

# Testing my Mathematica DSP package

copyright Nasser M. Abbasi

Last updated Nov 14, 2010

```
In[403]:= Get["G:\\nabbasi\\data\\nabbasi_web_Page\\my_notes\\my_dsp_mathematica_package\\dsp.m"]
```

```
In[404]:= ? dsp`*
```

▼ dsp`

butterd	linspace
butterToBandPass	makePulseTrain
butterToBandStop	matrix2DLexOrdering
butterToHighPass	matrix2DredBlackPositions
butterToLowPass	numIt
centerImageData	numItse
diagonalBlockMatrix	plotFourierTransform
diagonalDominantMatrixQ	poissonMatrixOnSquareDirichlet
fcPulseTrain	quantizationLevel
formatBinary	str
getButterworthPolynomial	toOffsetBinary
getFourierApproximation	toOnesComplement
getMagnitudeOfPulseTrainFourierCoeff	toSignMagnitude
getPhaseOfPulseTrainFourierCoeff	toTwoComplement
getPositiveAndNegativeTerms	

## matrix2DredBlackPositions

```
In[363]:= ? dsp`matrix2DredBlackPositions
```

matrix2DLexOrdering[nRow,nCol] returns {r,b}, where r is a list of indices in a matrix of dimensions[nRow,nCol] where the red balls are, and b is list of indices where black balls are. These are the red/black positions to use for Gauss–Seidel red, black algorithm

### Example

```
A=Table[0,{i,3},{j,3}];
dsp`matrix2DLexOrdering[A]
Out[310]= {{{1,3},{2,3},{3,3}},{(1,2),(2,2),(3,2)},{(1,1),(2,1),(3,1)}}
```

```
In[388]:= nRow = 3;
nCol = 3;
A = Table[0, {i, nRow}, {j, nCol}];
{r, b} = dsp`matrix2DredBlackPositions [nRow, nCol];
(A[[Sequence @@ #]] = 1) & /@ r;
(A[[Sequence @@ #]] = -1) & /@ b;
MatrixForm[A]
```

```
Out[394]/MatrixForm=

$$\begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix}$$

```

```
In[395]:= nRow = 2;
nCol = 5;
A = Table[0, {i, nRow}, {j, nCol}];
{r, b} = dsp`matrix2DredBlackPositions [nRow, nCol];
(A[[Sequence @@ #]] = 1) & /@ r;
(A[[Sequence @@ #]] = -1) & /@ b;
MatrixForm[A]
```

```
Out[401]/MatrixForm=

$$\begin{pmatrix} 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 \end{pmatrix}$$

```

## matrix2DLexOrdering

```
In[402]:= ? dsp`matrix2DLexOrdering
```

matrix2DLexOrdering[mat] returns {A,B}, where A is matrix of the same as as mat, which contains in each entry, the coordinates of each cell in the matrix in Lexicographical order and B contains the sequential Lexicographical order

### Example

```
A=Table[0,{i,3},{j,3}];
```

```
dsp`matrix2DLexOrdering[A]
```

```
Out[310]= {{{1,3},{2,3},{3,3}},{{1,2},{2,2},{3,2}},{{1,1},{2,1},{3,1}}}
```

```
In[329]:= (c = Table[0, {i, 3}, {j, 1}]) // MatrixForm
```

```
Out[329]//MatrixForm=
```

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

```
In[330]:= {A, B} = dsp`matrix2DLexOrdering [c];  
A // MatrixForm
```

```
Out[331]//MatrixForm=
```

$$\begin{pmatrix} \begin{pmatrix} 1 \\ 3 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 1 \end{pmatrix} \end{pmatrix}$$

```
In[332]:= B // MatrixForm
```

```
Out[332]//MatrixForm=
```

$$\begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$$

```
In[336]:= (c = Table[0, {i, 3}, {j, 2}]) // MatrixForm  
{A, B} = dsp`matrix2DLexOrdering [c];  
A // MatrixForm
```

```
Out[336]//MatrixForm=
```

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

```
Out[338]//MatrixForm=
```

$$\begin{pmatrix} \begin{pmatrix} 1 \\ 3 \end{pmatrix} & \begin{pmatrix} 2 \\ 3 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 2 \end{pmatrix} & \begin{pmatrix} 2 \\ 2 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 2 \\ 1 \end{pmatrix} \end{pmatrix}$$

```
In[339]:= B // MatrixForm
```

```
Out[339]//MatrixForm=
```

$$\begin{pmatrix} 5 & 6 \\ 3 & 4 \\ 1 & 2 \end{pmatrix}$$

```
In[340]:= (c = Table[0, {i, 3}, {j, 3}]) // MatrixForm
{A, B} = dsp`matrix2DLexOrdering[c];
A // MatrixForm
```

```
Out[340]//MatrixForm=
```

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

```
Out[342]//MatrixForm=
```

$$\begin{pmatrix} \begin{pmatrix} 1 \\ 3 \end{pmatrix} & \begin{pmatrix} 2 \\ 3 \end{pmatrix} & \begin{pmatrix} 3 \\ 3 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 2 \end{pmatrix} & \begin{pmatrix} 2 \\ 2 \end{pmatrix} & \begin{pmatrix} 3 \\ 2 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 1 \end{pmatrix} & \begin{pmatrix} 2 \\ 1 \end{pmatrix} & \begin{pmatrix} 3 \\ 1 \end{pmatrix} \end{pmatrix}$$

```
In[343]:= B // MatrixForm
```

```
Out[343]//MatrixForm=
```

$$\begin{pmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}$$

## centerImageData

```
? dsp`centerImageData
```

centerImageData[im] centers the image by multiplying each pixel by  $(-1)^{(i+j)}$  where  $i, j$  are the row number and column number of the pixel in normal matrix meaning of row and column. This is useful to do before taking the FFT of the image.

```

img = Import["ExampleData/lena.tif"];
ImageDimensions[img] (*look at image*)

ImageChannels[img] (*see how many channels*)

data = ImageData[img]; (*get data*)
{nRow, nCol, nChannel} = Dimensions[data]

Map[Image[data[[All, All, #]]] &, Range[1, nChannel]] (*look at each channel*)

d = data[[All, All, 2]]; (*get channel 2*) Image[d, ImageSize -> 400]
{150, 116}
3
{116, 150, 3}

```



```

(*center Image first*)
d = dsp`centerImageData[d];

Dimensions[d]

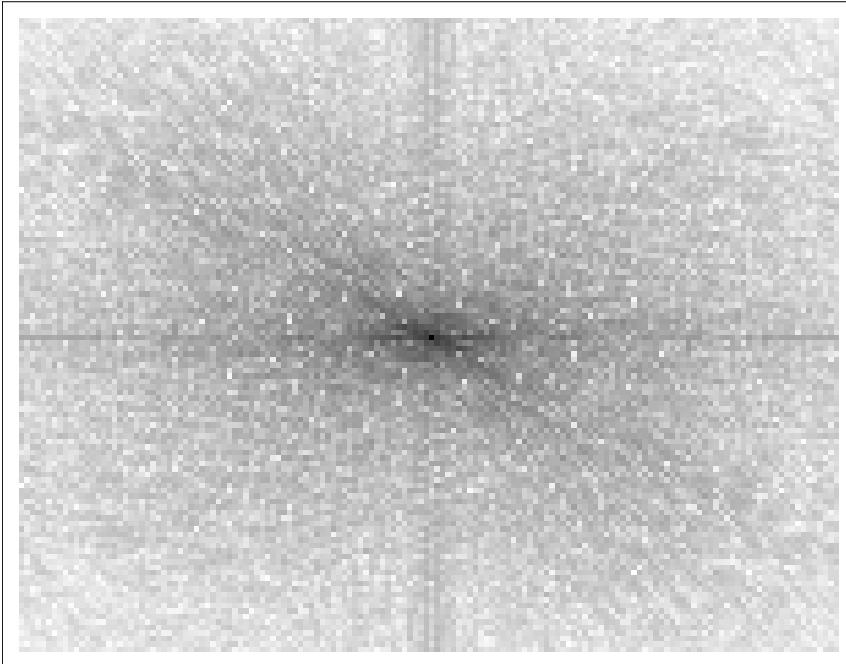
d[[2, 2]]

0.6549019607843137

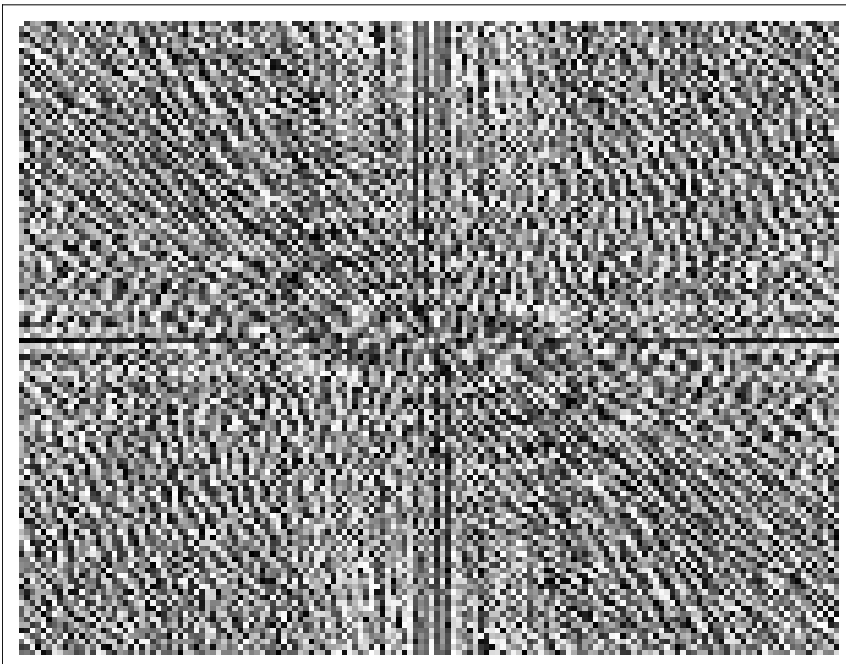
```

```
(*make FFT and look at spectrum and phase*)  
fw = Fourier[d, FourierParameters -> {1, 1}];  
  
fudgeFactor = 100; (*adjust for better viewing as needed*)  
ArrayPlot[(fudgeFactor * Log[1 + Abs@fw]), PlotLabel -> "Magnitude spectrum", ImageSize -> 400]  
ArrayPlot[Arg[fw], PlotLabel -> "Phase spectrum", ImageSize -> 400]
```

Magnitude spectrum



Phase spectrum



---

## diagonalBlockMatrix

? dsp`diagonalBlockMatrix

diagonalBlockMatrix[block,n] gives a diagonal square matrix with the block along the diagonal repeated n times

example:

In[14]:= block={{1,2},{3,4}}

Out[14]= {{1,2},{3,4}}

Normal@dsp`diagonalBlockMatrix[block,4]

Out[15]=

{{1,2,0,0,0,0,0,0},{3,4,0,0,0,0,0,0},{0,0,1,2,0,0,0,0},{0,0,3,4,0,0,0,0},{0,0,0,0,1,2,0,0},{0,0,0,0,3,4,0,0},{0,0,0,0,0,0,1,2},{0,0,0,0,0,0,3,4}}

```
(block = {{1, 2}, {3, 4}}) // MatrixForm
dsp`diagonalBlockMatrix [block, 4];
MatrixForm [Normal [%]]
```

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 4 \end{pmatrix}$$


---

## poissonMatrixOnSquareDirichlet

? dsp`poissonMatrixOnSquareDirichlet

poissonMatrixOnSquareDirichlet[n,m] gives the discretization A matrix for  $\partial_{x,x}u + \partial_{y,y}u$

using Laplacian stencil of size m (currently only m=5 is supported). n is the number of grid points along one dimension of the square. Hence, for n=3, this means the square has 9 internal grid points. These are the internal grid points only, as this is Dirichlet boundary conditions problem

```
dsp`poissonMatrixOnSquareDirichlet [3, 5] // MatrixForm
```

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}$$

```
dsp`poissonMatrixOnSquareDirichlet [2, 5] // MatrixForm
```

$$\begin{pmatrix} -4 & 1 & 1 & 0 \\ 1 & -4 & 0 & 1 \\ 1 & 0 & -4 & 1 \\ 0 & 1 & 1 & -4 \end{pmatrix}$$

```
dsp`poissonMatrixOnSquareDirichlet [4, 5] // MatrixForm
```

$$\begin{pmatrix} -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -4 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -4 \end{pmatrix}$$

---

## diagonalDominantMatrixQ

? dsp`diagonalDominantMatrixQ

diagonalDominantMatrixQ[A] gives True if matrix is diagonal dominant matrix, and False otherwise. The following are available options

strict->{True,False}

strict->False, is the default. The check is made using " $\geq$ " for dominance of diagonal.

strict->True, the check is made using " $>$ " for dominance of diagonal

Examples:

```
<<dsp`
mat={{4,4},{4,4}}
```

```
dsp`diagonalDominantMatrixQ[mat,strict->False]
Out[27]= True
```

```
dsp`diagonalDominantMatrixQ[mat,strict->True]
Out[31]= False
```

```
dsp`diagonalDominantMatrixQ[mat]
Out[38]= True
```

---

## numItsec

? dsp`numItse

numItse[num,w,d,fontsize] formats a number in scientific form.

num is the number to format

w is the total width of the field including the decimal point

d is width of field to right of decimal point

fontsize is the font size to use

EXAMPLE

```
dsp`numItse[234.4,6,4,10,12]
```

2.3440e2

```
dsp`numItse [RandomReal [1], 5, 3, 10]
```

```
6.187e-1
```

```
dsp`numItse [RandomReal [10], 5, 3, 10]
```

```
9.330e
```

## linspace

```
? dsp`linspace
```

generate linearly spaced list.

`linspace(x1, x2)` generate list of 100 equally spaced points between x1 and x2.

`linspace(x1, x2, N)` generate N equally spaced points between x1 and x2. if  $N < 2$ , it returns x2.

```
dsp`linspace [2, 3, 5]
```

```
{2,  $\frac{9}{4}$ ,  $\frac{5}{2}$ ,  $\frac{11}{4}$ , 3}
```

```
dsp`linspace [2 + I, 3, 5]
```

```
{2 + i,  $\frac{9}{4} + \frac{3i}{4}$ ,  $\frac{5}{2} + \frac{i}{2}$ ,  $\frac{11}{4} + \frac{i}{4}$ , 3}
```

```
dsp`linspace [2 + I, 3, 5] // N
```

```
{2. + 1. i, 2.25 + 0.75 i, 2.5 + 0.5 i, 2.75 + 0.25 i, 3.}
```

```
dsp`linspace [2, 3, 2] // N
```

```
{2., 3.}
```

```
dsp`linspace [2, 2.1, 10] // N
```

```
{2., 2.0111111111111111, 2.0222222222222222, 2.0333333333333333, 2.0444444444444443, 2.0555555555555556, 2.0666666666666667, 2.0777777777777778, 2.0888888888888889, 2.1}
```

## str

```
? dsp`str
```

`str[expr]` removes \$nnn from the expression. Used for printing from inside a function

```
test := Module[{a = 4}, Print["a=", dsp`str[a]];
```

```
test
```

```
a=4
```

---

## formatBinary

? dsp`formatBinary

formatBinary[num,nBits]

returns a string representation of the number num in base 2. nBits is the number of bits available to use for displaying the final result.

EXAMPLE

```
dsp`formatBinary[4,4]
```

```
0100
```

```
dsp`formatBinary[4,8]
```

```
0000 0100
```

```
dsp`formatBinary [4, 4]
```

```
0100
```

```
dsp`formatBinary [4, 8]
```

```
0000 0100
```

```
dsp`formatBinary [127, 8]
```

```
0111 1111
```

```
dsp`formatBinary [127, 16]
```

```
0000 0000 0111 1111
```

```
dsp`formatBinary [30, 16]
```

```
0000 0000 0001 1110
```

---

## toTwoComplement

? dsp`toTwoComplement

toTwoComplement[x,mp,nBits] returns {delta,xQuantized,code}

This function calculates the 2 complement value (in decimal) of the analog value x after quantization. mp is the maximum level allowed for the quantization. nBits is the number of bits allowed to store the final result in. The code is the two's complement value generated. To convert this value to binary representation, use the function formatBinary[] in this package.

```

mp = 16;
nBits = 8;
x = -8.7
{delta, quantized, code} = dsp`toTwoComplement [x, mp, nBits]
formatBinary [code, nBits]
-8.7
{0.125, -8.75, 186}
1011 1010

mp = 1;
nBits = 4;
x = -8.7
{delta, quantized, code} = dsp`toTwoComplement [x, mp, nBits]
formatBinary [code, nBits]
-8.7
{0.125, -1., 8}
1000

```

---

## toOffsetBinary

? dsp`toOffsetBinary

toOffsetBinary[x,mp,nBits] returns {delta,xQuantized,code}

This function calculates the offset binary value (in decimal) of the analog value x after quantization. mp is the maximum level allowed for the quantization. nBits is the number of bits allowed to store the final result in. The code is the offset binary value generated. To convert this value to binary representation, use the function formatBinary[] in this package.

Example

```

nBits=8;
mp=16;
x=-8.7;
{delta,xQuantized,code}=toOffsetBinary[x,mp,nBits]

```

```

mp = 16;
nBits = 8;
x = -8.7
{delta, quantized, code} = dsp`toOffsetBinary [x, mp, nBits]
formatBinary [code, nBits]
-8.7
{0.125, -8.75, 58}
0011 1010

```

```

mp = 1;
nBits = 4;
x = 1 / 15.
{delta, quantized, code} = dsp`toOffsetBinary[x, mp, nBits]
formatBinary[code, nBits]

0.06666666666666667

{0.125, 0.125, 9}

1001

mp = 1;
nBits = 4;
x = -100;
{delta, quantized, code} = dsp`toOffsetBinary[x, mp, nBits]
formatBinary[code, nBits]

{0.125, -1., 0}

0000

```

---

## toSignMagnitude

? dsp`toSignMagnitude

toSignMagnitude[x,mp,nBits] returns {delta,xQuantized,code}

This function calculates the sign magnitude value (in decimal) of the analog value x after quantization. mp is the maximum level allowed for the quantization. nBits is the number of bits allowed to store the final result in. The code is the sign magnitude value generated. To convert this value to binary representation, use the function formatBinary[] in this package.

### Example

```

nBits=8;
mp=16;
x= 8.7;
{delta,xQuantized,code}=toSignMagnitude[x,mp,nBits]

```

```

mp = 16;
nBits = 8;
x = -8.7
{delta, quantized, code} = dsp`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]

-8.7

{0.125, -8.75, 198}

1100 0110

```

```
mp = 10;
nBits = 8;
x = -6.5;
{delta, quantized, code} = dsp`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]

{0.078125, 83, 211}

1101 0011

mp = 10;
nBits = 8;
x = 6.5;
{delta, quantized, code} = dsp`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]

{0.078125, 83, 83}

0101 0011

mp = 10;
nBits = 8;
x = 600;
{delta, quantized, code} = dsp`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]

{0.078125, 127, 127}

0111 1111

mp = 10;
nBits = 8;
x = -600;
{delta, quantized, code} = dsp`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]

{0.078125, -9.921875, 255}

1111 1111

mp = 16;
nBits = 8;
x = -8.7;
{delta, quantized, code} = dsp`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]

{0.125, -8.75, 198}

1100 0110
```

```

mp = 10;
nBits = 8;
x = 600;
{delta, quantized, code} = dsp`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]

{0.078125, 9.921875, 127}

0111 1111

```

---

## toOnesComplement

? dsp`toOnesComplement

toOnesComplement[x,mp,nBits] returns {delta,xQuantized,code}

This function calculates the ones complement value (in decimal) of the analog value x after quantization. mp is the maximum level allowed for the quantization. nBits is the number of bits allowed to store the final result in. The code is the ones complement value generated. To convert this value to binary representation, use the function formatBinary[] in this package.

```

mp = 10;
nBits = 8;
x = N[-6.5]
{delta, quantized, code} = dsp`toOnesComplement[x, mp, nBits]
formatBinary[code, nBits]

-6.5

{0.078125, -6.484375, 172}

1010 1100

mp = 1;
nBits = 4;
x = -1 / 17.
{delta, quantized, code} = dsp`toOnesComplement[x, mp, nBits]
dsp`formatBinary[code, nBits]

-0.058823529411764705

{0.125, 0., 15}

1111

mp = 1;
nBits = 4;
x = 13 / 17.
{delta, quantized, code} = dsp`toOnesComplement[x, mp, nBits]
formatBinary[code, nBits]

0.7647058823529411

{0.125, 0.75, 6}

0110

```

```
mp = 16;  
nBits = 8;  
x = -8.7;  
{delta, quantized, code} = dsp`toOnesComplement[x, mp, nBits]  
formatBinary[code, nBits]  
  
{0.125, -8.75, 185}  
  
1011 1001  
  
mp = 1;  
nBits = 4;  
x = 100;  
{delta, quantized, code} = dsp`toOnesComplement[x, mp, nBits]  
formatBinary[code, nBits]  
  
{0.125, 0.875, 7}  
  
0111  
  
mp = 1;  
nBits = 4;  
x = -100;  
{delta, quantized, code} = dsp`toOnesComplement[x, mp, nBits]  
formatBinary[code, nBits]  
  
{0.125, -0.875, 8}  
  
1000
```

```
delta = 1 / 16.;
data = Table[{n * delta, formatBinary[dsp`toOnesComplement[n * delta, 1, 4][[3]], 4]},
  {n, 2^4 - 1, -2^4, -1}] // TableForm
```

0.9375	0111
0.875	0111
0.8125	0110
0.75	0110
0.6875	0110
0.625	0101
0.5625	0100
0.5	0100
0.4375	0100
0.375	0011
0.3125	0010
0.25	0010
0.1875	0010
0.125	0001
0.0625	0000
0	0000
-0.0625	1111
-0.125	1110
-0.1875	1101
-0.25	1101
-0.3125	1101
-0.375	1100
-0.4375	1011
-0.5	1011
-0.5625	1011
-0.625	1010
-0.6875	1001
-0.75	1001
-0.8125	1001
-0.875	1000
-0.9375	1000
-1.	1000

## makePulseTrain

`? dsp`makePulseTrain`

makes a pulse train. Returns List of Lines that can be used in a plot

```
makePulseTrain[delay_?(NumberQ[#]&&Positive[#]&),(* delay of a pulse from time=0*)
period_?(NumberQ[#]&&Positive[#]&), (* the time between each pulse*)
range_?(NumberQ[#]&&Positive[#]&), (* the time duration to use*)
tao_?(NumberQ[#]&&Positive[#]&), (* width of each pulse*)
h_?(NumberQ[#]&&Positive[#]&) (* height of the pulse*)
]
```

Returns List of Lines.

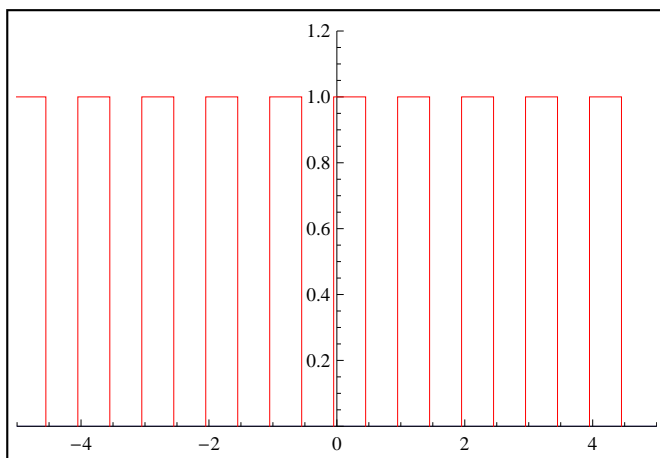
EXAMPLE USAGE:

```
delay=.2;period=1;range=5.;tao=.5;h=1;
```

```
currentPulses=dsp`makePulseTrain[delay,period,range,tao,h];
```

```
Plot[0,{x,-range,range},
PlotRange->{{-range,range},{0,h}},
Epilog->{Thin,Red,currentPulses}
]
```

```
delay = .2; period = 1; range = 5.; tao = .5; h = 1;
currentPulses = dsp`makePulseTrain[delay, period, range, tao, h];
Framed@Plot[0, {x, -range, range},
PlotRange -> {{-range, range}, {0, 1.2}}, Epilog -> {Thin, Red, currentPulses}]
```



## fcPulseTrain

```
? dsp`fcPulseTrain
```

```
fcPulseTrain[n,h,width,period]
```

returns the n'th fourier series complex coeff for a pulse train of height h, of width and period

EXAMPLE:

```
h=1;width=1;period=2;
```

```
Table[{n,dsp`fcPulseTrain[n,h,width,period]},{n,-4,4}]/TableForm
```

```
-4 0
-3  $-\frac{1}{3\pi}$ 
-2 0
-1  $\frac{1}{\pi}$ 
0  $\frac{1}{2}$ 
1  $\frac{1}{\pi}$ 
2 0
3  $-\frac{1}{3\pi}$ 
4 0
```

get nth fourier series complex coeff

```
h = 1; width = 1; period = 2;
```

```
Table[{n, dsp`fcPulseTrain[n, h, width, period]}, {n, -4, 4}] // TableForm
```

```
-4 0
-3  $-\frac{1}{3\pi}$ 
-2 0
-1  $\frac{1}{\pi}$ 
0  $\frac{1}{2}$ 
1  $\frac{1}{\pi}$ 
2 0
3  $-\frac{1}{3\pi}$ 
4 0
```

---

## Function to obtain Fourier series approximation given fundamental period and complex fourier coefficients 0..N-1

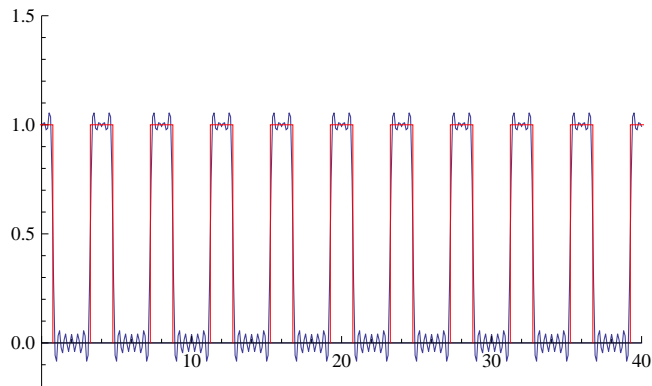
### ■ test on a square pulse

```
h = 1; width = 1.5; period = 4; n = 10; delay = 0; range = 10 * period;
Clear[t, x];
dsp`getFourierCoeffPulseTrain[h, width, period, n];

data = Table[{t, dsp`getFourierApproximation[t, %, period]}, {t, 0, range, .1}];

currentPulses = dsp`makePulseTrain[delay, period, range, width, h];

Show[{Plot[0, {x, 0, range},
  PlotRange -> {{0, range}, {-(.2 h), 1.5 h}}, Epilog -> {Thin, Red, currentPulses}],
  ListPlot[data, Joined -> True]}
```



---

## numIt

? dsp`numIt

Function takes a number and return it as string formatted

numIt[v\_?(NumberQ[##]&), (\*the number to format\*)  
 s1\_?(IntegerQ[##]&& Positive[##] &), (\* total number of digits to use, including the decimal digits\*)  
 s2\_?(IntegerQ[##]&& Positive[##] &), (\*number of digits to display to the right of the decimal point\*)  
 fontSize\_?(IntegerQ[##]&& Positive[##] &) (\*font size to use for string formatting\*)

EXAMPLE USAGE:

```
dsp`numIt[23.321,5,1,14]
```

23.3

```
dsp`numIt [23.321, 5, 1, 14]
```

```
dsp`numIt [23.321, 5, 2, 10]
```

23.3

23.32

---

## getButterworthPolynomial

Generate butterworth H(s) given order and cutoff

### ■ test1

```
Clear[s, form];
order = 2;
cutoff = .2 Pi;
{poles, hs} = dsp`getButterworthPolynomial [order, cutoff, s];
poles
TraditionalForm@hs
```

$$\frac{0.3947841760435742}{s^2 + 0.8885765876316731 s + 0.3947841760435742}$$

## ■ test2

```
{poles, hs} = dsp`getButterworthPolynomial [order, cutoff, s, form -> "linear"];
poles
TraditionalForm@hs
```

$$\{-0.44428829381583657 + 0.44428829381583657 i, -0.44428829381583657 - 0.44428829381583657 i\}$$

$$\frac{0.4442882938158365 i}{s + (0.44428829381583657 + 0.44428829381583657 i)} - \frac{0.4442882938158365 i}{s + (0.44428829381583657 - 0.44428829381583657 i)}$$

## ■ test3

```
{poles, hs} = dsp`getButterworthPolynomial [order, cutoff, s, form -> "quadratic"];
poles
TraditionalForm@hs
```

$$\{-0.44428829381583657 + 0.44428829381583657 i, -0.44428829381583657 - 0.44428829381583657 i\}$$

$$\frac{0.3947841760435742}{s^2 + 0.8885765876316731 s + 0.3947841760435742}$$

## ■ test4 find normalized Butterworth

```
{poles, hs} = dsp`getButterworthPolynomial [5, 1, s, form -> "quadratic"];
poles
TraditionalForm@hs
```

$$\left\{ \frac{1}{4} (1 - \sqrt{5}) + i \sqrt{\frac{5}{8} + \frac{\sqrt{5}}{8}}, \frac{1}{4} (-1 - \sqrt{5}) + i \sqrt{\frac{5}{8} - \frac{\sqrt{5}}{8}}, \right.$$

$$\left. -1, \frac{1}{4} (-1 - \sqrt{5}) - i \sqrt{\frac{5}{8} - \frac{\sqrt{5}}{8}}, \frac{1}{4} (1 - \sqrt{5}) - i \sqrt{\frac{5}{8} + \frac{\sqrt{5}}{8}} \right\}$$

$$\frac{-0.5527864045000425 s - 1.788854381999832}{2. s^2 + 1.2360679774997898 s + 2.} - \frac{3.23606797749979 s}{2. s^2 + 3.23606797749979 s + 2.} + \frac{1.8944271909999164}{s + 1}$$


---

**butterd**

```
? dsp`butterd
```

```
butterd[fs, variable, pass, stop, apass, astop, method, form];
```

design a low pass IIR digital filter from Butterworth analog filter.

fs : sampling frequency in Hz

variable: the variable to use. example z

pass : passband corner frequency in Hz

stop : stopband corner frequency in Hz

apass : attenuation at passband in db, negative value

astop : attenuation at stopband in db, negative value

method: design method to use, can be method->"bilinear",

method->"impulse", method->"matchedz". If not given, then the default is "bilinear"

form: the form to return H(z) in. can be form->"linear" returns H(z) in the form linear

terms in the variable, form->"quadratic" returns H(z) in quadratic terms in the

variable. If this option is not selected, then H(z) will be returned as one polynomial.

Returns {order, cutoff, digitalFilter, digitalFilterPoles, digitalFilterzeros, a, b},

order: Butterworth filter order found from the specifications

cutoff: Butterworth cutoff frequency found from the specifications

digitalFilter:  $H(z) = B(z)/A(z)$ , transfer function in the form selected above.

digitalFilterPoles: locations of H(z) poles

digitalFilterzeros: locations of H(z) zeros

a : the order+1 filter coefficients for A(z). Coefficients are listed in descending powers of z

b : the order+1 filter coefficients for B(z). Coefficients are listed in descending powers of z

EXAMPLE usage:

Example1

```
-----

test1
Clear[z];
fs=20000 (*hz*);
pass=2000 (*hz*);
stop=3000 (*hz*);
apass=-1 (*db*);
astop=-15 (*db*);
```



```
zeros
{-1,-1,-1,-1,-1,-1}
```

butterworth digital filter design

#### ■ test1

```
Clear[z];
fs = 20 000 (*hz*);
pass = 2000 (* hz*);
stop = 3000 (*hz*);
apass = -1 (*db*);
astop = -15 (*db*);
{filterOrder, Ωc, hz, poles, zeros, a, b} =
  dsp`butterd[fs, z, pass, stop, apass, astop, form → "quadratic"];
TraditionalForm@
```

$$\frac{-0.8786598279577791 z^2 - 0.37458656177813776 z + 0.5040732661796413}{1. z^2 - 1.0105788810461338 z + 0.35827133770639796} + \frac{0.05327834218253358 z^2 - 0.1260013381027728 z - 0.1792796802853064}{1. z^2 - 1.2686468043871897 z + 0.7051282432185233} + \frac{0.8261193057058389 z^2 + 0.5806131391232774 z - 0.24550616658256155}{1. z^2 - 0.9043660641139339 z + 0.2155157075998965}$$

**poles**

```
{0.6343234021935948 + 0.550238189012782 i, 0.5052894405230669 + 0.320864331146801 i,
0.4521830320569668 + 0.10510096631175422 i, 0.4521830320569668 - 0.10510096631175422 i,
0.5052894405230669 - 0.320864331146801 i, 0.6343234021935948 - 0.550238189012782 i}
```

**zeros**

```
{-1, -1, -1, -1, -1, -1}
```

#### ■ test2

```
{filterOrder, Ωc, hz, poles, zeros, a, b} =
  dsp`butterd[fs, z, pass, stop, apass, astop, method → "impulse"];
TraditionalForm@
```

$$\frac{(12.619276514087685 z^5 + 202.07004065216458 z^4 + 322.8682701355137 z^3 + 82.01389599025786 z^2 + 2.0650372218951816 z)}{(z^6 - 3.3635196107885204 z^5 + 5.068420161781825 z^4 - 4.275864216158864 z^3 + 2.1066205743828084 z^2 - 0.5706492537421755 z + 0.06607428351012314)}$$

**poles**

```
{0.6485799325392105 + 0.5236709777967435 i, 0.5345537369865101 + 0.2901159614276238 i,
0.49862613586853954 + 0.09176688744135254 i, 0.49862613586853954 - 0.09176688744135254 i,
0.5345537369865101 - 0.2901159614276238 i, 0.6485799325392105 - 0.5236709777967435 i}
```

**zeros**

$\{\infty, \infty, \infty, \infty, \infty, \infty\}$

```
{filterOrder, Ωc, hz, poles, zeros, a, b} =
  dsp`butterd[fs, z, pass, stop, apass, astop, form → "linear"];
```

**TraditionalForm@**

**hz**

$$\frac{3128.118552949461 - 5418.060265807306 i}{1 - \frac{0.6055598764724284 - 0.5530645361185251 i}{z}} - \frac{23348.59474341246}{1 - \frac{0.4983854027120604 + 0.2999718179941991 i}{z}} -$$

$$\frac{23348.59474341246}{1 - \frac{0.4983854027120604 - 0.2999718179941991 i}{z}} + \frac{20220.476190462996 - 35022.89211511871 i}{1 - \frac{0.46770597726956914 + 0.09398839450941299 i}{z}} +$$

$$\frac{20220.476190462996 + 35022.89211511871 i}{1 - \frac{0.46770597726956914 - 0.09398839450941299 i}{z}} + \frac{3128.118552949461 + 5418.060265807306 i}{1 - \frac{0.6055598764724284 + 0.5530645361185251 i}{z}}$$

## getPositiveAndNegativeTerms

```
? dsp`getPositiveAndNegativeTerms
```

```
{neg,pos}=getPositiveAndNegativeTerms[expr]
```

Takes an expression expr and returns the terms in it which have negative sign and the terms which have positive sign

EXAMPLE

```
In[24]:= expr=3-c+b+d-h
```

```
{neg,pos}=getPositiveAndNegativeTerms[expr]
```

```
Out[25]= {{-c,-h},{3,b,d}}
```

Thanks to a post by Albert Retey on Math news group for the above idea of a solution to a question posted, which I saw on sept 9, 2010.

```
Clear[c, b, d, h]
```

```
expr = 3 - c + b + d - h
```

```
{neg, pos} = dsp`getPositiveAndNegativeTerms [expr]
```

```
3 + b - c + d - h
```

```
{{-c, -h}, {3, b, d}}
```

## butterToLowPass

? dsp`butterToLowPass

```
butterToLowPass[Hs,Ωc,variable];
```

convert Butterworth normalized low pass analog filter Hs to low pass analog filter with cutoff frequency  $\Omega_c$ . Variable is the symbolic variable of the polynomial.

### ■ test1

make a normalized butter low pass analog filter of order 4

```
Clear[s, ω];
{poles, hs} = dsp`getButterworthPolynomial[4, 1, s];
TraditionalForm@hs
```

$$\frac{1.}{s^4 + 2.613125929752753 s^3 + 3.414213562373095 s^2 + 2.613125929752753 s + 1.}$$

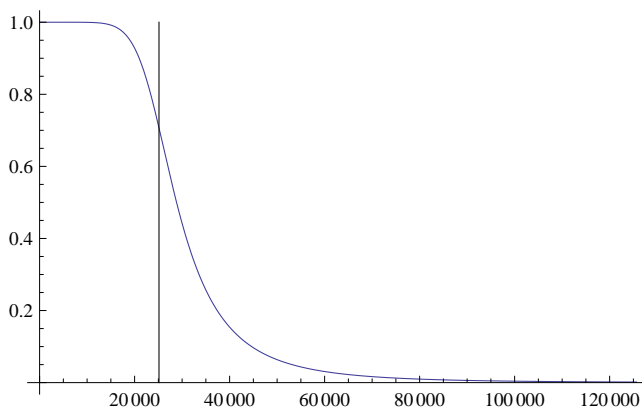
convert the cut off frequency to a new cutoff frequency

```
Ωc =  $\frac{2 \text{ Pi}}{0.25 * 10^{(-3)}}$ ;
hsLow = dsp`butterToLowPass[hs, Ωc, s];
TraditionalForm@hsLow
```

$$1./\left(2.5063433238975437 \times 10^{-18} s^4 + 1.646041423861472 \times 10^{-13} s^3 + 5.4051899898025724 \times 10^{-9} s^2 + 0.0001039729771604389 s + 1.\right)$$

Plot the magnitude spectrums

```
ftNew = hsLow /. s -> (I ω);
Plot[Abs[ftNew], {ω, 0, 5 Ωc}, PlotRange -> All,
  AxesOrigin -> {0, 0}, Epilog -> Line[{{Ωc, 0}, {Ωc, 1}}]]
```



## butterToHighPass

? dsp`butterToHighPass

butterToHighPass[Hs,Ω,variable];  
 convert Butterworth normalized low pass analog filter Hs to high pass  
 analog filter with corner frequency Ω. Variable is the symbolic variable of the polynomial.

### ■ test1

make a normalized butter low pass analog filter of order 4

```
Clear[s, ω];
{poles, hs} = dsp`getButterworthPolynomial[4, 1, s];
TraditionalForm@hs
```

$$\frac{1.}{s^4 + 2.613125929752753 s^3 + 3.414213562373095 s^2 + 2.613125929752753 s + 1.}$$

convert to high pass

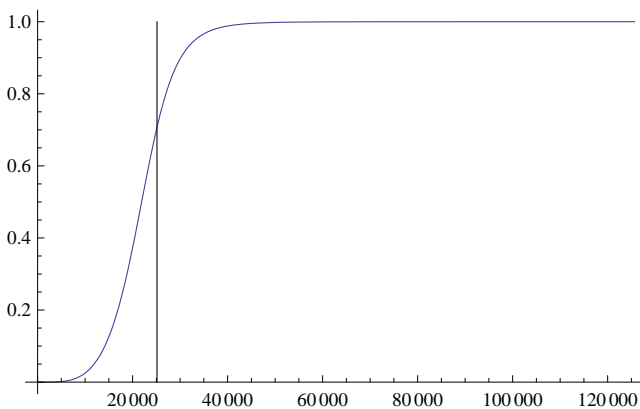
```
Ω =  $\frac{2 \text{ Pi}}{0.25 * 10^{(-3)}}$ ;
hsHigh = dsp`butterToHighPass[hs, Ω, s];
TraditionalForm@hsHigh
```

25 132.741228718343

$$\frac{1.}{1. + \frac{3.9898763687527386 \times 10^{17}}{s^4} + \frac{4.1483932456130336 \times 10^{13}}{s^3} + \frac{2.156603980893214 \times 10^9}{s^2} + \frac{65675.01779052998}{s}}$$

Plot the magnitude spectrums

```
ftNew = hsHigh /. s -> (I ω);
Plot[Abs[ftNew], {ω, 0, 5 Ω}, PlotRange -> All,
  AxesOrigin -> {0, 0}, Epilog -> Line[{{Ωc, 0}, {Ωc, 1}}]]
```



## butterToBandPass

? dsp`butterToBandPass

butterToBandPass[Hs,Ω<sub>1</sub>,Ω<sub>2</sub>,variable];  
 convert Butterworth normalized low pass analog filter Hs to band pass analog filter with low corner frequency Ω<sub>1</sub> and high corner frequency Ω<sub>2</sub>. Variable is the symbolic variable of the polynomial.

### ■ test1

make a normalized butter low pass analog filter of order 4

```
Clear [s, ω];
{poles, hs} = dsp`getButterworthPolynomial [4, 1, s];
TraditionalForm@hs
```

$$\frac{1.}{s^4 + 2.613125929752753 s^3 + 3.414213562373095 s^2 + 2.613125929752753 s + 1.}$$

convert to band pass

$$\Omega_1 = \frac{2 \text{ Pi}}{0.25 * 10^{(-3)}};$$

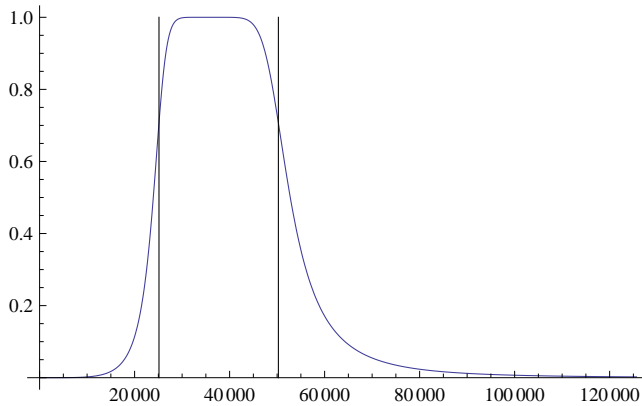
$$\Omega_2 = \frac{4 \text{ Pi}}{0.25 * 10^{(-3)}};$$

```
hsBandPass = dsp`butterToBandPass [hs, Ω1, Ω2, s];
TraditionalForm@hsBandPass
```

$$1. / \left( \frac{5.4051899898025724 \times 10^{-9} (s^2 + 1.2633093633394377 \times 10^9)^2}{s^2} + \frac{0.0001039729771604389 (s^2 + 1.2633093633394377 \times 10^9)}{s} + \frac{2.5063433238975437 \times 10^{-18} (s^2 + 1.2633093633394377 \times 10^9)^4}{s^4} + \frac{1.646041423861472 \times 10^{-13} (s^2 + 1.2633093633394377 \times 10^9)^3}{s^3} + 1. \right)$$

Plot the magnitude spectrums

```
ftNew = hsBandPass /. s -> (I ω);
Plot[Abs[ftNew], {ω, 0, 5 Ωc}, PlotRange -> All, AxesOrigin -> {0, 0},
  Epilog -> {Line[{{Ω1, 0}, {Ω1, 1}}, Line[{{Ω2, 0}, {Ω2, 1}}]}]
```




---

## butterToBandStop

```
? dsp`butterToBandStop
```

```
butterToBandStop[Hs,Ω1,Ω2,variable];
```

convert Butterworth normalized low pass analog filter Hs to band stop analog filter with low corner frequency  $\Omega_1$  and high corner frequency  $\Omega_2$ . Variable is the symbolic variable of the polynomial.

### ■ test1

make a normalized butter low pass analog filter of order 4

```
Clear[s, ω];
{poles, hs} = dsp`getButterworthPolynomial[4, 1, s];
TraditionalForm@hs
```

$$1.$$

$$s^4 + 2.613125929752753 s^3 + 3.414213562373095 s^2 + 2.613125929752753 s + 1.$$

convert to band stop

```

Ω1 =  $\frac{2 \text{ Pi}}{0.25 * 10^{(-3)}};$ 
Ω2 =  $\frac{4 \text{ Pi}}{0.25 * 10^{(-3)}};$ 
hsBandStop = dsp`butterToBandStop[hs, Ω1, Ω2, s];
TraditionalForm@hsBandStop

```

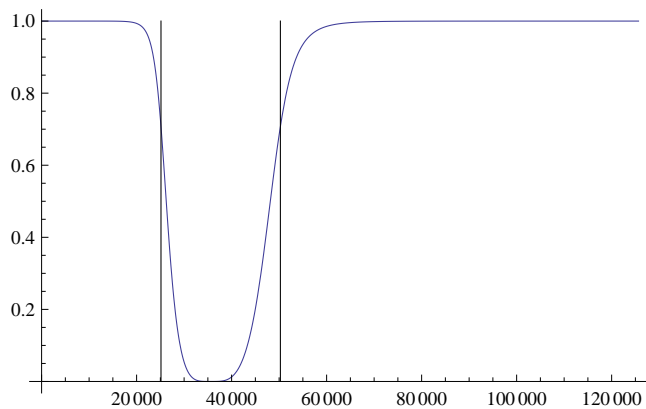
$$1. / \left( \frac{2.156603980893214 \times 10^9 s^2}{(s^2 + 1.2633093633394377 \times 10^9)^2} + \frac{65675.01779052998 s}{s^2 + 1.2633093633394377 \times 10^9} + \frac{3.9898763687527386 \times 10^{17} s^4}{(s^2 + 1.2633093633394377 \times 10^9)^4} + \frac{4.1483932456130336 \times 10^{13} s^3}{(s^2 + 1.2633093633394377 \times 10^9)^3} + 1. \right)$$

Plot the magnitude spectrums

```

ftNew = hsBandStop /. s -> (I ω);
Plot[Abs[ftNew], {ω, 0, 5 Ωc}, PlotRange -> All, AxesOrigin -> {0, 0},
  Epilog -> {Line[{{Ω1, 0}, {Ω1, 1}}, Line[{{Ω2, 0}, {Ω2, 1}}]}]

```



---

## plotFourierTransform

`? dsp`plotFourierTransform`

`plotFourierTransform[ft,w,from,to,ymin,ymax,arrowheadsize]`  
Plot fourier transform, including pulses

### EXAMPLE

```
x[t_]:=2 Cos[2 Pi 10 t]
Plot[x[t],{t,0, 2/10}]
FourierTransform[x[t],t,w,FourierParameters→{-1,-1}];

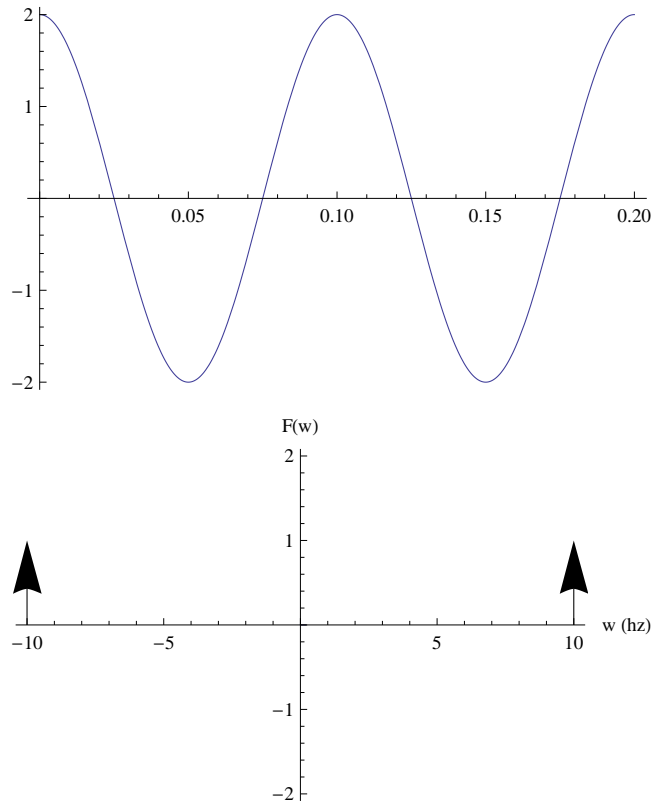
dsp`plotFourierTransform[%t,0,2 period,-2,2,Small]
```

```

fm = 10;
period = 1 / fm;
x[t_] := 2 Cos[2 Pi fm t]
Plot[x[t], {t, 0, 2 / 10}]
FourierTransform[x[t], t, w, FourierParameters -> {0, 2 Pi}];

Show[First@dsp`plotFourierTransform[%, w, 0, 2 period, -2, 2, Large],
  AxesLabel -> {"w (hz)", "F(w)"}]

```




---

## Funtion to divide polynomial by powers of variable and keep powers in form $z^{-n}$

```

dividePolyFormattedV1[poly_, var_, order_?(IntegerQ[#] && Positive[#] &)] :=
  Module[{}, Apart[poly / var^order]];

```

(\*thanks to Carlos Casar de Araujo on mathgroup for the following 2 functions\*)

```

MakeBoxes[Power[x_, y_?(NumberQ[#] && Negative[#] &)], TraditionalForm] :=
  SuperscriptBox[MakeBoxes[x, TraditionalForm], MakeBoxes[y, TraditionalForm]];
MakeBoxes[Times[c_, p : Power[x_, y_?(NumberQ[#] && Negative[#] &)]], TraditionalForm] := If[
  OrderedQ[{c, p}], RowBox[{MakeBoxes[c, TraditionalForm], MakeBoxes[p, TraditionalForm]}],
  RowBox[{MakeBoxes[p, TraditionalForm], MakeBoxes[c, TraditionalForm]}]];

```

■ test

```
Framed@Style[TraditionalForm[dividePolyFormattedV1[1 + x, x, 3]], 14]
TraditionalForm[dividePolyFormattedV1[1 + x + x^3, x, 3]]
TraditionalForm[dividePolyFormattedV1[1 + x + x^2 + x^7, x, 3]]
```

```
Grid[{
  {Framed@Style[TraditionalForm[1 / z], 14]}, Alignment -> Center, Spacings -> 1
}]
```

$$\boxed{x^{-2} + x^{-3}}$$

$$x^{-2} + x^{-3} + 1$$

$$x^{-1} + x^{-2} + x^{-3} + x^4$$

$$\boxed{z^{-1}}$$

## Function to convert low pass digital filter to high pass using frequency transformation

■ test

```
Clear[z, hz, alpha, cutOffFrequency, wp]
```

```
hz[z_] := Module[{}, 
$$\frac{0.001836 (1 + z^{-1})^4}{(1 - 1.5548 z^{-1} + 0.6493 z^{-2}) (1 - 1.4996 z^{-1} + 0.8482 z^{-2})}$$
];
```

```
expectedHz[z_] := Module[{}, 
$$\frac{0.02426 (1 - z^{-1})^4}{(1 - 1.0416 z^{-1} + 0.4019 z^{-2}) (1 - 0.5561 z^{-1} + 0.7647 z^{-2})}$$
];
```

```
cutOffFrequency = .2 Pi;
```

```
wp = .6 Pi;
```

```
alpha = - 
$$\frac{\cos\left[\frac{wp + cutOffFrequency}{2}\right]}{\cos\left[\frac{wp - cutOffFrequency}{2}\right]}$$
;
```

```
foundHz[z_] := Module[{}, hz[
$$\left(-\frac{z^{-1} + \alpha}{1 + \alpha z^{-1}}\right)$$
];
```

```
magPhasePlot[foundHz, 1]
```

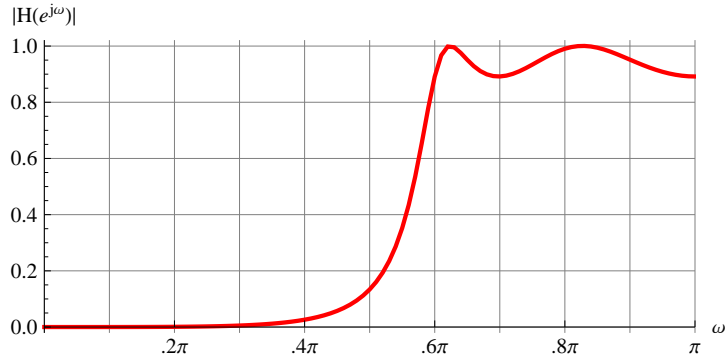
```
-0.38196601125010515
```

$$\left( 0.001836 \left( 1 - \frac{1 - 0.38196601125010515 e^{-i \omega 47822}}{-0.38196601125010515 + e^{-i \omega 47822}} \right)^4 \right) /$$

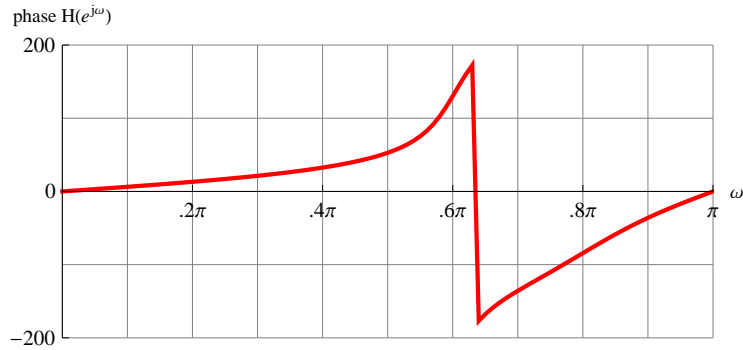
$$\left( \left( 1 + \frac{0.8482 (1 - 0.38196601125010515 e^{-i \omega 47822})^2}{(-0.38196601125010515 + e^{-i \omega 47822})^2} + \frac{1.4996 (1 - 0.38196601125010515 e^{-i \omega 47822})}{-0.38196601125010515 + e^{-i \omega 47822}} \right) \right)$$

$$\left( 1 + \frac{0.6493 (1 - 0.38196601125010515 e^{-i \omega 47822})^2}{(-0.38196601125010515 + e^{-i \omega 47822})^2} + \frac{1.5548 (1 - 0.38196601125010515 e^{-i \omega 47822})}{-0.38196601125010515 + e^{-i \omega 47822}} \right) \right)$$

magnitude spectrum of digital filter



phase spectrum of digital filter



## ■ test2

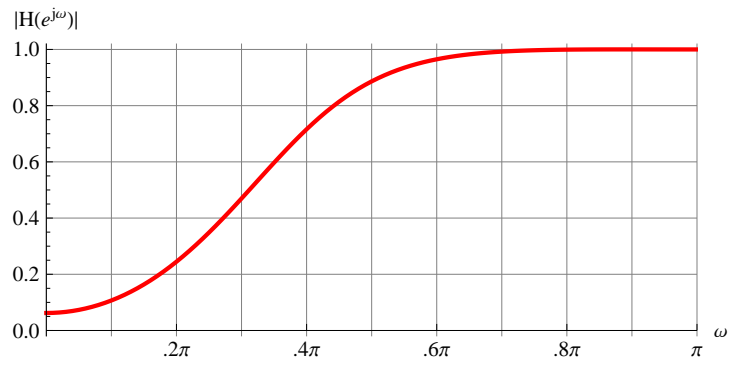
```

Clear[z, hz, x, expectedHz, foundHz]
hz[z_] := Module[{},  $\frac{0.175 z^{-1}}{1 - 1.318 z^{-1} + 0.493 z^{-2}}$ ];
expectedHz[z_] := Module[{},  $\frac{0.12 z^{-2} - 0.258 z^{-1} + 0.12}{0.059 z^{-2} - 0.111 z^{-1} + 0.327}$ ];
cutOffFrequency = .16 Pi;
wp = .4 Pi;
alpha = -  $\frac{\cos\left[\frac{wp+cutOffFrequency}{2}\right]}{\cos\left[\frac{wp-cutOffFrequency}{2}\right]}$ ;
foundHz[z_] := Module[{}, hz[ $\left(-\frac{z^{-1} + \alpha}{1 + \alpha z^{-1}}\right)$ ]];
magPhasePlot[foundHz, 1]

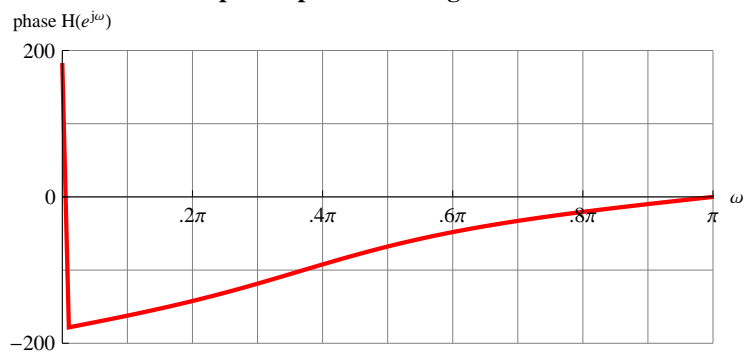
```

$$\begin{aligned}
& - \left( 0.175 \left( 1 - 0.6855669071257852 e^{-i \omega 49394} \right) \right) / \left( -0.6855669071257852 + e^{-i \omega 49394} \right) \\
& \left( 1 + \frac{0.493 \left( 1 - 0.6855669071257852 e^{-i \omega 49394} \right)^2}{\left( -0.6855669071257852 + e^{-i \omega 49394} \right)^2} + \frac{1.318 \left( 1 - 0.6855669071257852 e^{-i \omega 49394} \right)}{-0.6855669071257852 + e^{-i \omega 49394}} \right)
\end{aligned}$$

magnitude spectrum of digital filter



phase spectrum of digital filter



---

## Function to convert low pass digital filter to bandpass digital filter using frequency transformation

### ■ test

```

Clear[z, hz, alpha, cutOff, w1, w2, k]

hz[z_] := Module[{}, 
$$\frac{0.001836 (1 + z^{-1})^4}{(1 - 1.5548 z^{-1} + 0.6493 z^{-2}) (1 - 1.4996 z^{-1} + 0.8482 z^{-2})}$$
];

cutOff = .2 Pi;
w1 = .4 Pi;
w2 = .6 Pi;

alpha = 
$$\frac{\cos\left[\frac{w2+w1}{2}\right]}{\cos\left[\frac{w2-w1}{2}\right]}$$
;

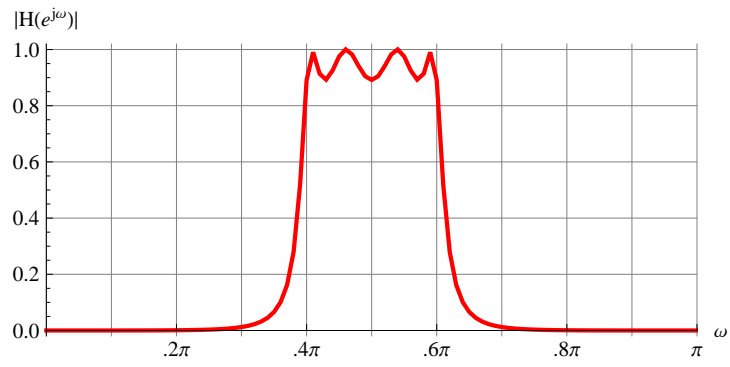
k = 
$$\cot\left[\frac{w2 - w1}{2}\right] \tan[\text{cutOff} / 2]$$
;

foundHz[z_] := Module[{}, hz[-N[
$$\left(\frac{z^{-2} - \frac{2 \text{alpha} k}{k+1} z^{-1} + \frac{k-1}{k+1}}{\frac{k-1}{k+1} z^{-2} - \frac{2 \text{alpha} k}{k+1} z^{-1} + 1}\right)$$
]]];

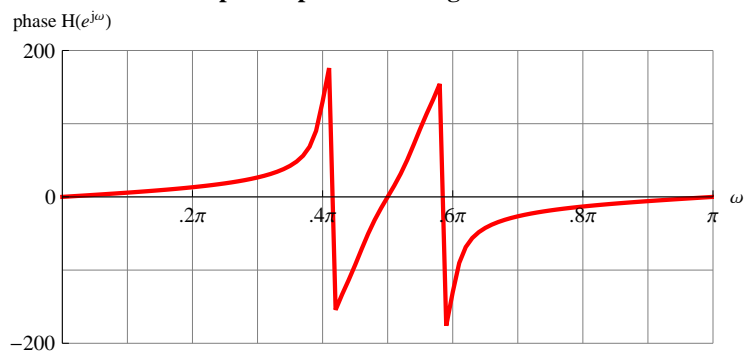
magPhasePlot [foundHz, 1]

```

magnitude spectrum of digital filter



phase spectrum of digital filter



---

## Function to convert low pass digital filter to bandstop digital filter by frequency transformation

### ■ test

```

Clear[z, hz, alpha, cutOff, w1, w2, k]

hz[z_] := Module[{}, 
$$\frac{0.001836 (1 + z^{-1})^4}{(1 - 1.5548 z^{-1} + 0.6493 z^{-2}) (1 - 1.4996 z^{-1} + 0.8482 z^{-2})}$$
];

cutOff = .2 Pi;
w1 = .4 Pi;
w2 = .6 Pi;

alpha = 
$$\frac{\cos\left[\frac{w2+w1}{2}\right]}{\cos\left[\frac{w2-w1}{2}\right]}$$
;

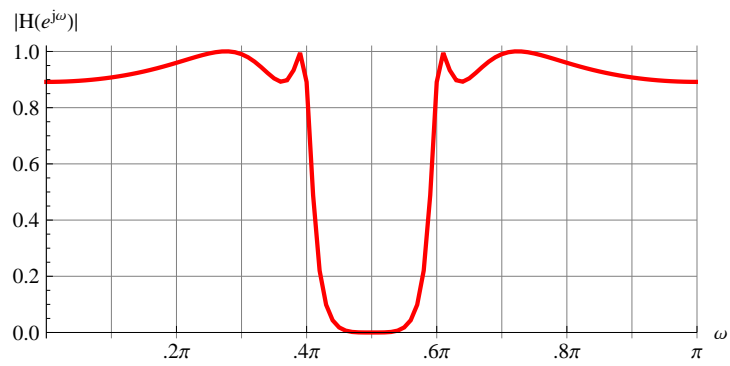
k = 
$$\tan\left[\frac{w2 - w1}{2}\right] \tan[\text{cutOff} / 2]$$
;

foundHz[z_] := Module[{}, hz[N[
$$\left(\frac{z^2 - 2 - \frac{2\alpha}{k+1} z^{-1} + \frac{1-k}{k+1}}{\frac{1-k}{k+1} z^2 - 2 - \frac{2\alpha}{k+1} z^{-1} + 1}\right)$$
]]];

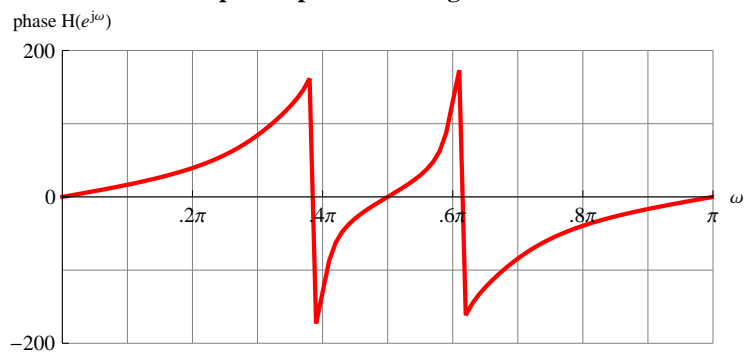
magPhasePlot [foundHz, 1]

```

magnitude spectrum of digital filter



phase spectrum of digital filter



```
a = 2.
```

```
2.
```

```
Apply[a, Integer]
```

```
Integer
```

```
Head[a]
```

```
Real
```

```
a /. Real -> Integer
```

```
2.
```

```
a
```

```
2.
```

```
Round[a]
```

```
2
```