csv"}]]

In[891]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"tsv"}]]
```

Explicitly specified the "CompressionLevel" Export option for lossy image formats (ignore the fact that the option name is red highlighted):

```
In[892]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"jpg", "j2k", "jp2"},
            "CompressionLevel" → 0.9, "Elements" → None]]

In[893]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"jpg", "j2k", "jp2"},
            "CompressionLevel" → 0]]
```

Some image formats may not be available on every platform:

```
In[894]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"pct"}]]

In[895]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"pict"}]]
```

The *Wolfram Language* binary dump format gets the operating system label:

```
In[896]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"mx"}]]
```

Other formats one can reasonably export images to are, e.g.

```
In[897]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"htm"}]]

In[898]:= If[runX, sBulkExport[{i1, i2}, "Extensions" → {"html"}]]

In[899]:= If[runX, sBulkExport[{i2}, "Extensions" → {"rtf"}]]
```

```
In[900]:= If[runX, sBulkExport[{i2}, "Extensions" → {"tex"},
            "Directory" → FileNameJoin[{"sBulkExport", "TeX"}]]]

In[901]:= If[runX, sBulkExport[{i2}, "Extensions" → {"tex"}, "Elements" → "TeXFragment",
            "Directory" → FileNameJoin[{"sBulkExport", "TeXFragment"}]]]
```

But check what is exported in these cases:

```
In[902]:= If[runX, hello = 1; sBulkExport[{hello}]]

In[903]:= If[runX, hello = 1;
           sBulkExport[{hello}, "Extensions" → {"jpg"}, "CompressionLevel" → 0.95]]
```

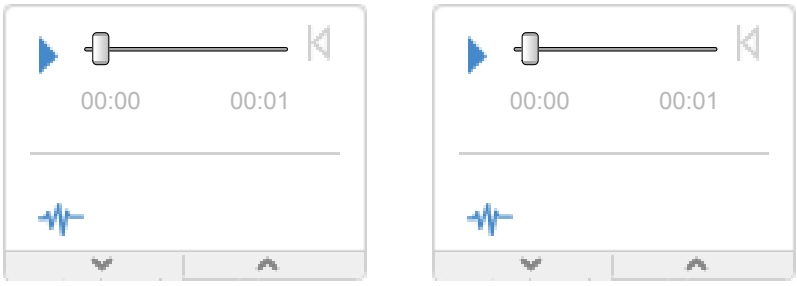
Audio export

First of all, we define some tiny audio signals to be exported:

```
In[904]:= sine = AudioGenerator["Sin"];
          sineModulated = AudioGenerator[{"Sin", 1000 + 1000 Sin[2 * Pi #] &}];

In[906]:= Row[{sine, sineModulated}, Spacer[20]]
```

Out[906]=



Then we ask for tabulating usable audio formats:

```
In[907]:= sBulkExport[Select[MemberQ[#["Purpose"], "Sampled Audio"] &]] //
           Magnify[#, 1 / 4] &
```

Out[907]=

Format	Ext	Purpose	MIME type	Options	Limitations	Note
AIFF	aif aifc aiff	Sampled Audio	audio/aiff audio/x-aiff audio/x-aifc	"AudioChannels" "AudioEncoding" IncludeMetaInformation SampleRate		Audio Intercha since 1988 by Uncompressed Binary format
AU	au	Sampled Audio	audio/basic	"AudioChannels" "AudioEncoding" IncludeMetaInformation SampleRate		By Sun Micros Identical to th Various sample Binary format
FLAC	flac	Sampled Audio	audio/x-flac	"AudioChannels" "AudioEncoding" IncludeMetaInformation SampleRate	Up to 8 channels	Free Lossless Open standard Linear predicti Binary format
M4A	m4a mp4	Sampled Audio	audio/mp4M4A	"AudioChannels" "BitRate" IncludeMetaInformation SampleRate	Sample rate 8–96kHz Up to 48 channels	ISO/IEC 13818 A perception-l Binary format
MP3	mp3	Sampled Audio	audio/mpeg audio/mpeg3 audio/x-mpeg-3MP3	"AudioChannels" "CompressionLevel" IncludeMetaInformation SampleRate	Sample rate 8–48kHz Up to 2 channels	MPEG Audio La ISO/IEC 13818 Binary format
OggVorbis	oga ogg	Sampled Audio	audio/ogg audio/vorbis	"AudioChannels" "CompressionLevel" IncludeMetaInformation SampleRate	Sample rate 8–192kHz Supports up to 255 audio channels	Ogg Vorbis dig Uses VBR (vari Binary format
SND	snd	Sampled Audio	audio/basic	"AudioChannels" "AudioEncoding" IncludeMetaInformation SampleRate		By Sun Micros Identical to th Various sample Binary format
WAV	wav	Sampled Audio	audio/x-wav	"AudioChannels" "AudioEncoding" IncludeMetaInformation SampleRate	4GB file size limit	Microsoft WAV Variant of Micr Arbitrary samp Binary format
Wave64	w64	Sampled Audio		"AudioChannels" "AudioEncoding" SampleRate	Overcomes 4GB file size limit of WAV	By Sonic Foun Sony Wave64 Arbitrary samp Binary format

Now we are ready to export to two lossless (WAV, FLAC) and two lossy (MP3, OggVorbis) formats. The “CompressionLevel” option is relevant only for the latter ones.

```
In[908]:= If[runX, sBulkExport[{sine, sineModulated},
            "Extensions" → {"wav", "flac", "mp3", "ogg"}, "CompressionLevel" → 0.95]]

In[909]:= If[runX, sBulkExport[{sine, sineModulated},
            "Extensions" → {"wav", "flac", "mp3", "ogg"}, "CompressionLevel" → 0.1]]
```

Other formats can also be used to save audio:

```
In[910]:= If[runX, sBulkExport[{sine, sineModulated}, "Extensions" → {"mx"},
            "CompressionLevel" → 0.8]]
```

One can export to the PNG format, but observe what output is provided:

```
In[911]:= If[runX, sBulkExport[{sine, sineModulated}]]

In[912]:= SetOptions[sBulkExport, Sequence @@ optssBulkExport];
```

4

Miscellanea

4.1 sHeadsOptionCheck

In[913]:= `infoAbout@sHeadsOptionCheck`

sHeadsOptionCheck[*symbs*] from the sequence of
symbols *symbs*, selects only system symbols implementing the Heads
option with other than default value, and returns the selection as a list.

```
Attributes[sHeadsOptionCheck] = {HoldAll, Protected, ReadProtected}
```

```
SyntaxInformation[sHeadsOptionCheck] = {ArgumentsPattern → {___}}
```

Details

sHeadsOptionCheck[*symbs*] checks whether a non-default value of the Heads option occurs for symbols *symbs* or not. Returns (as a list) a subset (possibly empty) of *symbs* containing those system symbols that implement the Heads option with setting to other than default. Returning empty list signals a negative check result.

- The symbol subset returned is sorted in ascending order.
- *symbs* can be sequence of symbols, numbers or strings, but only symbols are taken into account. Effectively, intersection of *symbs* and the system symbols listed below is processed.
- Other symbol than the following 19 system ones, or any other global symbol, is ignored (valid for both 11.3 and 12.0, default value and versions of introduction/update are shown in parentheses):
  - Activate (True, 10.0/10.0)

- Apply (False, 1.0/10.0)
- Cases (False, 1.0/10.0)
- Count (False, 1.0/10.0)
- DeleteCases (False, 2.0/10.0)
- Depth (False, 1.0/5.0)
- FirstCase (False, 10.0/10.0)
- FirstPosition (True, 10.0/10.0)
- FreeQ (True, 1.0/10.0)
- Inactivate (True, 10.0/10.0)
- Level (False, 1.0/1.0)
- Map (False, 1.0/10.0)
- MapAll (False, 1.0/1.0)
- MapIndexed (False, 2.0/10.0)
- MemberQ (False, 1.0/10.0)
- Position (True, 1.0/10.0)
- Replace (False, 1.0/10.0)
- ReplacePart (Automatic, 2.0/10.0)
- Scan (False, 1.0/10.0)

- Symbols like $MachinePrecision and $Version that produce numbers or strings are ignored.

Examples

The default value of FirstCase and ReplacePart are

```
In[914]:= Options /@ {FirstCase, ReplacePart}
Out[914]= {{Heads → False}, {Heads → Automatic}}
```

Now we change them as follows

```
In[915]:= SetOptions[FirstCase, Heads → True];
          SetOptions[ReplacePart, Heads → False];
```

and detect the change. First, we check all symbols adopting the Heads option,

```
In[916]:= sHeadsOptionCheck[]
Out[916]= {FirstCase, ReplacePart}
```

which is the same as

```
In[917]:= sHeadsOptionCheck[All]
Out[917]= {FirstCase, ReplacePart}
```

We can also select a subset to be checked:

```
In[918]:= sHeadsOptionCheck[Cases, FreeQ, Apply, FirstCase, Activate, ReplacePart]

Out[918]= {FirstCase, ReplacePart}

In[919]:= sHeadsOptionCheck[Cases, FreeQ, Apply, (*FirstCase,*)Activate,
           ReplacePart]

Out[919]= {ReplacePart}
```

But symbols (both system and global) out of the Heads adopting option (see the list above), numbers and strings are ignored. This behavior includes symbols like $MachinePrecision and $Version that evaluate to a number or string:

```
In[920]:= Clear[zz];
          sHeadsOptionCheck[Cases, FreeQ, Apply, FirstCase, Activate, Plot,
           ReplacePart, Print, zz, 1, 22 / 7, 1.2, π, 0.5 + i, MachinePrecision,
           $MachinePrecision, $Version, $VersionNumber]

Out[921]= {FirstCase, ReplacePart}

In[922]:= Clear[zz];
          sHeadsOptionCheck[Cases, FreeQ, Apply, (*FirstCase,*)Activate,
           Plot, ReplacePart, Print, zz, 1, 22 / 7, 1.2, π, 0.5 + i, MachinePrecision,
           $MachinePrecision, $Version, $VersionNumber]

Out[923]= {ReplacePart}
```

Here are all Heads implementing symbols used by Snapdragon code:

```
In[924]:= sHeadsOptionCheck[Apply, Cases, Depth, FirstCase, FirstPosition,
           FreeQ, Map, MapIndexed, MemberQ, Position, Replace, ReplacePart]

Out[924]= {FirstCase, ReplacePart}
```

Changing the Heads option back to default value

```
In[925]:= SetOptions[FirstCase, Heads → False];
          SetOptions[ReplacePart, Heads → Automatic];
```

silences the check:

```
In[926]:= sHeadsOptionCheck[]

Out[926]= {}
```

Applications

This is a straightforward application:

In[927]:=
```
SetOptions[FirstCase, Heads → True];
SetOptions[ReplacePart, Heads → False];
With[{nondfl = sHeadsOptionCheck[]},
 If[nondfl =!= {},
  Print[
   "WARNING: Symbols whose Heads option is globally set to non-defaults:\n",
   Row[nondfl, ","]]]]
SetOptions[FirstCase, Heads → False];
SetOptions[ReplacePart, Heads → Automatic];
(* It is important to make sure to reset to defaults: *)
sHeadsOptionCheck[]
```

```
WARNING: Symbols whose Heads option is globally set to non-defaults:
FirstCase, ReplacePart
```

Out[930]= { }

Discussion

Someone who uses a function defined in Snapdragon has set Heads → True globally by the SetOptions command (which is a bad practice by the way), for Map or Apply, etc., then your program will use that option instead, and consequently will probably not work correctly. Since the author is too procrastinator to rewrite all the shorthand notations like f/@expr, f@@expr, f@@@expr etc. (which are ubiquitous in the Snapdragon code) to explicit Map[f, expr, Heads → False], Apply[f, expr, Heads → False], Apply[f, expr, {1}, Heads → False] etc., sHeadsOptionCheck[] is an easy and quick way how to check whether the user is on the safe side. The code of sHeadsOptionCheck is an exception; it is of course bulletproofed against what it should detect.

4.2 sBoole

In[931]:=
```
infoAbout@sBoole
```

sBoole[*expr*] yields 1 if *expr* is True or 1 and 0 if it is False or 0, yields Null otherwise.

```
Attributes[sBoole] = {Listable, Protected, ReadProtected}
```

```
SyntaxInformation[sBoole] = {ArgumentsPattern → {_}}
```

Details

sBoole[*expr*] yields 1 if *expr* is True or 1 and 0 if it is False or 0. Yields Null otherwise. sBoole is Listable. A simple extension to the Boole builtin.

Examples

While the Boole built-in symbol does not recognize the arguments 0 and 1,

```
In[932]:= Boole[{True, 1, False, 0}]
Out[932]= {1, Boole[1], 0, Boole[0]}
```

the sBoole does:

```
In[933]:= sBoole /@ {True, 1, False, 0}
Out[933]= {1, 1, 0, 0}
```

sBoole is Listable likewise the Boole built-in:

```
In[934]:= sBoole[{True, 1, False, 0}]
Out[934]= {1, 1, 0, 0}
```

Any argument different from the appropriate ones gives Null:

```
In[935]:= sBoole[^] // FullForm
Out[935]//FullForm=
Null
```

Applications

sBoole is applied in some other symbols defined within the Snapdragon application, but occasionally might come handy for general use. It is used, e.g., in the sNextPow2 symbol (see Section 4.10).

4.3 sUniqueName

```
In[936]:= infoAbout@sUniqueName
```

sUniqueName[] gives a unique string made up of the base-36 digits
0…9a…z based on AbsoluteTime, with an interval of change 0.05 seconds.

```
Attributes[sUniqueName] = {Protected, ReadProtected}

SyntaxInformation[sUniqueName] = {ArgumentsPattern → {_., _., _.}}
```

Details

sUniqueName[*unit*] returns a unique string made up of the base-36 digits 0…9a…z based on the AbsoluteTime built-in symbol with a positive real number *unit* defining the interval of change in seconds (default 0.05).

- Based on the way of their construction, generated unique strings are lexicographically sorted by time.
- The smaller *unit*, the longer the unique string.
- For frequency of change 1 / sec, 1 / min, 1 / hr, 1 / day, … set unit to 1., 60., 3600., 86400., …, respectively.
- If *unit* ≤ 0.0, unique string is based on the CreateUUID built-in symbol, but provides unique strings without the 8-4-4-4-12 grouping (no chronological sorting in this case).

sUniqueName[*base*] base *base* (an integer between 2 and 36) is used instead of default 36.

- If *base* == 64, machine-use numeric system "Base64" with 64 "portable" characters A…Za…z0…9-_ is used (deprecated on case-insensitive systems like Windows).

sUniqueName[*pref*] an optional prefix string *pref* is prepended to the unique string.

Any two or all three arguments can be combined, but the order *base*, *unit*, *pref* must be maintained.

Examples

With no argument, default base 36, unit 0.05, and no prefix are used. Change once per 0.05 s = 50 ms:

```
In[937]:= sUniqueName[]
          % // InputForm

Out[937]= ym3ykc4

Out[938]//InputForm=
          "ym3ykc4"
```

Argument *unit* only

With argument *unit* only (positive real number), default base 36, and no prefix is used, and change once per *unit* seconds:

```
In[939]:= sUniqueName[1.]

Out[939]= 1qazxf0

In[940]:= Dynamic[sUniqueName[1.], UpdateInterval → 1.]

Out[940]= 1qazxig
```

Change once per hour and once per day:

```
In[941]:= sUniqueName[3600.]

Out[941]= mfh9

In[942]:= sUniqueName[86 400.]

Out[942]= xn7
```

Specifying *unit* ≤ 0 causes the unique string will be based on the CreateUUID builtin:

```
In[943]:= sUniqueName[0.]
          % // StringLength
          %% // InputForm

Out[943]= 4tmpew0enjqmd5ahq8gafnk9s

Out[944]= 25

Out[945]//InputForm=
          "4tmpew0enjqmd5ahq8gafnk9s"
```

Argument *base* only

With argument *base* only (an integer between 2 and 36), number base *base* is used instead of default 36, while *unit* remains default 0.05:

In the following examples, binary, octal, hexadecimal, and 36 bases are used, respectively (the last one gives the same as sUniqueName[] does):

```
In[946]:= sUniqueName /@ {2, 8, 16, 36} // Column

Out[946]= 10001100010110000111110110110011101111
          1061303733167
          118b0fb677
          ym3ykc7
```

Setting *base* == 64, the machine-use numeric system "Base64" with 64 "portable" characters A…Za…z0…9-_ is used (deprecated on case-insensitive systems like Windows):

```
In[947]:= sUniqueName[64]
          % // InputForm

          sUniqueName: NOTE: Windows-x86-64 is a case-insensitive system.

Out[947]= BGLD7Z3

Out[948]//InputForm=
          "BGLD7Z3"
```

Argument *pref* only

With argument *pref* only (a string), default *base* and *unit* set to 36 and 0.05, respectively, but prefix *pref* is prepended:

```
In[949]:= sUniqueName["HELLO"]

Out[949]= HELLOym3ykc8
```

All whitespace is eliminated:

```
In[950]:= sUniqueName["\nThis Is\nA\tUniqueName - "]

Out[950]= ThisIsAUniqueName-ym3ykc9
```

Combining arguments *base* and *unit* only

For example:

```
In[951]:= sUniqueName[16, 0.1]
Out[951]= 8c587db3c
```

Same as sUniqueName[0.] but in hexadecimal:

```
In[952]:= sUniqueName[16, 0.]
Out[952]= 401c523098064af2859d1b97b1c89db6
```

The last example gives output similar to CreateUUID built-in but without splitting:

```
In[953]:= CreateUUID[]
Out[953]= afa9aa2b-1245-42c9-b36f-d5c9b0582b55
```

Various bases:

```
In[954]:= sUniqueName[8, 0.]
          % // StringLength
Out[954]= 212272524042440550247101654771473245664275 4
Out[955]= 43

In[956]:= sUniqueName[2, 0.]
          % // StringLength
Out[956]= 11101100011010011000011101110101001100100110010100111100110111100101011100 1
             1101010110110010010101111101111101000110001111101 00
Out[957]= 126
```

Machine-use numeric system “Base64” with 64 “portable” chars A…Za…z0…9-_

```
In[958]:= sUniqueName[64, 0.]
          % // StringLength
          %% // InputForm
          sUniqueName: NOTE: Windows-x86-64 is a case-insensitive system.
Out[958]= DXOMe2MTlIyqaNnS3wmv25
Out[959]= 22
Out[960]//InputForm=
          "DXOMe2MTlIyqaNnS3wmv25"
```

Combining arguments *base* and *pref* only

```
In[961]:= sUniqueName @@@ {{2, "binary-"}, {8, "octal -"}, {16, "hexadecimal_"},
            {36, "Base36-"}, {64, "Base64-"}} // Column

          sUniqueName: NOTE: Windows-x86-64 is a case-insensitive system.

          binary-1000110001011000011111011011001111100
          octal-1061303733174
Out[961]= hexadecimal_118b0fb67c
          Base36-ym3ykcc
          Base64-BGLD7Z8
```

The “Base36” example is similar to sUniqueName[].

Combining arguments *unit* and *pref* only

Similar to sUniqueName[]:

```
In[962]:= With[{u = 0.05}, sUniqueName[u, ToString[u] <> "sec-"]]

Out[962]= 0.05sec-ym3ykcd
```

Change once per second, per 10 seconds, once per minute, once per hour:

```
In[963]:= sUniqueName[#, ToString[#] <> "sec-"] & /@ {1., 10., 60., 3600.} //
           Column

          1.sec-1qazxf0
Out[963]= 10.sec-68asji
          60.sec-11dsr9
          3600.sec-mfh9
```

Based on the CreateUUID builtin (*unit* ≤ 0.0):

```
In[964]:= With[{u = 0.}, sUniqueName[u, ToString[u] <> "s-"]]

Out[964]= 0.s-4yxotnx0s4l4bknhx24ut1pfb
```

Combining all three arguments

Finally, we can combine all three arguments; there are no limits for imagination:

```
In[965]:= sUniqueName[#1, #2, "base" <> ToString[#1] <> "_unit" <> ToString[#2] <>
              "sec_"] &@@@ {{36, 0.05}, {16, 10.}, {8, 60.}, {4, 3600.}, {2, 86400.}} //
           Column

          base36_unit0.05sec_ym3ykce
          base16_unit10.sec_16748efe
Out[965]= base8_unit60.sec_357413725
          base4_unit3600.sec_3333133131
          base2_unit86400.sec_1010101001010011
```

Based on the CreateUUID builtin (*unit* ≤ 0.0):

In[966]:=
```
With[{b = 8, u = 0.},
 sUniqueName[b, u, "base" <> ToString[b] <> "_unit" <> ToString[u] <> "sec_"]]
% // InputForm
```

Out[966]=
```
base8_unit0.sec_257226606603524111373326543546721355132325 6
```

Out[967]//InputForm=
```
"base8_unit0.sec_257226606603524111373326543546721355132325 6"
```

Keep in mind that the order base, unit, pref must be obeyed. The same applies in case of using two arguments only as described above.

In[968]:=
```
sUniqueName[16, 3600., "hour-"]
```

Out[968]=
```
hour-ff7dd
```

Applications

sUniqueName is used in the definition of the symbol sBulkExport (see Section 3.3) to create a uniquely named directory when exporting from and unsaved notebook; in such case the export is saved to directory $TemporaryDirectory/sBulkExport/sUniqueName[]. However, it may facilitate many other function, e.g., those connected with compiling.

Discussion

4.4 sReferTo

In[969]:=
```
infoAbout@sReferTo
```

sReferTo[*symb*] returns hyperlink to the symbol *symb* in the WLSDC "ref" category.
sReferTo[*symb*, *cat*] the same but for *symb* in the *cat* category "catname".
sReferTo[*symb*, *grp*] the same but for *symb* in the *grp* group {"grpname"}.

```
Attributes[sReferTo] = {HoldFirst, Protected, ReadProtected}

Options[sReferTo] =
 {Website → False, Active → True, Context → System, Debug → False}

SyntaxInformation[sReferTo] =
 {ArgumentsPattern → {_, _., _., OptionsPattern[]}}
```

Details

sReferTo[*symb*, *cat*, *grp*, *opts*] returns documentation hyperlink labeled by the symbol *symb* contained

- in a *Wolfram Language* & System Documentation Center (WLSDC) category "ref" or "" (default), "guide", "tutorial" given by the string *cat*,
- in a WLSDC group {} (default), {"format"}, {"character"}, {"message"}, {"menuitem"} given by the list *grp*.
- A context or package can be specified with the option "Context" (the default value "System" limits to Wolfram System built-in symbols).
- The option "Website" → False (default) uses local WLSDC, "Website" → True redirects to Wolfram reference website.
- The option "Active" → False makes a plain output {"*label*", "uri"} to be returned instead of an active hyperlink coerced with the default "Active" → True.
- Any other options are passed to the Hyperlink function.
- Reference to detailed information about the General::plln message and like are achievable through giving two-element group, e.g. specifying group as {"message", "plln"} creates reference to ref/message/General/plln.

Examples

Keep in mind that the option keys pertinent to the Hyperlink builtin (typically ActiveStyle) are highlighted by SyntaxInformation but working anyway.

Built-in symbols

The simplest task is creating an active link to a symbol in the local WLSDC. Clicking the output will open the WLSDC window directly for the appropriate symbol:

```
In[970]:= sReferTo[List]
```

```
Out[970]= List
```

Here we only change the color when the mouse cursor is moved over the output:

```
In[971]:= sReferTo[List, ActiveStyle → Green]
```

```
Out[971]= List
```

This makes the link to online WLSDC:

```
In[972]:= sReferTo[List, "Website" → True]
```

```
Out[972]= List
```

When the "Active" option is set to False (case insensitive), a nonactive output suitable for pasting to hyperlinks etc. is returned. Remember that the elements of the list are strings:

```
In[973]:= sReferTo[List, "Active" → False]
```

```
Out[973]= {List, paclet:ref/List}
```

```
In[974]:= sReferTo[List, "Website" → True, "Active" → False]

Out[974]= {List, http://reference.wolfram.com/language/ref/List}
```

Further examples:

```
In[975]:= sReferTo[Integrate]

Out[975]= Integrate

In[976]:= sReferTo[$BaseDirectory]

Out[976]= $BaseDirectory
```

Packages and applications

This is how to make a reference to a package, Combinatorica in this case:

```
In[977]:= sReferTo[Algorithm, "Context" → "Combinatorica"]

Out[977]= Algorithm

In[978]:= sReferTo[Algorithm, "Context" → "Combinatorica", "Website" → True,
           "Active" → False]

Out[978]= {Algorithm,
           http://reference.wolfram.com/language/Combinatorica/ref/Algorithm}

In[979]:= sReferTo[MakeContents, "Context" → "AuthorTools"]

Out[979]= MakeContents

In[980]:= sReferTo[MakeContents, "Context" → "AuthorTools", "Website" → True]

Out[980]= MakeContents
```

Guides

When referencing to a guide, this would not work:

```
In[981]:= sReferTo[ListManipulation] (* could not be found *)

Out[981]= ListManipulation
```

Instead, we have to use the syntax as follows:

```
In[982]:= sReferTo[ListManipulation, "guide"]

Out[982]= ListManipulation
```

```
In[983]:= sReferTo[ListManipulation, "guide", {}]
Out[983]= ListManipulation

In[984]:= sReferTo[ListManipulation, "guide", "Website" → True]
Out[984]= ListManipulation

In[985]:= sReferTo[ListManipulation, "guide", "Website" → True, "Active" → False]
Out[985]= {ListManipulation,
           http://reference.wolfram.com/language/guide/ListManipulation}
```

Other examples of guide referencing (including package guides):

```
In[986]:= sReferTo[Calculus, "guide"]
Out[986]= Calculus

In[987]:= sReferTo[Calculus, "guide", "Website" → True, "Active" → False]
Out[987]= {Calculus, http://reference.wolfram.com/language/guide/Calculus}

In[988]:= sReferTo[MathematicalTypesetting, "guide"]
Out[988]= MathematicalTypesetting

In[989]:= sReferTo[Permutations, "guide", "Context" → "Combinatorica"]
Out[989]= Permutations

In[990]:= sReferTo[AuthorTools, "guide", "Context" → "AuthorTools"]
Out[990]= AuthorTools
```

Tutorials

When referencing to a tutorial, this would not work:

```
In[991]:= sReferTo[MakingListsOfObjects] (* could not be found *)
Out[991]= MakingListsOfObjects
```

Instead, we have to use the syntax by the following examples:

```
In[992]:= sReferTo[MakingListsOfObjects, "tutorial"]
Out[992]= MakingListsOfObjects
```

```
In[993]:= sReferTo[MakingListsOfObjects, "tutorial", {}]
Out[993]= MakingListsOfObjects

In[994]:= sReferTo[MakingListsOfObjects, "tutorial", "Website" → True]
Out[994]= MakingListsOfObjects
```

Further examples of tutorial referencing (including package tutorials):

```
In[995]:= sReferTo[TwoDimensionalExpressionInputOverview, "tutorial"]
Out[995]= TwoDimensionalExpressionInputOverview

In[996]:= sReferTo[IndefiniteIntegrals, "tutorial"]
Out[996]= IndefiniteIntegrals

In[997]:= sReferTo[Combinatorica, "tutorial", "Context" → "Combinatorica"]
Out[997]= Combinatorica

In[998]:= sReferTo[Combinatorica, "tutorial", "Context" → "Combinatorica",
           "Website" → True, "Active" → False]
Out[998]= {Combinatorica,
           http://reference.wolfram.com/language/Combinatorica/tutorial/Combinatorica}

In[999]:= sReferTo[MakeContents, "tutorial", "Context" → "AuthorTools"]
Out[999]= MakeContents
```

Formats

The following command works,

```
In[1000]:= sReferTo[PDF]
Out[1000]= PDF
```

but proably refers to something completely different (Probability Density Function) than you want. The third argument has to be use to create output aiming at the right target (PDF format). Note that the second argument may be omitted:

```
In[1001]:= sReferTo[PDF, "ref", {"format"}]
Out[1001]= PDF
```

```
In[1002]:= sReferTo[PDF, "ref", {"format"}, "Website" → False, "Active" → False]
Out[1002]= {PDF, paclet:ref/format/PDF}

In[1003]:= sReferTo[PDF, {"format"}]
Out[1003]= PDF

In[1004]:= sReferTo[PDF, {"format"}, "Website" → True]
Out[1004]= PDF

In[1005]:= sReferTo[PDF, {"format"}, "Website" → True, "Active" → False]
Out[1005]= {PDF, http://reference.wolfram.com/language/ref/format/PDF}
```

Some further examples of format referencing:

```
In[1006]:= sReferTo[#, {"format"}] & /@ {AIFF, GIF, OggVorbis} // Row[#, Spacer@10] &
Out[1006]= AIFF  GIF  OggVorbis
```

All formats *Mathematica* is aware:

```
In[1007]:= sReferTo[ListingOfAllFormats, "guide"]
Out[1007]= ListingOfAllFormats
```

Here is a tutorial of importing and exporting data:

```
In[1008]:= sReferTo[ImportingAndExportingData, "tutorial"]
Out[1008]= ImportingAndExportingData
```

Characters and typesetting

For example, detailed information about the → character can be obtained using

```
In[1009]:= sReferTo[Rule, {"character"}]
Out[1009]= Rule

In[1010]:= sReferTo[Rule, {"character"}, "Website" → True, "Active" → False]
Out[1010]= {Rule, http://reference.wolfram.com/language/ref/character/Rule}
```

Compare to the following code; it works but probably takes you different place than you intended:

```
In[1011]:= sReferTo[Rule]
Out[1011]= Rule
```

Messages

Detailed information about the General::plln message:

```
In[1012]:= sReferTo[General, {"message", "plln"}]
Out[1012]= General
```

Guide and tutorial for messages can be accessed as follows:

```
In[1013]:= sReferTo[Messages, "guide"]
Out[1013]= Messages

In[1014]:= sReferTo[Messages, "tutorial"]
Out[1014]= Messages
```

But compare the above outputs to

```
In[1015]:= sReferTo[Messages]
Out[1015]= Messages
```

Menu items

An explanation of the File ▷ New menu command:

```
In[1016]:= sReferTo[New, {"menuitem"}]
Out[1016]= New

In[1017]:= sReferTo[New, {"menuitem"}, "Website" → True, "Active" → False]
Out[1017]= {New, http://reference.wolfram.com/language/ref/menuitem/New}
```

A list of the File menu items:

```
In[1018]:= sReferTo[FileMenu, "guide"]
Out[1018]= FileMenu

In[1019]:= sReferTo[FileMenu, "guide", "Website" → True, "Active" → False]
Out[1019]= {FileMenu, http://reference.wolfram.com/language/guide/FileMenu}
```

Applications

sReferTo makes it very easy to create links to multitude of items in the *Wolfram Language* & System Documentation Center both local (offline) and online (on the web). For example, here is one way of referring to whole the Cell ▷ Cell Properties ▷ Evaluatable menu sequence:

```
In[1020]:= {sReferTo[##], sReferTo[##, "Website" → True]} &[CellMenu, "guide",
            "Active" → False]

Out[1020]= {{CellMenu, paclet:guide/CellMenu},
            {CellMenu, http://reference.wolfram.com/language/guide/CellMenu}}

In[1021]:= {sReferTo[##], sReferTo[##, "Website" → True]} &[CellProperties,
            {"menuitem"}, "Active" → False]

Out[1021]= {{CellProperties, paclet:ref/menuitem/CellProperties}, {CellProperties,
             http://reference.wolfram.com/language/ref/menuitem/CellProperties}}

In[1022]:= {sReferTo[##], sReferTo[##, "Website" → True]} &[Evaluatable,
            {"menuitem"}, "Active" → False]

Out[1022]= {{Evaluatable, paclet:ref/menuitem/Evaluatable}, {Evaluatable,
             http://reference.wolfram.com/language/ref/menuitem/Evaluatable}}
```

Discussion

Suppose we want to make references both to the WLSDC and online help. This can be accomplished as follows:

```
In[1023]:= {sReferTo[#, "Active" → False],
              sReferTo[#, "Website" → True, "Active" → False]} &[List]

Out[1023]= {{List, paclet:ref/List},
            {List, http://reference.wolfram.com/language/ref/List}}
```

However, applying the same solution on something different than List (Infinity in this case) fails,

```
In[1024]:= {sReferTo[#, "Active" → False],
              sReferTo[#, "Website" → True, "Active" → False]} &[Infinity]

           ▪▪▪ Snapdragon: sReferTo[∞, Active → False] does not match the rule base.

           ▪▪▪ Snapdragon: sReferTo[∞, Website → True, Active → False] does not match the rule base.

Out[1024]= {$Failed, $Failed}
```

although the plain function calls do work:

```
In[1025]:= {sReferTo[Infinity, "Active" → False],
            sReferTo[Infinity, "Website" → True, "Active" → False]}

Out[1025]= {{Infinity, paclet:ref/Infinity},
            {Infinity, http://reference.wolfram.com/language/ref/Infinity}}
```

We can however overcome the problem like this:

In[1026]:=
```
{sReferTo[#, "Active" → False],
   sReferTo[#, "Website" → True, "Active" → False]} &[Unevaluated@Infinity]
```

Out[1026]=
```
{{Infinity, paclet:ref/Infinity},
 {Infinity, http://reference.wolfram.com/language/ref/Infinity}}
```

4.5 sPushKeyUp

In[1027]:=
```
infoAbout@sPushKeyUp
```

sPushKeyUp[*assoc*, *xckey*, *k*] duplicates row keys of the
"named columns and named rows" association *assoc* as respective
values of extra inserted *k*-th column (default 1st) with key string *xckey*.

```
Attributes[sPushKeyUp] = {Protected, ReadProtected}
```

```
SyntaxInformation[sPushKeyUp] = {ArgumentsPattern → {_, _, _.}}
```

Details

sPushKeyUp[*assoc*, *xckey*, *k*] duplicates primary (row) keys of a "full table with named columns and named rows" association *assoc* as respective values of extra inserted *k*-th column (integer *k* is 1 by default) specified by a key string *xckey*.

- The type of associations mentioned above is also known as "association of associations" or "indexed table".
- Observe the following example:

  sPushKeyUp[<|"a" → <|"x" → 11, "y" → 12|>, "b" → <|"x" → 21, "y" → 22|>|>, "new"]

  yields

  <|"a" → <|"new" → "a", "x" → 11, "y" → 12|>,
  "b" → <|"new" → "b", "x" → 21, "y" → 22|>|>

- The idiom

  Values@sPushKeyUp[…]

  effectively transforms assoc to a "full table with named columns (and anonymous rows)" a.k.a. "list of associations" without loss of information. The "array fullness" of assoc is not checked for. For example, this is how the above example will end up:

  {<|"new" → "a", "x" → 11, "y" → 12|>, "b" → <|"new" → "b", "x" → 21, "y" → 22|>}

- The column number *k* follows part specification rules (|*k*| cannot be 0 or greater than the length of *assoc*).

Examples

Clearing global symbols:

In[1028]:=
```
Clear[assoc, assoc3];
```

For exemplifying this function, we define two “full table with named columns and named rows” associations, one 2D, the other 3D:

In[1029]:=
```
assoc = <|"a" → <|"x" → 11, "y" → 12|>, "b" → <|"x" → 21, "y" → 22|>,
   "c" → <|"x" → 31, "y" → 32|>|>
assoc // Dataset // Magnify[#, 2 / 3] &
assoc // ArrayDepth
```

Out[1029]=
```
<|a → <|x → 11, y → 12|>, b → <|x → 21, y → 22|>, c → <|x → 31, y → 32|>|>
```

Out[1030]=

	x	y
a	11	12
b	21	22
c	31	32

Out[1031]= 2

In[1032]:=
```
assoc3 =
 <|"a" → <|"x" → <|"p" → 111, "q" → 112, "r" → 113, "s" → 114|>,
    "y" → <|"p" → 121, "q" → 122, "r" → 123, "s" → 124|>|>,
  "b" → <|"x" → <|"p" → 211, "q" → 212, "r" → 213, "s" → 214|>,
    "y" → <|"p" → 221, "q" → 222, "r" → 223, "s" → 224|>|>,
  "c" → <|"x" → <|"p" → 311, "q" → 312, "r" → 313, "s" → 314|>,
    "y" → <|"p" → 321, "q" → 322, "r" → 323, "s" → 324|>|>|>
assoc3 // Dataset // Magnify[#, 2 / 3] &
assoc3 // ArrayDepth
```

Out[1032]=
```
<|a → <|x → <|p → 111, q → 112, r → 113, s → 114|>,
    y → <|p → 121, q → 122, r → 123, s → 124|>|>,
 b → <|x → <|p → 211, q → 212, r → 213, s → 214|>,
    y → <|p → 221, q → 222, r → 223, s → 224|>|>,
 c → <|x → <|p → 311, q → 312, r → 313, s → 314|>,
    y → <|p → 321, q → 322, r → 323, s → 324|>|>|>
```

Out[1033]=

	x				y			
	p	q	r	s	p	q	r	s
a	111	112	113	114	121	122	123	124
b	211	212	213	214	221	222	223	224
c	311	312	313	314	321	322	323	324

Out[1034]= 3

First we simply apply the sPushKeyUp function. We observe duplication (the column with the "extra" heading):

```
In[1035]:= sPushKeyUp[assoc, "extra"]
           % // Dataset // Magnify[#, 2 / 3] &
```

```
Out[1035]= ⟨| a → ⟨| extra → a, x → 11, y → 12 |⟩,
             b → ⟨| extra → b, x → 21, y → 22 |⟩, c → ⟨| extra → c, x → 31, y → 32 |⟩ |⟩
```

Out[1036]=

	extra	x	y
a	a	11	12
b	b	21	22
c	c	31	32

```
In[1037]:= sPushKeyUp[assoc3, "extra"]
           % // Dataset // Magnify[#, 2 / 3] &
```

```
Out[1037]= ⟨| a → ⟨| extra → a, x → ⟨| p → 111, q → 112, r → 113, s → 114 |⟩,
                 y → ⟨| p → 121, q → 122, r → 123, s → 124 |⟩ |⟩,
             b → ⟨| extra → b, x → ⟨| p → 211, q → 212, r → 213, s → 214 |⟩,
                 y → ⟨| p → 221, q → 222, r → 223, s → 224 |⟩ |⟩,
             c → ⟨| extra → c, x → ⟨| p → 311, q → 312, r → 313, s → 314 |⟩,
                 y → ⟨| p → 321, q → 322, r → 323, s → 324 |⟩ |⟩ |⟩
```

Out[1038]=

	extra	x				y			
		p	q	r	s	p	q	r	s
a	a	111	112	113	114	121	122	123	124
b	b	211	212	213	214	221	222	223	224
c	c	311	312	313	314	321	322	323	324

The position of the extra inserted column can easily be changed:

```
In[1039]:= sPushKeyUp[assoc, "extra", 2]
           % // Dataset // Magnify[#, 2 / 3] &
```

```
Out[1039]= ⟨| a → ⟨| x → 11, extra → a, y → 12 |⟩,
             b → ⟨| x → 21, extra → b, y → 22 |⟩, c → ⟨| x → 31, extra → c, y → 32 |⟩ |⟩
```

Out[1040]=

	x	extra	y
a	11	a	12
b	21	b	22
c	31	c	32

In[1041]:=
```
sPushKeyUp[assoc3, "extra", 2]
% // Dataset // Magnify[#, 2 / 3] &
```

Out[1041]=
```
⟨| a → ⟨| x → ⟨| p → 111, q → 112, r → 113, s → 114 |⟩,
    extra → a, y → ⟨| p → 121, q → 122, r → 123, s → 124 |⟩ |⟩,
 b → ⟨| x → ⟨| p → 211, q → 212, r → 213, s → 214 |⟩, extra → b,
    y → ⟨| p → 221, q → 222, r → 223, s → 224 |⟩ |⟩,
 c → ⟨| x → ⟨| p → 311, q → 312, r → 313, s → 314 |⟩, extra → c,
    y → ⟨| p → 321, q → 322, r → 323, s → 324 |⟩ |⟩ |⟩
```

Out[1042]=

	x				extra	y			
	p	q	r	s		p	q	r	s
a	111	112	113	114	a	121	122	123	124
b	211	212	213	214	b	221	222	223	224
c	311	312	313	314	c	321	322	323	324

In[1043]:=
```
sPushKeyUp[assoc, "extra", -1]
% // Dataset // Magnify[#, 2 / 3] &
```

Out[1043]=
```
⟨| a → ⟨| x → 11, y → 12, extra → a |⟩,
 b → ⟨| x → 21, y → 22, extra → b |⟩, c → ⟨| x → 31, y → 32, extra → c |⟩ |⟩
```

Out[1044]=

	x	y	extra
a	11	12	a
b	21	22	b
c	31	32	c

In[1045]:=
```
sPushKeyUp[assoc3, "extra", -1]
% // Dataset // Magnify[#, 2 / 3] &
```

Out[1045]=
```
⟨| a → ⟨| x → ⟨| p → 111, q → 112, r → 113, s → 114 |⟩,
    y → ⟨| p → 121, q → 122, r → 123, s → 124 |⟩, extra → a |⟩,
 b → ⟨| x → ⟨| p → 211, q → 212, r → 213, s → 214 |⟩,
    y → ⟨| p → 221, q → 222, r → 223, s → 224 |⟩, extra → b |⟩,
 c → ⟨| x → ⟨| p → 311, q → 312, r → 313, s → 314 |⟩,
    y → ⟨| p → 321, q → 322, r → 323, s → 324 |⟩, extra → c |⟩ |⟩
```

Out[1046]=

	x				y				extra
	p	q	r	s	p	q	r	s	
a	111	112	113	114	121	122	123	124	a
b	211	212	213	214	221	222	223	224	b
c	311	312	313	314	321	322	323	324	c

Now we transform the “full table with named columns and named rows” or “association of

associations" into "a table with named columns (and anonymous rows)" or "list of associations" without loss of information using the Values idiom:

In[1047]:=
```
Values@sPushKeyUp[assoc, "extra"]
% // Dataset // Magnify[#, 2 / 3] &
```

Out[1047]=
```
{<|extra → a, x → 11, y → 12|>,
 <|extra → b, x → 21, y → 22|>, <|extra → c, x → 31, y → 32|>}
```

Out[1048]=

extra	x	y
a	11	12
b	21	22
c	31	32

In[1049]:=
```
Values@sPushKeyUp[assoc, "extra", 2]
% // Dataset // Magnify[#, 2 / 3] &
```

Out[1049]=
```
{<|x → 11, extra → a, y → 12|>,
 <|x → 21, extra → b, y → 22|>, <|x → 31, extra → c, y → 32|>}
```

Out[1050]=

x	extra	y
11	a	12
21	b	22
31	c	32

In[1051]:=
```
Values@sPushKeyUp[assoc3, "extra"]
% // Dataset // Magnify[#, 2 / 3] &
```

Out[1051]=
```
{<|extra → a, x → <|p → 111, q → 112, r → 113, s → 114|>,
   y → <|p → 121, q → 122, r → 123, s → 124|>|>,
  <|extra → b, x → <|p → 211, q → 212, r → 213, s → 214|>,
   y → <|p → 221, q → 222, r → 223, s → 224|>|>,
  <|extra → c, x → <|p → 311, q → 312, r → 313, s → 314|>,
   y → <|p → 321, q → 322, r → 323, s → 324|>|>}
```

Out[1052]=

extra	x				y			
	p	q	r	s	p	q	r	s
a	111	112	113	114	121	122	123	124
b	211	212	213	214	221	222	223	224
c	311	312	313	314	321	322	323	324

In[1053]:=
```
Values@sPushKeyUp[assoc3, "extra", -1]
% // Dataset // Magnify[#, 2 / 3] &
```

Out[1053]=
```
{ ⟨| x → ⟨| p → 111, q → 112, r → 113, s → 114 |⟩,
    y → ⟨| p → 121, q → 122, r → 123, s → 124 |⟩, extra → a |⟩,
  ⟨| x → ⟨| p → 211, q → 212, r → 213, s → 214 |⟩,
    y → ⟨| p → 221, q → 222, r → 223, s → 224 |⟩, extra → b |⟩,
  ⟨| x → ⟨| p → 311, q → 312, r → 313, s → 314 |⟩,
    y → ⟨| p → 321, q → 322, r → 323, s → 324 |⟩, extra → c |⟩ }
```

Out[1054]=

x				y				extra
p	q	r	s	p	q	r	s	
111	112	113	114	121	122	123	124	a
211	212	213	214	221	222	223	224	b
311	312	313	314	321	322	323	324	c

Applications

In the original form, sPushKeyUp has been used for manipulation with an extensive biomedical research database. Hopefully it can beneficially be used in similar circumstances.

4.6 sEmptyListsQ

In[1055]:=
```
infoAbout@sEmptyListsQ
```

sEmptyListsQ[*expr*] gives True if *expr* is an arbitrary tree
of lists with leaves occupied with empty lists only, False otherwise.

```
Attributes[sEmptyListsQ] = {Protected, ReadProtected}
```

```
SyntaxInformation[sEmptyListsQ] = {ArgumentsPattern → {_}}
```

Details

sEmptyListsQ[*expr*] gives True if *expr* is an empty list or an arbitrary tree of lists with the deepest levels (leaves) occupied with empty lists only, and gives False otherwise (even for an unmatched argument pattern).

- Useful for the OptionsPattern quirk workaround.

An immediate comment on this function is due here: the only application of sEmptyListsQ is in rectifying the "strange" behavior of the OptionsPattern that is, however, well documented. The application in removing the OptionsPattern quirk is described further on in the Discussion below. Sadly, this function is likely completely useless for anything else (maybe except for teaching purposes).

Examples

Clearing global symbols:

```
In[1056]:= Clear[empty, nonempty, f, z];
           ClearAll[fun];
```

False is returned for any expression that contains lists on all tree levels:

```
In[1058]:= sEmptyListsQ /@ {1, 1.0, "A string", {1}, {10, 11}}

Out[1058]= {False, False, False, False, False}
```

True is returned for an empty list or an arbitrary tree of lists with the deepest levels occupied with empty lists only:

```
In[1059]:= empty = {{}, {{}}, {{{}, {{}}}}, {}, {{{}, {{}}}}};
           sEmptyListsQ /@ empty

Out[1060]= {True, True, True}
```

Flattening any of these elements ultimately end up as an empty list {}:

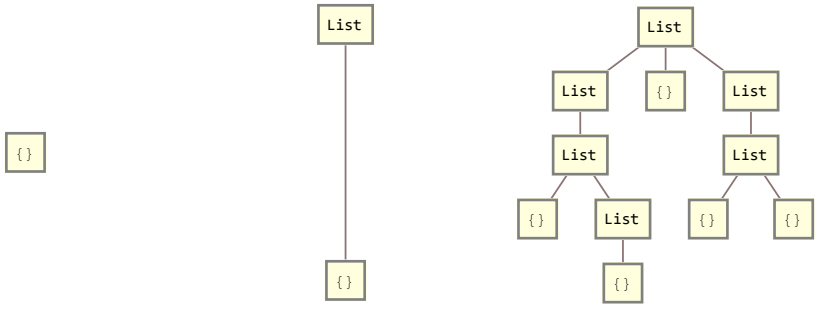
```
In[1061]:= Flatten /@ empty

Out[1061]= {{}, {}, {}}
```

Tree form of the above “empty” expressions:

```
In[1062]:= TreeForm[#, AspectRatio → 1, ImageSize → Small] & /@ empty // Row //
            Magnify[#, 2 / 3] &
```

Out[1062]=



Had any list be nonempty, False is returned:

```
In[1063]:= nonempty = {{{}, {{"a"}}}, {{{{}, {{}}}}}, {}, 0, {{}}},
              {{0}, {{}}, {{1}}, {{{}, {}}}}};
           sEmptyListsQ /@ nonempty

Out[1064]= {False, False, False}
```

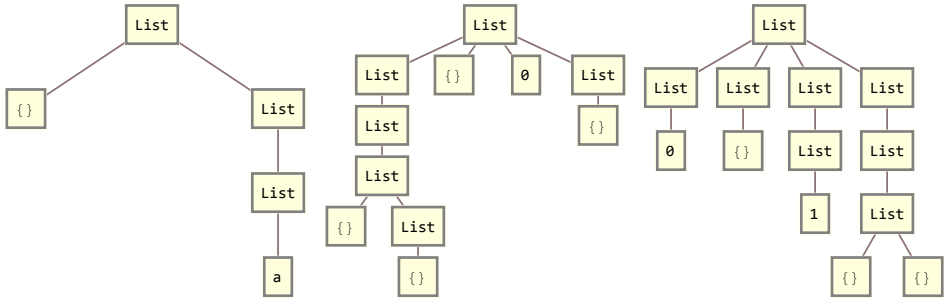
Flattening any of these elements ultimately end up as an nonempty list:

```
In[1065]:= Flatten /@ nonempty
Out[1065]= {{a}, {0}, {0, 1}}
```

Tree form of the above “nonempty” expressions:

```
In[1066]:= TreeForm[#, AspectRatio → 1, ImageSize → Small] & /@ nonempty // Row //
            Magnify[#, 2 / 3] &
```

Out[1066]=



False is returned in all other cases such as no argument, too many arguments, or incorrect type of argument are as follows. This is advantageous when using in function code design as described in the Discussion.

```
In[1067]:= sEmptyListsQ[]
Out[1067]= False

In[1068]:= sEmptyListsQ[{}, 2]
Out[1068]= False

In[1069]:= sEmptyListsQ[{{}, {}}, "string"]
Out[1069]= False

In[1070]:= sEmptyListsQ[{}, {{}, {}}]
Out[1070]= False
```

Applications

Workaround for any nested empty lists is as follows:

```
If[And[Length[{opts}] > 0, sEmptyListsQ[First[{opts}]]],
  Message[someFunction::optbug]; Return[$Failed]];
```

An example of a complete solution can be found at the end of Discussion.

Discussion

In the following we’ll described a peculiar behavior of OptionsPattern. Imagine that you have to define a function that accepts one integer argument with a default value, and one option. At this

moment it is not essential what the function yields, but rather how foolproof it is, i.e. when an unappropriate argument is given, an error message is issued and the function execution is quitted. Suppose that the function returns a constant vector of the length specified by the integer argument (default 2) and the element specified by the option (default 1). (Normally, one would use Table or ConstantArray for this purpose.) The function might look something like this.

```
In[1071]:= fun::badarg = "`1` argument given, Integer expected."(* error message *);
           Options[fun] = {"Elements" → 1}(* default optional element *);
           (* main function body: *)
           fun[n_Integer: 2, opts : OptionsPattern[]] :=
             Module[{elem}, {elem} = OptionValue[Keys@Options[fun]];
              ConstantArray[elem, Abs@n]];
           (* drop down here if n is not an integer: *)
           fun[n_, ___] := (Message[fun::badarg, Head[n]];
             $Failed)
```

Now let's test it. Omitting any argument, it yields what one expects (remember that default number of elements is 2).

```
In[1075]:= fun[]
Out[1075]= {1, 1}
```

It returns expected results with any other integer argument.

```
In[1076]:= fun[8]
Out[1076]= {1, 1, 1, 1, 1, 1, 1, 1}

In[1077]:= fun /@ {0, 1, 4, -4}
Out[1077]= {{}, {1}, {1, 1, 1, 1}, {1, 1, 1, 1}}
```

Also, the option works as expected.

```
In[1078]:= fun[8, "Elements" → -1]
Out[1078]= {-1, -1, -1, -1, -1, -1, -1, -1}
```

So far so good. Now a mischievous guy (a.k.a. BFU) comes and supplies a noninteger argument. But the function is safely defined, refusing all his nasty attempts, because any noninteger argument does not match the `fun[n_Integer:2,opts:OptionsPattern[]]` and drops down to `fun[n_,___]`.

```
In[1079]:= fun[1.0]
           ▪▪▪ fun: Real argument given, Integer expected.
Out[1079]= $Failed
```

```
In[1080]:= fun /@ {"hello", z, 4 / 5, 1 + i, #^2 &, Composition[Exp, Cos]}

          ••• fun: String argument given, Integer expected.

          ••• fun: Symbol argument given, Integer expected.

          ••• fun: Rational argument given, Integer expected.

          ••• General: Further output of fun::badarg will be suppressed during this calculation.

Out[1080]= {$Failed, $Failed, $Failed, $Failed, $Failed, $Failed}
```

Also a list ends up with an error, as one could anticipate.

```
In[1081]:= fun[{4}]

          ••• fun: List argument given, Integer expected.

Out[1081]= $Failed
```

```
In[1082]:= fun[Range[4], "Elements" → -1]

          ••• fun: List argument given, Integer expected.

Out[1082]= $Failed
```

So we expect the same behavior for the empty list. But

```
In[1083]:= fun[{}]

Out[1083]= {1, 1}
```

```
In[1084]:= fun[{}, "Elements" → -1]

Out[1084]= {-1, -1}
```

What the hell happened? Hint: read the *Possible Issues* section of the OptionsPattern documentation, where the very end says this: Any nesting of empty lists will match OptionsPattern:

```
In[1085]:= MatchQ[f[{}], f[OptionsPattern[]]]

Out[1085]= True
```

```
In[1086]:= MatchQ[f[{{}}], f[OptionsPattern[]]]

Out[1086]= True
```

So we see that the function OptionsPattern "gobbles" the empty list `{}`, leaving the argument sequence empty, allowing the default value take place! So it's not a bug, but a feature! But anyway, let's try to suggest a way to get the function `fun` to issue an error message even when you enter an empty list. Here is a possible solution:

```
In[1087]:= ClearAll[fun];
           fun::badarg = "`1` argument given, Integer expected.";
           fun::optbug = "OptionsPattern \"gobble\" occurred."(* new message *);
           Options[fun] = {"Elements" → 1};
           fun[n_Integer: 2, opts : OptionsPattern[]] :=
             Module[{elem},
              (* begin workaround *)
              If[And[Length[{opts}] > 0, sEmptyListsQ[First[{opts}]]],
               Message[fun::optbug]; Return[$Failed]];
              (* end workaround *)
              {elem} = OptionValue[Keys@Options[fun]];
              ConstantArray[elem, Abs@n]];
           fun[n_, ___] := (Message[fun::badarg, Head[n]];
             $Failed)
```

Now the last two examples will obediently issue an error message instead of weird behavior,

```
In[1093]:= fun[{}]
           fun: OptionsPattern "gobble" occurred.
Out[1093]= $Failed

In[1094]:= fun[{}, "Elements" → -1]
           fun: OptionsPattern "gobble" occurred.
Out[1094]= $Failed

In[1095]:= fun[{{}, {{}}}]
           fun: OptionsPattern "gobble" occurred.
Out[1095]= $Failed
```

and you can make sure that the behavior with a “normal” argument remained identical.

4.7 sFactorExactNumber

```
In[1096]:= infoAbout@sFactorExactNumber
```

sFactorExactNumber[*n*] like FactorInteger,
but repeats each prime factor as many times as its exponent.

```
Attributes[sFactorExactNumber] = {Listable, Protected, ReadProtected}

Options[sFactorExactNumber] = {Debug → False}

SyntaxInformation[sFactorExactNumber] =
 {ArgumentsPattern → {_, _., OptionsPattern[]}}
```

Details

sFactorExactNumber[*n*, *k*, *opts*] gives the list of prime factors $\{p_1, \ldots, p_1, p_2, \ldots, p_2, \ldots, p_m, \ldots, p_m\}$ of an exact number n with each factor p_j occurring exactly e_j times, $j \in \{1, \ldots, m\}$.

- n can be integer, rational, Gaussian integer or rational.
- The output is expanded and flattened from $\{\{p_1, e_1\}, \ldots, \{p_m, e_m\}\}$ provided by the FactorInteger built-in symbol.
- All arguments, options and properties are inherited from the FactorInteger built-in symbol.

Examples

In the examples below, we first factor a number using the FactorInteger built-in symbol, then using sFactorExactNumber, finally applying Tally built-in on the latter, comparing it to the FactorInteger output (True result is desirable):

```
In[1097]:= Block[{n = 0, fi, fen},
            Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n],
              fi === Tally[fen]}]

Out[1097]= {{0, 1}}
           {0}
           True
```

```
In[1098]:= Block[{n = 2434500, fi, fen},
            Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n],
              fi === Tally[fen]}]

Out[1098]= {{2, 2}, {3, 2}, {5, 3}, {541, 1}}
           {2, 2, 3, 3, 5, 5, 5, 541}
           True
```

In[1099]:=
```
Block[{n = 2048, fi, fen},
 Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n],
   fi === Tally[fen]}]
```

```
{{2, 11}}
{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}
True
```

In[1100]:=
```
Block[{n = 12, fi, fen},
 Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n],
   fi === Tally[fen]}]
Block[{n = 12, k = 1, fi, fen},
 Column@{fi = FactorInteger[n, k], fen = sFactorExactNumber[n, k],
   fi === Tally[fen]}]
```

```
{{2, 2}, {3, 1}}
{2, 2, 3}
True

{{12, 1}}
{12}
True
```

In[1102]:=
```
Block[{n = 2434500, fi, fen},
 Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n],
   fi === Tally[fen]}]
Block[{n = 2434500, k = 1, fi, fen},
 Column@{fi = FactorInteger[n, k], fen = sFactorExactNumber[n, k],
   fi === Tally[fen]}]
```

```
{{2, 2}, {3, 2}, {5, 3}, {541, 1}}
{2, 2, 3, 3, 5, 5, 5, 541}
True

{{2434500, 1}}
{2434500}
True
```

In[1104]:=
```
Block[{n = 2434500, fi, fen},
 Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n],
   fi === Tally[fen]}]
Block[{n = 2434500, k = 2, fi, fen},
 Column@{fi = FactorInteger[n, k], fen = sFactorExactNumber[n, k],
   fi === Tally[fen]}]
```

```
{{2, 2}, {3, 2}, {5, 3}, {541, 1}}
{2, 2, 3, 3, 5, 5, 5, 541}
True
```

Out[1104]=

```
{{541, 1}, {4500, 1}}
{541, 4500}
True
```

Out[1105]=

In[1106]:=
```
Block[{n = 2434500, fi, fen},
 Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n],
   fi === Tally[fen]}]
Block[{n = 2434500, k = 3, fi, fen},
 Column@{fi = FactorInteger[n, k], fen = sFactorExactNumber[n, k],
   fi === Tally[fen]}]
```

```
{{2, 2}, {3, 2}, {5, 3}, {541, 1}}
{2, 2, 3, 3, 5, 5, 5, 541}
True
```

Out[1106]=

```
{{5, 3}, {36, 1}, {541, 1}}
{5, 5, 5, 36, 541}
True
```

Out[1107]=

In[1108]:=
```
Block[{n = 10^100 + 1, fi, fen},
 Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n], Times @@ fen,
   fi === Tally[fen]}]
Block[{n = 10^100 + 1, k = 4, fi, fen},
 Column@{fi = FactorInteger[n, k], fen = sFactorExactNumber[n, k],
   Times @@ fen, fi === Tally[fen]}]
```

Out[1108]=
```
{{73, 1}, {137, 1}, {401, 1}, {1201, 1},
 {1601, 1}, {1 676 321, 1}, {5 964 848 081, 1},
 {129 694 419 029 057 750 551 385 771 184 564 274 499 075 700 947 656 757 821 537 291 527 ↓
   196 801, 1}}
{73, 137, 401, 1201, 1601, 1 676 321, 5 964 848 081,
 129 694 419 029 057 750 551 385 771 184 564 274 499 075 700 947 656 757 821 537 291 527 ↓
  196 801}
10 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 ↓
 000 000 000 000 000 000 000 000 000 000 000 001
True
```

Out[1109]=
```
{{1201, 1}, {1601, 1}, {4 010 401, 1},
 {1 296 814 508 839 693 536 173 209 832 765 271 992 846 610 925 502 473 758 289 451 540 ↓
   212 712 414 540 699 659 186 801, 1}}
{1201, 1601, 4 010 401,
 1 296 814 508 839 693 536 173 209 832 765 271 992 846 610 925 502 473 758 289 451 540 212 ↓
  712 414 540 699 659 186 801}
10 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 ↓
 000 000 000 000 000 000 000 000 000 000 000 001
True
```

In[1110]:=
```
Block[{n = 2048, fi, fen},
 Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n], Length@fen,
   fi === Tally[fen]}]
Block[{n = 2048, k = 2, fi, fen},
 Column@{fi = FactorInteger[n, k], fen = sFactorExactNumber[n, k],
   Length@fen, fi === Tally[fen]}]
```

Out[1110]=
```
{{2, 11}}
{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}
11
True
```

Out[1111]=
```
{{2, 11}}
{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}
11
True
```

```
In[1112]:= Block[{n = 501760, fi, fen},
            Column@{fi = FactorInteger[n], fen = sFactorExactNumber[n], Length@fen,
              fi === Tally[fen]}]
           Block[{n = 501760, k = 2, fi, fen},
            Column@{fi = FactorInteger[n, k], fen = sFactorExactNumber[n, k],
              Length@fen, fi === Tally[fen]}]

Out[1112]= {{2, 11}, {5, 1}, {7, 2}}
           {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 5, 7, 7}
           14
           True

Out[1113]= {{7, 2}, {10240, 1}}
           {7, 7, 10240}
           3
           True
```

Since all arguments, options and properties of sFactorExactNumber are inherited from the FactorInteger built-in symbol, the following examples of extensions also work:

```
In[1114]:= With[{n = 3 / 8}, Column@{FactorInteger[n], sFactorExactNumber[n]}]

Out[1114]= {{2, -3}, {3, 1}}
           {1/2, 1/2, 1/2, 3}

In[1115]:= With[{n = 3 / 8, k = 1}, Column@{FactorInteger[n, k], sFactorExactNumber[n, k]}]

Out[1115]= {{3/8, 1}}
           {3/8}

In[1116]:= With[{n = 22 / 7}, Column@{FactorInteger[n], sFactorExactNumber[n]}]

Out[1116]= {{2, 1}, {7, -1}, {11, 1}}
           {2, 1/7, 11}

In[1117]:= With[{n = 22 / 7, k = 2},
            Column@{FactorInteger[n, k], sFactorExactNumber[n, k]}]

Out[1117]= {{2, 1}, {7, -1}, {11, 1}}
           {2, 1/7, 11}

In[1118]:= With[{n = 2345354 / 2424245},
            Column@{FactorInteger[n], sFactorExactNumber[n]}]

Out[1118]= {{2, 1}, {5, -1}, {11, 1}, {17, 1}, {311, -1}, {1559, -1}, {6271, 1}}
           {2, 1/5, 11, 17, 1/311, 1/1559, 6271}
```

In[1119]:=
```
With[{n = 2 345 354 / 2 424 245, k = 2},
 Column@{FactorInteger[n, k], sFactorExactNumber[n, k]}]
```

Out[1119]=
```
{{374, 1}, {1555, -1}, {1559, -1}, {6271, 1}}
{374, 1/1555, 1/1559, 6271}
```

In[1120]:=
```
With[{n = 2 270 302 672 / 72 921 849 600 595},
 Column@{FactorInteger[n], sFactorExactNumber[n]}]
```

Out[1120]=
```
{{2, 4}, {5, -1}, {11, 3}, {17, 1}, {311, -4}, {1559, -1}, {6271, 1}}
{2, 2, 2, 2, 1/5, 11, 11, 11, 17, 1/311, 1/311, 1/311, 1/311, 1/1559, 6271}
```

In[1121]:=
```
With[{n = 2 270 302 672 / 72 921 849 600 595, k = 2},
 Column@{FactorInteger[n, k], sFactorExactNumber[n, k]}]
Times @@ {1/1559, 6271, 362 032, 1/46 774 759 205}
```

Out[1121]=
```
{{1559, -1}, {6271, 1}, {362 032, 1}, {46 774 759 205, -1}}
{1/1559, 6271, 362 032, 1/46 774 759 205}
```

Out[1122]=
$$\frac{2\,270\,302\,672}{72\,921\,849\,600\,595}$$

In[1123]:=
```
With[{n = i}, Column@{FactorInteger[n], sFactorExactNumber[n]}]
```

Out[1123]=
```
{{i, 1}}
{i}
```

In[1124]:=
```
Block[{n = 2 i, fen},
 Column@{FactorInteger[n], fen = sFactorExactNumber[n], Times @@ fen}]
```

Out[1124]=
```
{{1 + i, 2}}
{1 + i, 1 + i}
2 i
```

In[1125]:=
```
Block[{n = 8 + 2 i, fen},
 Column@{FactorInteger[n], fen = sFactorExactNumber[n], Times @@ fen}]
```

Out[1125]=
```
{{-i, 1}, {1 + i, 2}, {4 + i, 1}}
{-i, 1 + i, 1 + i, 4 + i}
8 + 2 i
```

In[1126]:=
```
Block[{n = 1 / 8 + 2 i, fen},
 Column@{FactorInteger[n], fen = sFactorExactNumber[n], Times @@ fen}]
```

Out[1126]=
```
{{-i, 1}, {1 + i, -6}, {1 + 16 i, 1}}
{-i, 1/2 - i/2, 1/2 - i/2, 1/2 - i/2, 1/2 - i/2, 1/2 - i/2, 1/2 - i/2, 1 + 16 i}
1/8 + 2 i
```

In[1127]:=
```
Block[{n = 7 / 17 - (2 / 15) i, fen},
 Column@{FactorInteger[n], fen = sFactorExactNumber[n], Times @@ fen}]
```

Out[1127]=
```
{{i, 1}, {1 + 2 i, -1}, {1 + 4 i, -1}, {2 + i, -1},
 {3, -1}, {3 + 2 i, 1}, {4 + i, -1}, {24 + 19 i, 1}}
{i, 1/5 - (2 i)/5, 1/17 - (4 i)/17, 2/5 - i/5, 1/3, 3 + 2 i, 4/17 - i/17, 24 + 19 i}
7/17 - (2 i)/15
```

In[1128]:=
```
With[{n = 2}, Column@{FactorInteger[n], sFactorExactNumber[n]}]
With[{n = 2},
 Column@{FactorInteger[n, GaussianIntegers → True],
   sFactorExactNumber[n, GaussianIntegers → True]}]
```

Out[1128]=
```
{{2, 1}}
{2}
```

Out[1129]=
```
{{-i, 1}, {1 + i, 2}}
{-i, 1 + i, 1 + i}
```

In[1130]:=
```
With[{n = 0},
 Column@{FactorInteger[n, GaussianIntegers → True],
   sFactorExactNumber[n, GaussianIntegers → True]}]
```

Out[1130]=
```
{{0, 1}}
{0}
```

In[1131]:=
```
With[{n = 3 / 8},
 Column@{FactorInteger[n, GaussianIntegers → True],
   sFactorExactNumber[n, GaussianIntegers → True]}]
```

Out[1131]=
```
{{-i, 1}, {1 + i, -6}, {3, 1}}
{-i, 1/2 - i/2, 1/2 - i/2, 1/2 - i/2, 1/2 - i/2, 1/2 - i/2, 1/2 - i/2, 3}
```

In[1132]:=
```
With[{n = 2434500},
 Column@{FactorInteger[n, GaussianIntegers → True],
   sFactorExactNumber[n, GaussianIntegers → True]}]
```

Out[1132]=
```
{{-1, 1}, {1 + i, 4}, {1 + 2 i, 3}, {2 + i, 3}, {3, 2}, {10 + 21 i, 1}, {21 + 10 i, 1}}
{-1, 1 + i, 1 + i, 1 + i, 1 + i, 1 + 2 i, 1 + 2 i,
 1 + 2 i, 2 + i, 2 + i, 2 + i, 3, 3, 10 + 21 i, 21 + 10 i}
```

In[1133]:=
```
With[{n = 2 345 354 / 2 424 245},
 Column@{FactorInteger[n, GaussianIntegers → True],
   sFactorExactNumber[n, GaussianIntegers → True]}]
```

Out[1133]=
$$\{\{-i, 1\}, \{1+i, 2\}, \{1+2i, -1\}, \{1+4i, 1\}, \{2+i, -1\},$$
$$\{4+i, 1\}, \{11, 1\}, \{311, -1\}, \{1559, -1\}, \{6271, 1\}\}$$
$$\left\{-i, 1+i, 1+i, \frac{1}{5}-\frac{2i}{5}, 1+4i, \frac{2}{5}-\frac{i}{5}, 4+i, 11, \frac{1}{311}, \frac{1}{1559}, 6271\right\}$$

In[1134]:=
```
Block[{n = 2 270 302 672 / 72 921 849 600 595, fi, fen},
 Column@{fi = FactorInteger[n, GaussianIntegers → True],
   fen = sFactorExactNumber[n, GaussianIntegers → True], Times @@ fen}]
```

Out[1134]=
$$\{\{1+i, 8\}, \{1+2i, -1\}, \{1+4i, 1\}, \{2+i, -1\},$$
$$\{4+i, 1\}, \{11, 3\}, \{311, -4\}, \{1559, -1\}, \{6271, 1\}\}$$
$$\left\{1+i, 1+i, 1+i, 1+i, 1+i, 1+i, 1+i, 1+i, \frac{1}{5}-\frac{2i}{5},\right.$$
$$\left.1+4i, \frac{2}{5}-\frac{i}{5}, 4+i, 11, 11, 11, \frac{1}{311}, \frac{1}{311}, \frac{1}{311}, \frac{1}{311}, \frac{1}{1559}, 6271\right\}$$
$$\frac{2\,270\,302\,672}{72\,921\,849\,600\,595}$$

In[1135]:=
```
Block[{n = 10^100 + 1, fi, fen},
 Column@{fi = FactorInteger[n, GaussianIntegers → True],
   fen = sFactorExactNumber[n, GaussianIntegers → True], Times @@ fen}]
```

Out[1135]=
$$\{\{1+20i, 1\}, \{1+40i, 1\}, \{3+8i, 1\}, \{4+11i, 1\},$$
$$\{8+3i, 1\}, \{11+4i, 1\}, \{20+i, 1\}, \{24+25i, 1\}, \{25+24i, 1\},$$
$$\{40+i, 1\}, \{911+920i, 1\}, \{920+911i, 1\}, \{53\,791+55\,420i, 1\},$$
$$\{55\,420+53\,791i, 1\}, \{253\,487\,741\,966\,712\,446\,243\,392\,139\,061\,475\,545 +$$
$$255\,809\,272\,118\,262\,281\,386\,782\,632\,429\,702\,924\,i, 1\},$$
$$\{255\,809\,272\,118\,262\,281\,386\,782\,632\,429\,702\,924 +$$
$$253\,487\,741\,966\,712\,446\,243\,392\,139\,061\,475\,545\,i, 1\}\}$$
$$\{1+20i, 1+40i, 3+8i, 4+11i, 8+3i, 11+4i, 20+i,$$
$$24+25i, 25+24i, 40+i, 911+920i, 920+911i, 53\,791+55\,420i,$$
$$55\,420+53\,791i, 253\,487\,741\,966\,712\,446\,243\,392\,139\,061\,475\,545 +$$
$$255\,809\,272\,118\,262\,281\,386\,782\,632\,429\,702\,924\,i,$$
$$255\,809\,272\,118\,262\,281\,386\,782\,632\,429\,702\,924 +$$
$$253\,487\,741\,966\,712\,446\,243\,392\,139\,061\,475\,545\,i\}$$
$$10\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,001$$

In[1136]:=
```
Block[{n = 2048, fi, fen},
 Column@{fi = FactorInteger[n, GaussianIntegers → True],
   fen = sFactorExactNumber[n, GaussianIntegers → True], Times @@ fen}]
```

Out[1136]=
```
{{i, 1}, {1 + i, 22}}
{i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i,
 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i, 1 + i}
2048
```

Applications

Originally, sFactorExactNumber was developed for tree graphs manipulations. Below are some possible codes:

In[1137]:=
```
Table[FactorInteger[2^2^n + 1], {n, 7}] // Column
Table[sFactorExactNumber[2^2^n + 1], {n, 7}] // Column
```

Out[1137]=
```
{{5, 1}}
{{17, 1}}
{{257, 1}}
{{65 537, 1}}
{{641, 1}, {6 700 417, 1}}
{{274 177, 1}, {67 280 421 310 721, 1}}
{{59 649 589 127 497 217, 1}, {5 704 689 200 685 129 054 721, 1}}
```

Out[1138]=
```
{5}
{17}
{257}
{65 537}
{641, 6 700 417}
{274 177, 67 280 421 310 721}
{59 649 589 127 497 217, 5 704 689 200 685 129 054 721}
```

In[1139]:=
```
FactorInteger[20!]
sFactorExactNumber[20!]
%% === Tally[%]
```

Out[1139]=
```
{{2, 18}, {3, 8}, {5, 4}, {7, 2}, {11, 1}, {13, 1}, {17, 1}, {19, 1}}
```

Out[1140]=
```
{2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
 2, 3, 3, 3, 3, 3, 3, 3, 3, 5, 5, 5, 5, 7, 7, 11, 13, 17, 19}
```

Out[1141]=
```
True
```

Discussion

Possible Issues:

In[1142]:=
```
Table[Timing[sFactorExactNumber[2^n - 1]; n], {n, 50, 400, 50}]
```

Out[1142]=
```
{{0., 50}, {0., 100}, {0.0156001, 150}, {0., 200},
 {0.234002, 250}, {0.0156001, 300}, {0.577204, 350}, {0.0312002, 400}}
```

4.8 sENextPow2

In[1143]:=
```
infoAbout@sENextPow2
```

sENextPow2[n] returns the smallest integer
exponent p that satisfies $2^p \geq |n|$ for an integer, real or rational n.

```
Attributes[sENextPow2] = {Listable, Protected, ReadProtected}
```

```
SyntaxInformation[sENextPow2] = {ArgumentsPattern → {_}}
```

Details

sENextPow2[n] returns the smallest integer exponent p that satisfies $2^p \geq |n|$ for an integer, real or rational n, and Null otherwise.

- sENextPow2 automatically threads over lists.
- You can use it to pad the signal you pass to DFT (i.e., the Fourier, FourierDCT, FourierDST, InverseFourier and other spectral analysis related functions).
- Mimics the nextpow2 in MATLAB.
- Yields $-\infty$ for n integer, rational and real zero (c.f. RealExponent built-in symbol).
- Originally inspired by [Jason, 2014; MathWorks, 2019].

Examples

Clearing global symbols:

In[1144]:=
```
Clear[n];
```

The basic behavior of sENextPow2 can be demonstrated as follows, including $MachineEpsilon, $MinMachineNumber and $MaxMachineNumber:

In[1145]:=
```
n = {7, 6.8, 22 / 3, 0, 0.0, 1, -2, 3, -4, 31, 32, 33, $MachineEpsilon,
    $MinMachineNumber, $MaxMachineNumber};
{n, sENextPow2[n]} //
  TableForm[#, TableHeadings → {{"n", "sENextPow2[n]"}, None}] & //
 Style[#, 5] &
```

Out[1146]=

n	7	6.8	$\frac{22}{3}$	0	0.	1	−2	3	−4	31	32	33	2.22045×10^{-16}	2.22507×10^{-308}	1.79769×10^{308}
sENextPow2[n]	3	3	3	−∞	−∞	0	1	2	2	5	5	6	−52	−1022	1024

More precisely:

In[1147]:=
```
n = Sort@{0, 1 / 4, .3, 1 / 3, 1 / 2, 3 / 4, 1, 2, 3, 4, 5, 31, 32, 33};
{n, sENextPow2[n]} //
  TableForm[#, TableHeadings → {{"n", "sENextPow2[n]"}, None}] & //
 Style[#, 7] &
n = -Reverse@Rest@n;
{n, sENextPow2[n]} //
  TableForm[#, TableHeadings → {{"n", "sENextPow2[n]"}, None}] & //
 Style[#, 7] &
```

Out[1148]=

n	0	$\frac{1}{4}$	0.3	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{3}{4}$	1	2	3	4	5	31	32	33
sENextPow2[n]	$-\infty$	−2	−1	−1	−1	0	0	1	2	2	3	5	5	6

Out[1150]=

n	−33	−32	−31	−5	−4	−3	−2	−1	$-\frac{3}{4}$	$-\frac{1}{2}$	$-\frac{1}{3}$	−0.3	$-\frac{1}{4}$
sENextPow2[n]	6	5	5	3	2	2	1	0	0	−1	−1	−1	−2

In[1151]:=
```
n = Sort@{1, 2, 3, 4, 5, 8, 9, 14, 15, 16, 17, 31, 32, 33, 513, 1023, 1024, 1025};
{n, sENextPow2[n]} //
  TableForm[#, TableHeadings → {{"n", "sENextPow2[n]"}, None}] & //
 Style[#, 6] &
```

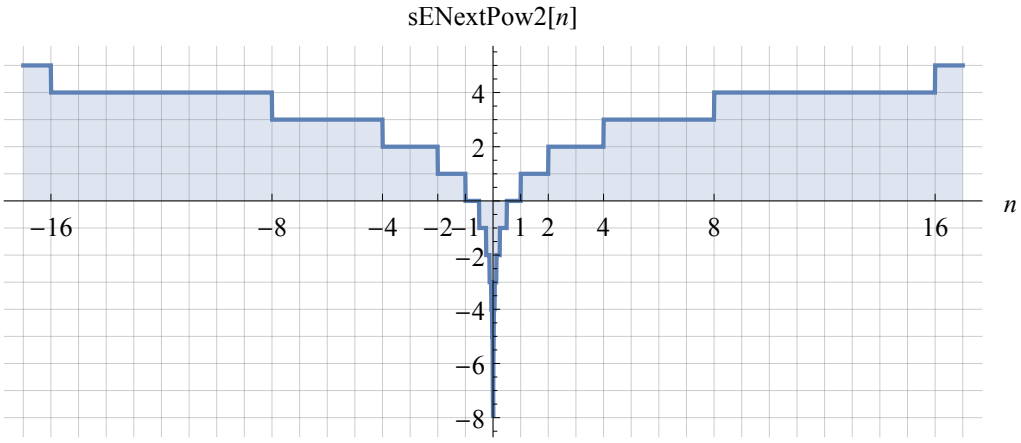
Out[1152]=

n	1	2	3	4	5	8	9	14	15	16	17	31	32	33	513	1023	1024	1025
sENextPow2[n]	0	1	2	2	3	3	4	4	4	4	5	5	5	6	10	10	10	11

Graphical illustration of the sENextPow2 behavior:

In[1153]:=
```
With[{r = 17}, Plot[sENextPow2[x], {x, -r, r}, PlotRange → Full,
  Ticks → {Join[-PowerRange[16, 1, 1 / 2], PowerRange[1, 16, 2]], Automatic},
  GridLines → {Range[-r, r], Range[-8, sENextPow2@r]},
  AxesLabel → {"n", "sENextPow2[n]"}, Filling → 0, AspectRatio → 0.4,
  ImageSize → 380]]
```

Out[1153]=



Applications

sENextPow2 was originally developed as a helper function for signal analysis. It may come handy in various circumstances as well as in development of more complex functions in the field of signal analysis.

4.9 sENextPow2N

In[1154]:=
```
infoAbout@sENextPow2N
```

sENextPow2N[n] returns the smallest integer exponent p that satisfies $2^p \geq |n|$ for an integer $n > 0$.

```
Attributes[sENextPow2N] = {Listable, Protected, ReadProtected}
```

```
SyntaxInformation[sENextPow2N] = {ArgumentsPattern → {_}}
```

Details

sENextPow2N[n] returns the smallest integer exponent p that satisfies $2^p \geq |n|$ for a positive integer (natural number) n, and Null otherwise.

- sENextPow2N automatically threads over lists.
- You can use it to pad the signal you pass to DFT (i.e., the Fourier, FourierDCT, FourierDST, InverseFourier and other spectral analysis related functions).
- Lightweight but faster version of sENextPow2.
- Originally inspired by [Jason, 2014; MathWorks, 2019].

Examples

Clearing global symbols:

In[1155]:=
```
Clear[n, num, u, tim, t1, t2, stats, ℋ];
```

The basic behavior of sENextPow2N can be demonstrated as follows:

In[1156]:=
```
n = Sort@{1, 2, 3, 4, 5, 7, 8, 9, 14, 15, 16, 17, 31, 32, 33, 513, 1023,
    1024, 1025};
{n, sENextPow2N[n]} //
  TableForm[#, TableHeadings → {{"n", "sENextPow2N[n]"}, None}] & //
 Style[#, 6] &
```

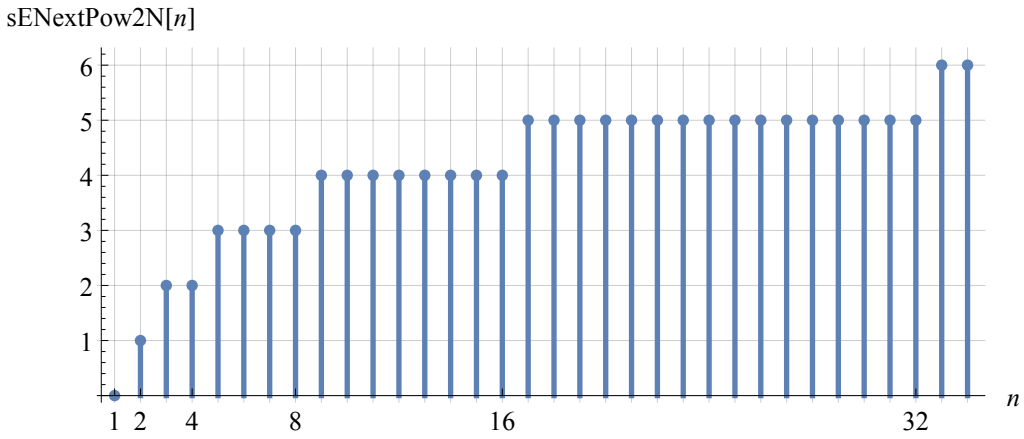
Out[1157]=

n	1	2	3	4	5	7	8	9	14	15	16	17	31	32	33	513	1023	1024	1025
sENextPow2N[n]	0	1	2	2	3	3	3	4	4	4	4	5	5	5	6	10	10	10	11

Graphical illustration of the sENextPow2N behavior:

```
In[1158]:= With[{r = 34}, ListPlot[Table[sENextPow2N[n], {n, r}], PlotRange → Full,
             Ticks → {PowerRange[1, 32, 2], Automatic},
             GridLines → {Range[1, r], Range[0, sENextPow2N@r]},
             AxesLabel → {"n", "sENextPow2N[n]"}, Filling → Axis, FillingStyle → Thick,
             AspectRatio → 0.4, ImageSize → 380]]
```

Out[1158]=



Noninteger arguments cause errors:

```
In[1159]:= sENextPow2N[6.8]
```

Snapdragon: sENextPow2N[6.8] does not match the rule base.

```
Out[1159]= $Failed
```

```
In[1160]:= sENextPow2N[22 / 3]
```

Snapdragon: sENextPow2N[$\frac{22}{3}$] does not match the rule base.

```
Out[1160]= $Failed
```

Test for positivity *n*_Integer?Positive intentionally omitted since its side effect is slowing down. However, the results provided by sENextPow2N are incorrect in such cases (compare with those provided by sENextPow2):

```
In[1161]:= {sENextPow2N[0], sENextPow2[0]}
Out[1161]= {1, -∞}

In[1162]:= {sENextPow2N[-7], sENextPow2[-7]}
Out[1162]= {4, 3}
```

Applications

sENextPow2N was originally developed as a helper function for signal analysis. It may come handy in various circumstances as well as in development of more complex functions in the field of signal analysis.

Discussion

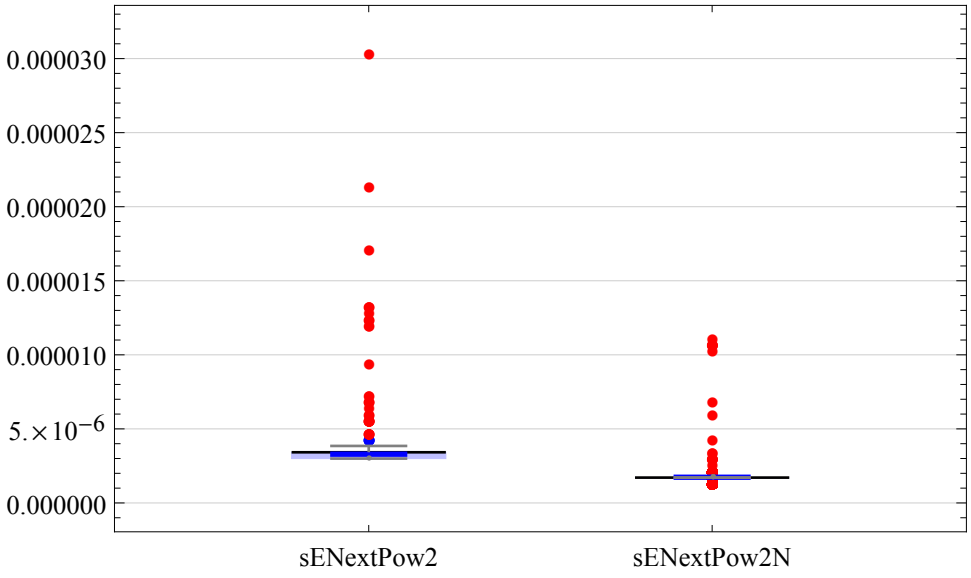
sENextPow2 vs sENextPow2N performance comparison

In[1163]:=
```
num = 5000; u = RandomInteger[{1, 10^7}]; tim = AbsoluteTiming;
t1 = Table[(sENextPow2[u]; // tim)[[1]], {num}];
t2 = Table[(sENextPow2N[u]; // tim)[[1]], {num}];
stats = {Mean, StandardDeviation, Median, MedianDeviation};
TableForm[Through[stats[#]] & /@ {t1, t2},
 TableHeadings → {{"sENextPow2", "sENextPow2N"}, stats}]
BoxWhiskerChart[{t1, t2},
 {{"Outliers", Directive[Tiny, Blue]}, {"FarOutliers", Directive[Tiny, Red]},
  {"MedianMarker", 1, Black}, {"MeanMarker", 0.5, Directive[Blue, Thick]}},
 ChartStyle → Lighter[Blue, 0.75], GridLines → {None, Automatic},
 ChartLabels → {"sENextPow2", "sENextPow2N"}]
ℋ = LocationTest[{t1, t2}, 0, "HypothesisTestData",
   VerifyTestAssumptions → All, AlternativeHypothesis → "Greater"];
ℋ["TestDataTable", All] // Magnify[#, 2 / 3] &
ℋ["TestConclusion", Automatic]
```

Out[1166]//TableForm=

	Mean	StandardDeviation	Median	MedianDe
sENextPow2	3.2925×10^{-6}	7.07463×10^{-7}	3.42096×10^{-6}	0.
sENextPow2N	1.73511×10^{-6}	3.96848×10^{-7}	1.71048×10^{-6}	0.

Out[1167]=



Out[1169]=

	Statistic	P-Value
Mann-Whitney	2.4932×10^{7}	0.
Sign	4987	0.

Out[1170]=
```
The null hypothesis that the mean difference is less than or equal to 0
 is rejected at the 5 percent level based on the Mann-Whitney test.
```

sENextPow2N appears to be about twice as fast as sENextPow2 in average.

4.10 sNextPow2

In[1171]:=
```
infoAbout@sNextPow2
```

sNextPow2[n] returns the smallest integer or rational
power of two that satisfies $2^p \geq |n|$ for an integer, real or rational n.

```
Attributes[sNextPow2] = {Listable, Protected, ReadProtected}
```

```
SyntaxInformation[sNextPow2] = {ArgumentsPattern → {_, _.}}
```

Details

sNextPow2[n, sgn] returns

- the smallest integer or rational power of two, 2^p, that satisfies $2^p \geq |n|$ for an integer, real or rational n if $sgn == 0$ (default) or False (i.e. even output with respect to n),
- preserves the sign of the original n if $sgn == 1$ or True (i.e. odd output),
- returns Null otherwise.
- Yields $0 == 2^{-\infty}$ for n zero.
- sNextPow2 automatically threads over lists.
- You can use it to pad the signal you pass to DFT (i.e., the Fourier, FourierDCT, FourierDST, InverseFourier and other spectral analysis related functions).
- Originally inspired by [Jason, 2014; MathWorks, 2019].

Examples

Clearing global symbols:

In[1172]:=
```
Clear[n];
```

The basic behavior of sENextPow2 can be demonstrated as follows, including $MachineEpsilon:

In[1173]:=
```
n = {7, 6.8, 22 / 3, 0, 0.0, 1, -2, 3, -4, 31, 32, 33, $MachineEpsilon};
{n, sNextPow2[n]} //
  TableForm[#, TableHeadings → {{"n", "sNextPow2[n]"}, None}] & //
 Style[#, 7] &
```

Out[1174]=

n	7	6.8	$\frac{22}{3}$	0	0.	1	−2	3	−4	31	32	33	2.22045×10^{-16}
sNextPow2[n]	8	8	8	0	0	1	2	4	4	32	32	64	$\frac{1}{4\,503\,599\,627\,370\,496}$

In[1175]:=
```
n = {7, 6.8, 22 / 3, 0, 0.0, 1, -2, 3, -4, 31, 32, 33, $MachineEpsilon};
{n, sNextPow2[n, 1]} //
  TableForm[#, TableHeadings → {{"n", "sNextPow2[n, 1]"}, None}] & //
 Style[#, 7] &
```

Out[1176]=

n	7	6.8	$\frac{22}{3}$	0	0.	1	−2	3	−4	31	32	33	2.22045×10^{-16}
sNextPow2[n, 1]	8	8	8	0	0	1	−2	4	−4	32	32	64	$\frac{1}{4\,503\,599\,627\,370\,496}$

More precisely:

In[1177]:=
```
n = Sort@{0, 1 / 4, .3, 1 / 3, 1 / 2, 3 / 4, 1, 2, 3, 4, 5, 31, 32, 33};
{n, sNextPow2[n]} //
  TableForm[#, TableHeadings → {{"n", "sNextPow2[n]"}, None}] & //
 Style[#, 7] &
n = -Reverse@Rest@n;
{n, sNextPow2[n]} //
  TableForm[#, TableHeadings → {{"n", "sNextPow2[n]"}, None}] & //
 Style[#, 7] &
{n, sNextPow2[n, 1]} //
  TableForm[#, TableHeadings → {{"n", "sNextPow2[n, 1]"}, None}] & //
 Style[#, 7] &
```

Out[1178]=

n	0	$\frac{1}{4}$	0.3	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{3}{4}$	1	2	3	4	5	31	32	33
sNextPow2[n]	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	2	4	4	8	32	32	64

Out[1180]=

n	−33	−32	−31	−5	−4	−3	−2	−1	$-\frac{3}{4}$	$-\frac{1}{2}$	$-\frac{1}{3}$	−0.3	$-\frac{1}{4}$
sNextPow2[n]	64	32	32	8	4	4	2	1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$

Out[1181]=

n	−33	−32	−31	−5	−4	−3	−2	−1	$-\frac{3}{4}$	$-\frac{1}{2}$	$-\frac{1}{3}$	−0.3	$-\frac{1}{4}$
sNextPow2[n, 1]	−64	−32	−32	−8	−4	−4	−2	−1	−1	$-\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{4}$

In[1182]:=
```
n = Sort@{1, 2, 3, 4, 5, 8, 9, 14, 15, 16, 17, 31, 32, 33, 513, 1023, 1024, 1025};
{n, sNextPow2[n]} //
  TableForm[#, TableHeadings → {{"n", "sNextPow2[n]"}, None}] & //
 Style[#, 6] &
```

Out[1183]=

n	1	2	3	4	5	8	9	14	15	16	17	31	32	33	513	1023	1024	1025
sNextPow2[n]	1	2	4	4	8	8	16	16	16	16	32	32	32	64	1024	1024	1024	2048

Apply sNextPow2 to $MachineEpsilon, $MinMachineNumber and $MaxMachineNumber:

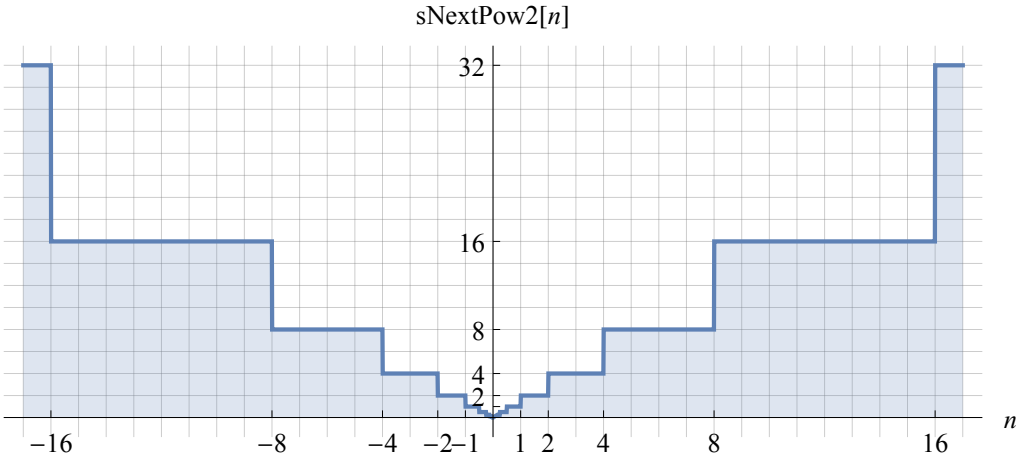
In[1184]:=
```
sNextPow2 /@ {$MachineEpsilon, $MinMachineNumber, $MaxMachineNumber} //
 Pane[#, 540] &
```

Out[1184]=
$$\left\{\frac{1}{4\,503\,599\,627\,370\,496},\ 1 /\right.$$
44 942 328 371 557 897 693 232 629 769 725 618 340 449 424 473 557 664 318 357 520 289 433 1
119 330 601 884 005 280 028 469 967 848 339 414 697 442 203 604 155 623 211 857 659 868 53
371 319 075 554 900 311 523 529 863 270 738 021 251 442 209 537 670 585 615 720 368 478 27
627 671 146 574 559 986 811 484 619 929 076 208 839 082 406 056 034 304,
179 769 313 486 231 590 772 930 519 078 902 473 361 797 697 894 230 657 273 430 081 157 732 6
477 322 407 536 021 120 113 879 871 393 357 658 789 768 814 416 622 492 847 430 639 474 124
485 276 302 219 601 246 094 119 453 082 952 085 005 768 838 150 682 342 462 881 473 913 110
510 684 586 298 239 947 245 938 479 716 304 835 356 329 624 224 137 216$\left.\right\}$

Graphical illustration of the sNextPow2 behavior:

In[1185]:=
```
Plot[sNextPow2[x], {x, -17, 17}, PlotRange → Full,
 Ticks → {Join[-PowerRange[16, 1, 1 / 2], PowerRange[1, 16, 2]],
   PowerRange[1, 32, 2]}, GridLines → {Range[-17, 17], Range[0, 32, 2]},
 AxesLabel → {"n", "sNextPow2[n]"}, Filling → 0, AspectRatio → 0.4,
 ImageSize → 380]
```

Out[1185]=



Applications

sNextPow2 was originally developed as a helper function for signal analysis. It may come handy in various circumstances as well as in development of more complex functions in the field of signal analysis.

4.11 sNextPow2N

In[1186]:=
```
infoAbout@sNextPow2N
```

sNextPow2N[n] returns the smallest integer power of two that satisfies $2^p \geq |n|$ for an integer $n > 0$.

```
Attributes[sNextPow2N] = {Listable, Protected, ReadProtected}
```

```
SyntaxInformation[sNextPow2N] = {ArgumentsPattern → {_}}
```

Details

sNextPow2N[n] returns the smallest integer power of two, 2^p, that satisfies $2^p \geq |n|$ for a positive integer (natural number) n, and Null otherwise.

- sNextPow2N automatically threads over lists.
- You can use it to pad the signal you pass to DFT (i.e., the Fourier, FourierDCT, FourierDST, InverseFourier and other spectral analysis related functions).
- Lightweight but faster version of sNextPow2.
- Originally inspired by [Jason, 2014; MathWorks, 2019].

Examples

Clearing global symbols:

In[1187]:=
```
Clear[n, num, u, tim, t1, t2, stats, ℋ];
```

The basic behavior of sENextPow2N can be demonstrated as follows:

In[1188]:=
```
n = Sort@{1, 2, 3, 4, 5, 7, 8, 9, 14, 15, 16, 17, 31, 32, 33, 513, 1023,
    1024, 1025};
{n, sNextPow2N[n]} //
  TableForm[#, TableHeadings → {{"n", "sNextPow2N[n]"}, None}] & //
 Style[#, 6] &
```

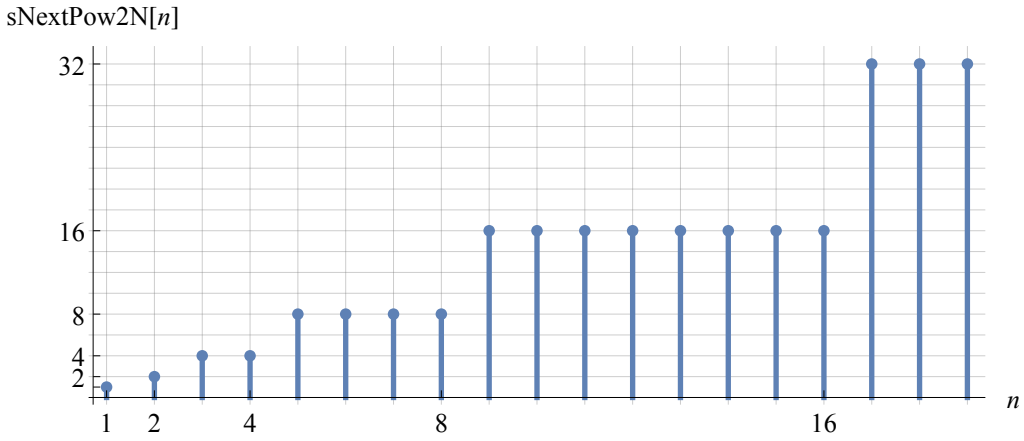
Out[1189]=

n	1	2	3	4	5	7	8	9	14	15	16	17	31	32	33	513	1023	1024	1025
sNextPow2N[n]	1	2	4	4	8	8	8	16	16	16	16	32	32	32	64	1024	1024	1024	2048

Graphical illustration of the sNextPow2N behavior:

```
In[1190]:= With[{r = 19}, ListPlot[Table[sNextPow2N[n], {n, r}], PlotRange → Full,
             Ticks → {PowerRange[1, 16, 2], PowerRange[1, 32, 2]},
             GridLines → {Range[1, r], Range[0, sNextPow2N[r], 2]},
             AxesLabel → {"n", "sNextPow2N[n]"}, Filling → Axis, FillingStyle → Thick,
             AspectRatio → 0.4, ImageSize → 380]]
```

Out[1190]=



Test for positivity *n*_Integer?Positive intentionally omitted since its side effect is slowing down. However, the results provided by sNextPow2N are incorrect in such cases (compare with those provided by sNextPow2):

```
In[1191]:= {sNextPow2N[0], sNextPow2[0]}
Out[1191]= {2, 0}

In[1192]:= {sNextPow2N[-7], sNextPow2[-7]}
Out[1192]= {16, 8}
```

Noninteger arguments cause errors:

```
In[1193]:= sNextPow2N[6.8]
           Snapdragon: sNextPow2N[6.8] does not match the rule base.
Out[1193]= $Failed

In[1194]:= sNextPow2N[22 / 3]
           Snapdragon: sNextPow2N[22/3] does not match the rule base.
Out[1194]= $Failed
```

Applications

sNextPow2N was originally developed as a helper function for signal analysis. It may come handy in various circumstances as well as in development of more complex functions in the field of signal analysis.

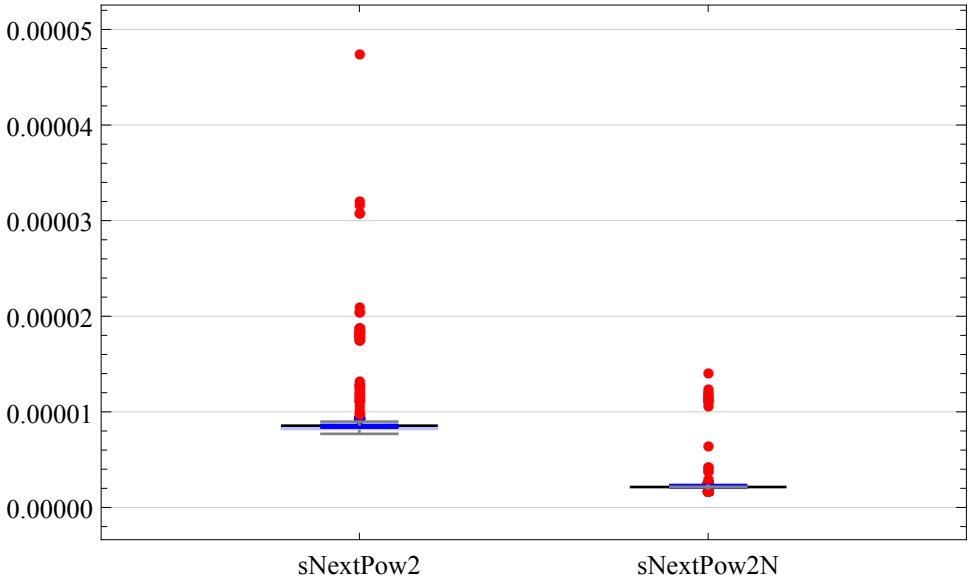
Discussion

In[1195]:=
```
num = 5000; u = RandomInteger[{1, 10^7}]; tim = AbsoluteTiming;
t1 = Table[(sNextPow2[u]; // tim)[[1]], {num}];
t2 = Table[(sNextPow2N[u]; // tim)[[1]], {num}];
stats = {Mean, StandardDeviation, Median, MedianDeviation};
TableForm[Through[stats[#]] & /@ {t1, t2},
 TableHeadings → {{"sNextPow2", "sNextPow2N"}, stats}]
BoxWhiskerChart[{t1, t2},
 {{"Outliers", Directive[Tiny, Blue]}, {"FarOutliers", Directive[Tiny, Red]},
  {"MedianMarker", 1, Black}, {"MeanMarker", 0.5, Directive[Blue, Thick]}},
 ChartStyle → Lighter[Blue, 0.75], GridLines → {None, Automatic},
 ChartLabels → {"sNextPow2", "sNextPow2N"}]
ℋ = LocationTest[{t1, t2}, 0, "HypothesisTestData",
   VerifyTestAssumptions → All, AlternativeHypothesis → "Greater"];
ℋ["TestDataTable", All] // Magnify[#, 2 / 3] &
ℋ["TestConclusion", Automatic]
```

Out[1198]//TableForm=

	Mean	StandardDeviation	Median	MedianDevia
sNextPow2	8.49022×10^{-6}	1.15869×10^{-6}	8.5524×10^{-6}	0.
sNextPow2N	2.20592×10^{-6}	4.83362×10^{-7}	2.1381×10^{-6}	0.

Out[1199]=



Out[1201]=

	Statistic	P-Value
Mann-Whitney	2.49454×10^{7}	0.
Sign	4989	0.

Out[1202]=
```
The null hypothesis that the mean difference is less than or equal to 0
 is rejected at the 5 percent level based on the Mann-Whitney test.
```

sNextPow2N appears to be about by half of order faster than sNextPow2 in average.

4.12 sParseIdNumber

```
In[1203]:= infoAbout@sParseIdNumber
```

sParseIdNumber[*pid*] parses a National (Personal) Identification Number specified by a string *pid*.

```
Attributes[sParseIdNumber] = {Protected, ReadProtected}

Options[sParseIdNumber] = {Country → CzechRepublic}

SyntaxInformation[sParseIdNumber] = {ArgumentsPattern → {_, OptionsPattern[]}}
```

Details

sParseIdNumber[*pid*, *opts*] parses a National (Personal) Identification Number specified by a string *pid*, pertinent to the country given by *opts*, and returns deduceable items like date of birth, sex etc. Option "Country" values implemented so far:

- "Country" → "CzechRepublic" (default) uses the Birth Number [Lorenc, 2013; Wikipedia, 2019a] (referred to as "rodné číslo") as *pid*, returning corresponding date of birth, sex (strings) and an integer "serial number" sno in the form {"YYYY-MM-DD", "M"|"F", sno}.
  - Some tests of the Birth Number syntax are performed.
  - The system enables to work until the year 2054 (i.e., the last day for which it works is December 31, 2053). Hopefully the government will introduce a better system until then.

Examples

Clearing global symbols:

```
In[1204]:= Clear[agesex];
```

Current examples below only apply to the Czech Birth Number. All National (Personal) Identification Number used in the following examples are purely fictitious and do not belong to any real person, living or dead.

Correct usage

The Birth Number with properly placed single slash or without slash:

```
In[1205]:= sParseIdNumber /@ {"690620/5273", "6906205273"}
Out[1205]= {{1969-06-20, M, 527}, {1969-06-20, M, 527}}
```

Somebody who was born in 1999:

```
In[1206]:= sParseIdNumber["990724/3214"]
Out[1206]= {1999-07-24, M, 321}
```

Somebody who was born in 2000:

```
In[1207]:= sParseIdNumber["007712/5466"]
Out[1207]= {2000-07-12, F, 546}
```

Somebody who was born in 2001:

```
In[1208]:= sParseIdNumber["015911/7673"]
Out[1208]= {2001-09-11, F, 767}
```

A fictitious person who will be born on the last day of the year 2053. The Birth Number system still works:

```
In[1209]:= sParseIdNumber["538231/9899"]
Out[1209]= {2053-12-31, F, 989}
```

Somebody who will be born on the first day of the year 2054. No error message is issued but the result is wrong. There is no way how to distinguish a baby who will be born in 2054 from a baby who was born in 1954, and for later years alike:

```
In[1210]:= sParseIdNumber["545101/0015"]
Out[1210]= {1954-01-01, F, 1}
```

There is so called "1954–1985 exception": the remainder after dividing 9-digit Birth Number by 11 is 10, hence, the true guard digit is 10, but 0 is used instead. This was practiced from 1954 (when the guard digit was introduced) till the end of 1985. As a result, the 10-digit Birth Number is not divisible by 11. This exception was applied to about one thousand Birth Numbers, and the allocation of such Birth Numbers has been terminated in 1985 according to the internal regulation of the Federal Statistical Office No. Vk. 2898/1985 [Lorenc, 2013]. As an example, this is the Birth Number of a person born in 1969, i.e. within the "1954–1985 exception" period, which mey be valid despite not being divisible by 11 (but giving remainder 10). In such a case, a warning message is issued:

```
In[1211]:= sParseIdNumber["690620/5340", "Country" → "CZ"]
          {Divisible[6 906 205 340, 11], Mod[690 620 534, 11]}

          ▪▪▪ sParseIdNumber: NOTE: Czech BN 690620/5340: a [1954–85] exception, may but may not be valid.

Out[1211]= {1969-06-20, M, 534}

Out[1212]= {False, 10}
```

Compare to the next case. The birth date is exactly the same, but now the remainder differs from 10, leading immediately to an error:

```
In[1213]:= sParseIdNumber["690620/5270", "Country" → "CZ"]
          {Divisible[6 906 205 270, 11], Mod[690 620 527, 11]}

          ▪▪▪ sParseIdNumber: Czech BN 690620/5270: neither guard digit match nor a 1954–85 exception.

Out[1213]= $Failed

Out[1214]= {False, 3}
```

The ranges of months and days in months are checked for:

```
In[1215]:= sParseIdNumber["691320/5277", "Country" → "CZ"]

          ▪▪▪ sParseIdNumber: Czech BN 691320/5277: ..MM../... out of range.

Out[1215]= $Failed
```

Feb 29th does not exist in the non-leap year 1969:

```
In[1216]:= sParseIdNumber["690229/5279", "Country" → "CZ"]

          ▪▪▪ sParseIdNumber: Czech BN 690229/5279: ....DD/... out of range.

Out[1216]= $Failed
```

But Feb 29th does exist in the leap year 1968:

```
In[1217]:= sParseIdNumber["680229/5478"]

Out[1217]= {1968-02-29, M, 547}
```

The following examples treat the short format valid till the end of 1953 only:

```
In[1218]:= sParseIdNumber["530620/527"]

Out[1218]= {1953-06-20, M, 527}
```

An attempt parsing short format in with later date fails:

```
In[1219]:= sParseIdNumber["540620/527"]

          ▪▪▪ sParseIdNumber: Czech BN 540620/527: short format until 1954 only.

Out[1219]= $Failed
```

However, this is a valid Birth Number:

```
In[1220]:= sParseIdNumber["530201/5477"]

Out[1220]= {2053-02-01, M, 547}
```

Applications

In one of our biomedical research, the (anonymized) Birth Number was the only patient identifier. Mass check in the list of Birth Numbers works as follows, and transforms the Birth Numbers into

pairs {current_age_in_years, sex}:

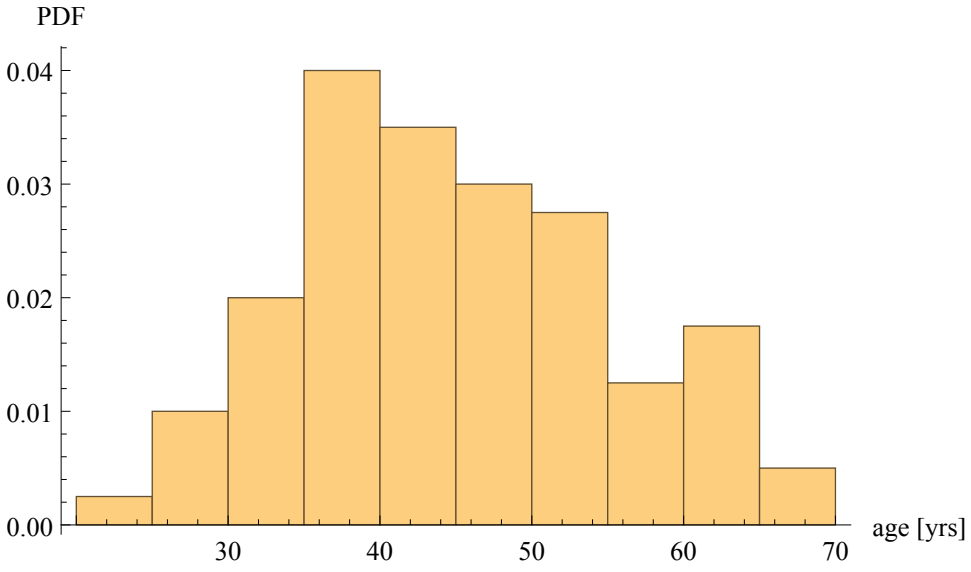
In[1221]:=
```
agesex =
 With[
  {pids = {"690620/5273", "781004/4979", "810822/4993", "740209/5536",
     "920422/5646", "791002/5035", "845604/5246", "780705/5245",
     "720319/5747", "670715/2397", "840503/5265", "680308/0768",
     "620712/0491", "640710/1987", "860221/4445", "871218/5713",
     "780306/5281", "710923/5287", "900519/5441", "780316/5271",
     "580906/1445", "770325/5241", "800329/5575", "671001/1704",
     "880113/2241", "670322/2042", "571013/1064", "830522/5258",
     "780205/5283", "800927/5241", "590301/1345", "720725/5242",
     "611225/0705", "821201/5240", "770106/5009", "810518/5880",
     "560522/2414", "8907095637", "8007205261", "7012295246",
     "8803245715", "7901315433", "6410082184", "8204254938",
     "840710/5003", "7707075860", "7156115241", "6812051422",
     "5702021798", "7411045895", "9705065656", "7008125256",
     "760415/5273", "7212134952", "6708300280", "8910015829",
     "8057305432", "5703162168", "8310035239", "7404255001",
     "8706025713", "7611265244", "5704112304", "7012255250",
     "7952135279", "8101305245", "720827/5283", "8011235254",
     "6904265555", "8707045710", "7105175253", "6312221443",
     "6005200212", "7701225279", "9862317267", "6403031239",
     "6508191250", "6601144550", "7255221666", "8555118110",
     "821220/1822", "6262307777", "7611096251", "8456307112",
     "8008204721", "8512074494", "8107103730", "505404/106",
     "520522/120", "520110/244"}},
  MapAt[First[DateDifference[#1, Today, "Years"]] &,
   (sParseIdNumber /@ pids)[[All, 1 ;; 2]], {All, 1}]]
```

Out[1221]=
```
{{49.9151, M}, {40.6247, M}, {37.7425, M}, {45.274, M}, {27.0765, M},
 {39.6301, M}, {34.9589, F}, {40.874, M}, {47.1694, M}, {51.8466, M},
 {35.0464, M}, {51.1995, M}, {56.8548, M}, {54.8603, M}, {33.2411, M},
 {31.4192, M}, {41.2049, M}, {47.6548, M}, {29.0027, M}, {41.1776, M},
 {60.7014, M}, {42.153, M}, {39.1421, M}, {51.6329, M}, {31.3479, M},
 {52.1612, M}, {61.6, M}, {35.9945, M}, {41.2849, M}, {38.6438, M},
 {60.2186, M}, {46.8192, M}, {57.4, M}, {36.4658, M}, {42.3671, M},
 {38.0055, M}, {62.9945, M}, {29.863, M}, {38.8329, M}, {48.389, M},
 {31.1557, M}, {40.2986, M}, {54.6137, M}, {37.0683, M}, {34.8603, M},
 {41.8685, M}, {47.9397, F}, {50.4548, M}, {62.2932, M}, {44.5397, M},
 {22.0383, M}, {48.7699, M}, {43.0956, M}, {46.4329, M}, {51.7205, M},
 {29.6329, M}, {38.8055, F}, {62.1776, M}, {35.6274, M}, {45.0683, M},
 {31.9644, M}, {42.4795, M}, {62.1066, M}, {48.4, M}, {40.263, F},
 {38.3014, M}, {46.7288, M}, {38.4877, M}, {50.0656, M}, {31.8767, M},
 {48.0082, M}, {55.4082, M}, {59, M}, {42.3233, M}, {20.3836, F},
 {55.2131, M}, {53.7507, M}, {53.3452, M}, {46.9945, F}, {34.0246, F},
 {36.4137, M}, {56.3863, F}, {42.526, M}, {34.8877, F}, {38.7479, M},
 {33.4493, M}, {37.8603, M}, {69.1257, F}, {66.9945, M}, {67.3562, M}}
```

The histogram estimating the probability density function of males' age:

In[1222]:=
```
Histogram[Cases[agesex, {_, "M"}][[All, 1]], {5}, "PDF",
 AxesLabel → {"age [yrs]", "PDF"}]
```

Out[1222]=

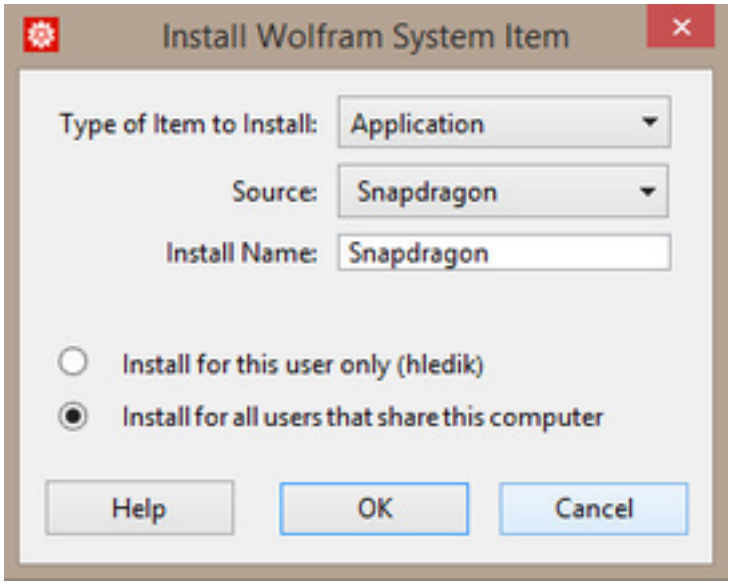


Appendices

Appendix A: System-wide installation

Occasionally, one needs to install Snapdragon into a system-wide directory instead of into the default user-specific directory as recommended above in section Getting. This may be advantageous if the computer is shared by multiple users with different user accounts. However, there is currently no easy way to install paclets for all users automatically, and one should use this installation on his/her own risk [Horvát, 2017c]. That is why I provide a provisional procedure for doing so before the paclet management system is officially documented.

1. Rename Snapdragon-M.m.R.paclet to Snapdragon-M.m.R.zip (paclet files appear to be simply zip files), and unpack using your file manager, archive manager or (un)packer program. Make sure to keep the directory tree upon unpacking. If you can force your packer program to unpack the *.paclet files directly (like, e.g., Total Commander), the renaming step can be skipped.
2. Rename the unpacked directory Snapdragon-M.m.R to Snapdragon (i.e., simply strip the dash and version from its name).
3. Run your *Mathematica*. Make sure you have administrator privileges.
4. Use menu item File ▷ Install…, choose *Application* in *Type of Item to Install* and click the *Install for all users that share this computer* radio button.
5. Choose *From Directory …* in the *Source* drop-down menu: the directory selection dialog window will open. Select the directory Snapdragon unpacked as described above, so the dialog looks like in the picture below, and confirm by clicking OK.



6. Snapdragon will be installed in FileNameJoin[{$BaseDirectory, "Applications"}] being accessible for all users. To install into that directory you will probably need administrator privileges!
7. There is no need to uninstall as soon as an upgrade is available. Just install a new version over the previous one.
8. To uninstall, just exit *Mathematica* and delete the directory Snapdragon in the above mentioned location by hand.

Appendix B: Dependencies

Most symbols defined by the Snapdragon application are constructed solely from system built-in symbols only. There exist, however, a few Snapdragon symbols which depend on other some other Snapdragon symbols. Here we list all such internal dependencies:

1.1 sListDecimate depends on: 4.6 sEmptyListsQ
1.2 sListPush depends on: built-in symbols only
1.3 sListTruncate depends on: built-in symbols only
1.4 sRotateHalfLength depends on: built-in symbols only
1.5 sDimensionsBoxed depends on: built-in symbols only
1.6 sExprOccupancy depends on: 1.5 sDimensionsBoxed
1.7 sDiagonals depends on: built-in symbols only

2.1 sSamplingSpan depends on: built-in symbols only
2.2 sCircularlySample depends on: built-in symbols only
2.3 sCVectorDFTUnpack depends on: built-in symbols only
2.4 sRVectorCPack depends on: built-in symbols only
2.5 sRealFourier depends on: built-in symbols only
2.6 sSavGolKernel1D depends on: built-in symbols only
2.7 sSavGolElasticKernels1D depends on: 2.6 sSavGolKernel1D
2.8 sSavGolBufferConvolve1D depends on: built-in symbols only
2.9 sGrandiSequence depends on: built-in symbols only

3.1 sFileSelect depends on: built-in symbols only
3.2 sDirectoryDigest depends on: built-in symbols only
3.3 sBulkExport depends on: 4.3 sUniqueName

4.1 sHeadsOptionCheck: built-in symbols only
4.2 sBoole depends on: built-in symbols only

4.3 sUniqueName depends on: built-in symbols only

4.4 sReferTo depends on: built-in symbols only

4.5 sPushKeyUp depends on: built-in symbols only

4.6 sEmptyListsQ depends on: built-in symbols only

4.7 sFactorExactNumber depends on: built-in symbols only

4.8 sENextPow2 depends on: built-in symbols only

4.9 sENextPow2N depends on: built-in symbols only

4.10 sNextPow2 depends on: 4.2 sBoole

4.11 sNextPow2N depends on: built-in symbols only

4.12 sParseIdNumber depends on: built-in symbols only

Appendix C: Creating table of contents

The following four code cells below in this Appendix facilitate creating fresh table-of-contents notebooks. Their Cell ▷ Cell Properties ▷ Evaluatable property is removed since evaluation is normally undesirable. To update the table-of-contents notebooks, restore their evaluatability, and obey the instructions.

Hypertext TOC for this notebook

If you need a hyperlinked table of contents for comfortable work with this notebook:

- It is strongly recommended to start with a fresh session of *Mathematica*.
- Evaluate this cell for necessary settings:

```
In[•]:= Needs["AuthorTools`"]; nb = EvaluationNotebook[]; ts = "_TOC_";
        SetOptions[MakeContents, CellTagPrefix → "TOC:",
          SelectedCellStyles → {"BookChapterTitle", "Section"}];
```

- Before evaluating each of the following cells *save this notebook* if unsaved:

```
In[•]:= Paginate[nb];
```

```
In[•]:= MakeContents[nb, #, ContentsFileName →
            FileBaseName[NotebookFileName[]] <> ts <> # <> ".nb"] & /@
          {"Book", "BookCondensed"};
        (*Clear[ts,nb];*)
```

- If everything has succeeded, you are done, and can start using the buttons in Contents above.

Plain TOC for PDF export

If you only need a table of contents for printed version from the PDF export:

- Open the standard TOC using the Contents left button, then save it as plain text (File ▷ Save As…, choose Plain Text (*.txt) in the dialog) with the basename unchanged.
- Evaluate the following code and delete the preceding cells (keeping a full copy is desirable):

In[1223]:=

```
nb = EvaluationNotebook[]; ts = "_TOC_";
Import[StringDrop[NotebookFileName[], -3] <> ts <> "Book.txt"];
Drop[StringReplace[StringSplit[%, "\n"], "\t" → "... "], 2] // Column //
 Style[#, FontFamily → "Times", FontSize → 7] &
Clear[ts, nb];
```

Out[1225]=

Preface … 5
The history and future of Snapdragon … 5
The audience for this manual … 6
Disclaimer … 6
Acknowledgements … 7
Preliminaries … 8
Snapdragon application getting and installing … 8
The organization of the manual … 9
Error handling … 10
Batch evaluation … 13
List and array manipulations … 14
1.1 sListDecimate … 14
1.2 sListPush … 29
1.3 sListTruncate … 52
1.4 sRotateHalfLength … 59
1.5 sDimensionsBoxed … 112
1.6 sExprOccupancy … 117
1.7 sDiagonals … 120
Signal analysis … 126
2.1 sSamplingSpan … 126
2.2 sCircularlySample … 128
2.3 sCVectorDFTUnpack … 141
2.4 sRVectorCPack … 145
2.5 sRealFourier … 150
2.6 sSavGolKernel1D … 156
2.7 sSavGolElasticKernels1D … 166
2.8 sSavGolBufferConvolve1D … 170
2.9 sGrandiSequence … 198
Files, import, export … 203
3.1 sFileSelect … 203
3.2 sDirectoryDigest … 208
3.3 sBulkExport … 210
Miscellanea … 218
4.1 sHeadsOptionCheck … 218
4.2 sBoole … 221
4.3 sUniqueName … 222
4.4 sReferTo … 227
4.5 sPushKeyUp … 235
4.6 sEmptyListsQ … 240
4.7 sFactorExactNumber … 245
4.8 sENextPow2 … 255
4.9 sENextPow2N … 257
4.10 sNextPow2 … 260
4.11 sNextPow2N … 262
4.12 sParseIdNumber … 266
Appendices … 271
Appendix A: System−wide installation … 271
Appendix B: Dependencies … 272
Appendix C: Creating table of contents … 273
Appendix D: Initialization … 274
Bibliography … 276
Built−in symbols, guides and tutorials … 279
Notes … 283

Appendix D: Initialization

In this section, initialization cells are gathered. It is worth of emphasizing that all the initialization code is independent of Snapdragon and may be safely evaluated before Snapdragon is loaded.

infoAbout

As of *Mathematica* 12.0, new features in the Information symbol somewhat change the look of this manual. The following code modifies the appearance selectively depending on the version.

In[1227]:=

```
ClearAll[infoAbout]
infoAbout[s : _Symbol | _String, Optional[m_ ?NumberQ, 0.5]] :=
 Module[{v = $VersionNumber},
  Which[v ≤ 11.3, Information[s], True,
   If[s =!= "Snapdragon", Magnify[Information[s], m],
    Information[s, "Usage"]]]]
```

Bibliography

This section lists cited books, articles, and selected online resources. The "access statement," if present, means the date of the *last* visit to the relevant website, thus checking its availability.

[Baumann, 2005a] Gerd Baumann, 2005. *Mathematica*® for Theoretical Physics. Classical Mechanics and Nonlinear Dynamics. 2nd Edition. CD-ROM Included. Springer Science+Business Media, Inc., New York. ISBN 978-0387-01674-0.

[Baumann, 2005a] Gerd Baumann, 2005. *Mathematica*® for Theoretical Physics. Electrodynamics, Quantum Mechanics, General Relativity, and Fractals. 2nd Edition. CD-ROM Included. Springer Science+Business Media, Inc., New York. ISBN 978-0387-21933-2.

[Friedrich, 2013] Václav Friedrich, 2013. *Mathematica* na počítači pro nematematiky. Průvodce programem *Mathematica* pro studenty a další zájemce. VŠB — Technická univerzita Ostrava, Ekonomická fakulta, Ostrava. viii+254 pages. ISBN 978-80-248-3162-6, URL: http://aleph.nkp.cz/publ/skc/006/13/88/006138820.htm [Accessed 24 Feb 2019].

[Gregory, 2005] Phil C. Gregory, 2005. Bayesian Logical Data Analysis for the Physical Sciences. A Comparative Approach with *Mathematica*™ Support. Cambridge University Press, Cambridge. 488 pages. ISBN: 978-0-521-84150-4. URL: http://www.cambridge.org/9780521841504 [Accessed 27 Apr 2019].

[Hledík, 2019a] Stanislav Hledík's Tumblr account https://hlediks.tumblr.com/ [Accessed 16 Apr 2019].

[Hledík, 2019b] Stanislav Hledík, 2019. Private *Mathematica* Dropbox repository [online]. Available at: https://www.dropbox.com/sh/bfzioelpvvuumob/AACKKAxqDC_gcrGpP8HLrTg9a/AddOns/Repository?dl=0 [Accessed 16 Apr 2019].

[Horvát, 2016] Szabolcs Horvát, 2016. PacletInfo.m documentation project [online]. Mathematica Stack Exchange. Available at: https://mathematica.stackexchange.com/q/132064 [Accessed 22 Feb 2019].

[Horvát, 2017a] Szabolcs Horvát, 2017. How to distribute *Mathematica* packages as paclets? [online]. Mathematica Stack Exchange. Available at: https://mathematica.stackexchange.com/q/131101 [Accessed 22 Feb 2019].

[Horvát, 2017b] Szabolcs Horvát, 2017. How can I install packages distributed as .paclet files? [online]. Mathematica Stack Exchange. Available at: https://mathematica.stackexchange.com/q/141887 [Accessed 22 Feb 2019].

[Horvát, 2017c] Szabolcs Horvát, 2017. Install paclets into $BaseDirectory on multi-user system

[online]. Mathematica Stack Exchange. Available at: https://mathematica.stackexchange.-com/q/155122 [Accessed 25 Feb 2019].

[Jason, 2014] Jason, 2014. Is there a function for nearest power of 2 in Mathematica like nextpow2 in MATLAB? [online]. Mathematica Stack Exchange. Available at: https://-mathematica.stackexchange.com/q/56050 [Accessed 10 Sep 2018].

[Kubetta, 2012] Adam Kubetta, 2012. *Mathematica* jako textový editor [online]. Archiv webu "Pravidelná setkání uživatelů systému Wolfram *Mathematica*". Available at: http://www.-mathematica.cz/setkani.php [Accessed 26 Apr 2019].

[Lorenc, 2013] Miroslav Lorenc, 2013. Ověření správnosti rodného čísla. Předmět 3MA381: "Manažerská informatika – kancelářské aplikace" [online]. Available at: https://lorenc.in-fo/3MA381/overeni-spravnosti-rodneho-cisla.htm [Accessed 26 Apr 2019].

[Lyons, 2011] Richard G. Lyons, 2011. Understanding Digital Signal Processing. 3rd Edition. Prentice Hall, Upper Saddle River, NJ. 992 pages. ISBN: 978-0-13-702741-5. URL: https://www.pearson.com/us/higher-education/program/Lyons-Understanding-Digital-Signal-Processing-3rd-Edition/PGM202823.html [Accessed 27 Apr 2019].

[Mangano, 2010] Salvatore Mangano, 2010. Mathematica Cookbook. Building Blocks for Sci-ence, Engineering, Finance, Music, and More. 1st Edition. O'Reilly, Sebastopol. xxiv+801 pages. ISBN: 978-0-596-52099-1, URL: https://books.google.cz/books/about/-Mathematica_Cookbook.html?id=BkDxC_O1WisC [Accessed 24 Feb 2019].

[*Mathematica* Stack Exchange, 2019] Mathematica Stack Exchange, 2019. A Question & Answer site for users of Wolfram *Mathematica* and the *Wolfram Language* [online]. Available at: https://mathematica.stackexchange.com/ [Accessed 30 Apr 2019].

[MathWorks, 2019] MathWorks, 2019. nextpow2 – Exponent of next higher power of 2 [online]. MathWorks Documentation Site. Available at: https://www.mathworks.com/help/mat-lab/ref/nextpow2.html [Accessed 10 Sep 2018].

[Press et al., 1992] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, 1992. Numerical Recipes in C. The Art of Scientific Computing. 2nd Edition. Cambridge University Press, Cambridge, USA. xxvi+925 pages. ISBN 0-521-43108-5, URL: http://numerical.recipes/ [Accessed 16 Apr 2019].

[Říha et al., 2011] Jan Říha, František Látal, Veronika Kainzová, Vratislava Mošová, Ivo Vyšín, Filip Švrček, and Lukáš Richterek, 2011. Software Mathematica v přírodních vědách a ekonomii. Univerzita Palackého v Olomouci. Available at: http://mathscience.upol.cz/ma-terialy/software_mathematica.pdf [Accessed 27 Apr 2019].

[Rila, 2014] Luciano Rila, 2014. When things get weird with infinite sums [online]. Plus Maga-zine: a production of the Millennium Mathematics Project. Available at: https://plus.math-s.org/content/when-things-get-weird-infinite-sums [Accessed 26 Apr 2019].

[Ruskeepää, 2009] Heikki Ruskeepää, 2009. Mathematica Navigator. Mathematics, Statistics, and Graphics. 3rd Edition. Academic Press (an imprint of Elsevier), Amsterdam. 1136 pages. ISBN 978-0-12-374164-6, URL: http://library.wolfram.com/infocenter/Books/7352/

[Accessed 24 Feb 2019].

[Savitzky & Golay, 1964] Abraham Savitzky and Marcel J. E. Golay, 1964. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Anal. Chem.*, **36**(8), pp. 1627–1639. DOI: 10.1021/ac60214a047 [Accessed 24 Apr 2019].

[Schafer, 2011] Ronald W. Schafer, 2011. What Is a Savitzky–Golay Filter? *IEEE Signal Processing Magazine*, **28**(4), pp. 111–117. DOI: 10.1109/MSP.2011.941097 [Accessed 24 Apr 2019].

[Shifrin, 2009] Leonid Shifrin, 2009. *Mathematica*® Programming—an advanced introduction. A moderately paced practical tutorial for *Mathematica* programming language [online]. Available at: http://www.mathprogramming-intro.org/ [Accessed 27 Apr 2019].

[Wellin et al., 2005] Paul R. Wellin, R. J. Gaylord, and S. N. Kamin, 2005. An Introduction to Programming with *Mathematica*. 3rd Edition. Cambridge University Press, Cambridge. xx+550 pages. ISBN 0-521-84678-1, URL: http://library.wolfram.com/infocenter/Books/5169/ [Accessed 24 Feb 2019].

[Wellin & Calkins, 2011] Paul R. Wellin and Harry Calkins, 2011. M101: A First Course with *Mathematica*®. Certified *Mathematica* Training. Version 2.3. Wolfram Education Group.

[Wellin, 2013] Paul R. Wellin, 2013. Programming with *Mathematica*. An Introduction. 1st Edition. Cambridge University Press, Cambridge. 728 pages. ISBN 978-1-107-00946-2. URL: http://www.cambridge.org/wellin [Accessed 24 Feb 2019].

[Wikipedia, 2019a] Wikipedia, 2019. National identification number – Czech Republic and Slovakia [online]. Available at: https://en.wikipedia.org/wiki/National_identification_number #Czech _Republic _and _Slovakia [Accessed 26 Apr 2019].

[Wolfram, 2017] Stephen Wolfram, 2017. An Elementary Introduction to the *Wolfram Language*. 2nd Edition. Wolfram Media. 339 pages. ISBN 978-1-944-18305-9. URL: http://www.wolfram.com/language/elementary-introduction/2nd-ed/ [Accessed 27 Apr 2019].

Built-in symbols, guides and tutorials

This section collects all built-in symbols, *Wolfram Language* & System Documentation Center (WLSDC) guides, tutorials and menu items referenced in the text (but not those used in the code). Every listed symbol, guide, tutorial and menu item name is linked to the corresponding local WLSDC page, the adjacent [URI] button is linked to a corresponding online WLSDC website page. Items are alphabetically sorted by name regardless of being a symbol, a guide, a tutorial, or a menu item.

1. Symbol $BaseDirectory [reference.wolfram.com/language/ref/$BaseDirectory].
2. Symbol $Canceled [reference.wolfram.com/language/ref/$Canceled].
3. Symbol $Failed [reference.wolfram.com/language/ref/$Failed].
4. Symbol $HomeDirectory [reference.wolfram.com/language/ref/$HomeDirectory].
5. Symbol $MachineEpsilon [reference.wolfram.com/language/ref/$MachineEpsilon].
6. Symbol $MachinePrecision [reference.wolfram.com/language/ref/$MachinePrecision].
7. Symbol $MaxMachineNumber [reference.wolfram.com/language/ref/$MaxMachineNumber].
8. Symbol $MinMachineNumber [reference.wolfram.com/language/ref/$MinMachineNumber].
9. Symbol $TemporaryDirectory [reference.wolfram.com/language/ref/$TemporaryDirectory].
10. Symbol $Version [reference.wolfram.com/language/ref/$Version].
11. Symbol Abs [reference.wolfram.com/language/ref/Abs].
12. Symbol AbsoluteTime [reference.wolfram.com/language/ref/AbsoluteTime].
13. Symbol Activate [reference.wolfram.com/language/ref/Activate].
14. Symbol ActiveStyle [reference.wolfram.com/language/ref/ActiveStyle].
15. Symbol All [reference.wolfram.com/language/ref/All].
16. Symbol Apply [reference.wolfram.com/language/ref/Apply].
17. Symbol Array [reference.wolfram.com/language/ref/Array].
18. Symbol ArrayQ [reference.wolfram.com/language/ref/ArrayQ].
19. Symbol AspectRatio [reference.wolfram.com/language/ref/AspectRatio].
20. Symbol Association [reference.wolfram.com/language/ref/Association].
21. Symbol Automatic [reference.wolfram.com/language/ref/Automatic].
22. Symbol BlankNullSequence [reference.wolfram.com/language/ref/BlankNullSequence].
23. Symbol Boole [reference.wolfram.com/language/ref/Boole].
24. Symbol Cases [reference.wolfram.com/language/ref/Cases].

25. Guide CellMenu [reference.wolfram.com/language/guide/CellMenu].
26. Menu item CellProperties [reference.wolfram.com/language/ref/menuitem/CellProperties].
27. Symbol ChebyshevT [reference.wolfram.com/language/ref/ChebyshevT].
28. Tut. Combinatorica [reference.wolfram.com/language/Combinatorica/tutorial/Combinatorica].
29. Symbol ConstantArray [reference.wolfram.com/language/ref/ConstantArray].
30. Symbol Cos [reference.wolfram.com/language/ref/Cos].
31. Symbol Count [reference.wolfram.com/language/ref/Count].
32. Symbol CreateUUID [reference.wolfram.com/language/ref/CreateUUID].
33. Symbol DeleteCases [reference.wolfram.com/language/ref/DeleteCases].
34. Symbol Depth [reference.wolfram.com/language/ref/Depth].
35. Symbol Diagonal [reference.wolfram.com/language/ref/Diagonal].
36. Symbol Dimensions [reference.wolfram.com/language/ref/Dimensions].
37. Symbol Downsample [reference.wolfram.com/language/ref/Downsample].
38. Symbol DownValues [reference.wolfram.com/language/ref/DownValues].
39. Menu item Evaluatable [reference.wolfram.com/language/ref/menuitem/Evaluatable].
40. M. it. EvaluateNotebook [reference.wolfram.com/language/ref/menuitem/EvaluateNotebook].
41. Guide EvaluationMenu [reference.wolfram.com/language/guide/EvaluationMenu].
42. Symbol Export [reference.wolfram.com/language/ref/Export].
43. Symbol FactorInteger [reference.wolfram.com/language/ref/FactorInteger].
44. Symbol False [reference.wolfram.com/language/ref/False].
45. Symbol FileHash [reference.wolfram.com/language/ref/FileHash].
46. Guide FileMenu [reference.wolfram.com/language/guide/FileMenu].
47. Symbol FileNameJoin [reference.wolfram.com/language/ref/FileNameJoin].
48. Symbol FileNames [reference.wolfram.com/language/ref/FileNames].
49. Symbol FirstCase [reference.wolfram.com/language/ref/FirstCase].
50. Symbol FirstPosition [reference.wolfram.com/language/ref/FirstPosition].
51. Symbol FixedPoint [reference.wolfram.com/language/ref/FixedPoint].
52. Symbol Fold [reference.wolfram.com/language/ref/Fold].
53. Symbol FoldList [reference.wolfram.com/language/ref/FoldList].
54. Symbol Fourier [reference.wolfram.com/language/ref/Fourier].
55. Symbol FourierDCT [reference.wolfram.com/language/ref/FourierDCT].
56. Symbol FourierDST [reference.wolfram.com/language/ref/FourierDST].
57. Symbol FourierParameters [reference.wolfram.com/language/ref/FourierParameters].
58. Symbol FreeQ [reference.wolfram.com/language/ref/FreeQ].
59. Symbol Function [reference.wolfram.com/language/ref/Function].
60. Symbol General [reference.wolfram.com/language/ref/General].
61. Symbol Get [reference.wolfram.com/language/ref/Get].
62. Symbol GoldenRatio [reference.wolfram.com/language/ref/GoldenRatio].

63. Symbol Heads [reference.wolfram.com/language/ref/Heads].
64. Symbol Hyperlink [reference.wolfram.com/language/ref/Hyperlink].
65. Symbol If [reference.wolfram.com/language/ref/If].
66. Symbol IgnoreCase [reference.wolfram.com/language/ref/IgnoreCase].
67. Symbol Inactivate [reference.wolfram.com/language/ref/Inactivate].
68. Symbol Infinity [reference.wolfram.com/language/ref/Infinity].
69. Symbol Information [reference.wolfram.com/language/ref/Information].
70. Menu item Install [reference.wolfram.com/language/ref/menuitem/Install].
71. Symbol Integer [reference.wolfram.com/language/ref/Integer].
72. Symbol Interval [reference.wolfram.com/language/ref/Interval].
73. Symbol InverseFourier [reference.wolfram.com/language/ref/InverseFourier].
74. Symbol Length [reference.wolfram.com/language/ref/Length].
75. Symbol Level [reference.wolfram.com/language/ref/Level].
76. Symbol List [reference.wolfram.com/language/ref/List].
77. Symbol Listable [reference.wolfram.com/language/ref/Listable].
78. Symbol ListConvolve [reference.wolfram.com/language/ref/ListConvolve].
79. Symbol ListLinePlot [reference.wolfram.com/language/ref/ListLinePlot].
80. Guide ListManipulation [reference.wolfram.com/language/guide/ListManipulation].
81. Symbol ListPlot [reference.wolfram.com/language/ref/ListPlot].
82. Symbol Magnify [reference.wolfram.com/language/ref/Magnify].
83. Symbol Map [reference.wolfram.com/language/ref/Map].
84. Symbol MapAll [reference.wolfram.com/language/ref/MapAll].
85. Symbol MapIndexed [reference.wolfram.com/language/ref/MapIndexed].
86. Symbol MemberQ [reference.wolfram.com/language/ref/MemberQ].
87. Symbol Method [reference.wolfram.com/language/ref/Method].
88. Symbol Mod [reference.wolfram.com/language/ref/Mod].
89. Symbol Needs [reference.wolfram.com/language/ref/Needs].
90. Menu item New [reference.wolfram.com/language/ref/menuitem/New].
91. Symbol None [reference.wolfram.com/language/ref/None].
92. Symbol NotebookDirectory [reference.wolfram.com/language/ref/NotebookDirectory].
93. Symbol Null [reference.wolfram.com/language/ref/Null].
94. Symbol NumberQ [reference.wolfram.com/language/ref/NumberQ].
95. Symbol NumericQ [reference.wolfram.com/language/ref/NumericQ].
96. Symbol Options [reference.wolfram.com/language/ref/Options].
97. Symbol OptionsPattern [reference.wolfram.com/language/ref/OptionsPattern].
98. Symbol Padding [reference.wolfram.com/language/ref/Padding].
99. Symbol PadLeft [reference.wolfram.com/language/ref/PadLeft].
100. Symbol PadRight [reference.wolfram.com/language/ref/PadRight].

101. Symbol Partition [reference.wolfram.com/language/ref/Partition].
102. Symbol Pick [reference.wolfram.com/language/ref/Pick].
103. Symbol Plot [reference.wolfram.com/language/ref/Plot].
104. Symbol Position [reference.wolfram.com/language/ref/Position].
105. Symbol Positive [reference.wolfram.com/language/ref/Positive].
106. Symbol Power [reference.wolfram.com/language/ref/Power].
107. Symbol Quiet [reference.wolfram.com/language/ref/Quiet].
108. Symbol Range [reference.wolfram.com/language/ref/Range].
109. Symbol RealAbs [reference.wolfram.com/language/ref/RealAbs].
110. Symbol RealExponent [reference.wolfram.com/language/ref/RealExponent].
111. Symbol Replace [reference.wolfram.com/language/ref/Replace].
112. Symbol ReplacePart [reference.wolfram.com/language/ref/ReplacePart].
113. Symbol Riffle [reference.wolfram.com/language/ref/Riffle].
114. Symbol SameQ [reference.wolfram.com/language/ref/SameQ].
115. Menu item SaveAs [reference.wolfram.com/language/ref/menuitem/SaveAs].
116. Symbol SavitzkyGolayMatrix [reference.wolfram.com/language/ref/SavitzkyGolayMatrix].
117. Symbol Scan [reference.wolfram.com/language/ref/Scan].
118. Symbol Select [reference.wolfram.com/language/ref/Select].
119. Symbol SetOptions [reference.wolfram.com/language/ref/SetOptions].
120. Symbol Subscript [reference.wolfram.com/language/ref/Subscript].
121. Symbol Switch [reference.wolfram.com/language/ref/Switch].
122. Symbol SyntaxInformation [reference.wolfram.com/language/ref/SyntaxInformation].
123. Symbol Table [reference.wolfram.com/language/ref/Table].
124. Symbol Tally [reference.wolfram.com/language/ref/Tally].
125. Symbol True [reference.wolfram.com/language/ref/True].
126. Symbol Values [reference.wolfram.com/language/ref/Values].
127. Symbol Which [reference.wolfram.com/language/ref/Which].
128. Symbol WorkingPrecision [reference.wolfram.com/language/ref/WorkingPrecision].

Notes

Some general notes and remarks are collected here.

1. The Qualcomm® Snapdragon™ Mobile Platform web page: www.qualcomm.com/snapdragon [Accessed 26 Apr 2019].

---

2. Gardenia: Antirrhinum majus (Snapdragon): www.gardenia.net/plant/antirrhinum-majus-snapdragon [Accessed 26 Apr 2019].

---

3. The 6-digit release number R encompasses the date as YYdddF with YY being a short year, ddd a day of that year (001..365 or 366 in case of a leap year), and F a decimal fraction of that day (0..9).

---

4. Do *not* use the *mouse right click* and *Save Link As*… This would not work.

---

5. The directory $UserBaseDirectory (in which user-specific files to be loaded by the Wolfram System are conventionally placed) is platform dependent—see the documentation's Details for more information.

---

6. Length@Names["Snapdragon`s*"] should return the major version number M.
