Test notebook for my Mathematica package

In[1]:=
```
Options[$FrontEnd, NotebookBrowseDirectory]
```

Out[1]=
```
{NotebookBrowseDirectory → G:\nabbasi\data\tmp}
```

In[2]:=
```
NotebookDirectory[]
```

Out[2]=
```
X:\data\public_html\my_notes\my_dsp_mathematica_package\
```


Last updated Jan 10, 2017

This notebook tests each function on nma.m, which is package I maintain that contains some useful functions.

In[1]:=
```
SetDirectory[NotebookDirectory[]]
Get["nma.m"]
```

Out[1]=
```
X:\data\public_html\my_notes\my_dsp_mathematica_package
```

In[3]:=
```
?nma`*
```

▼ nma`

butterd	generateLTVObservabilityMatrix	matrix2DredBlackPositions
butterToBandPass	getButterworthPolynomial	numIt
butterToBandStop	getControllableForm	numItse
butterToHighPass	getMagnitudeOfPulseTrainFourierCoeff	plotFourierTransform
butterToLowPass	getMaxPadding	poissonMatrixOnSquareDirichlet
centerImageData	getPhaseOfPulseTrainFourierCoeff	polarForm
checkForSingularity	getPositiveAndNegativeTerms	quantizationLevel
convertToLCD	getStateGainVector	snip
diagonalBlockMatrix	hurwitz	str
diagonalDominantMatrixQ	kharitonovPoly	stripGraphMouseAppearance
displayHurwitz	leadingDet	toOffsetBinary
fcPulseTrain	linspace	toOnesComplement
formatBinary	makePulseTrain	toSignMagnitude
generateLTVControllabilityMatrix	matrix2DLexOrdering	toTwoComplement

checkForSingularity[p, q, at, var]

Classifies singularity in ODE in the form y''+p(x) y' + q(x) y = 0

at is list of the points to check singularity at. The var is the
indepenent variable symbol, such as x. It returns a matrix, where
each row is of form {point, p limit, q limit}

If any one of the p limit or q limit is infinity, it means the singularity
is not removable. If the limits are finite, it means this is removable
singularity.

Examples:

```
ClearAll[c, a, b, x];
p=(c – (a + b + 1) x)/(x (1 – x));
q=(–a b)/(x (1 – x));
at={0,1,Infinity};

m = checkForSingularity[p,q ,at, x];
Grid[Join[{{"point", "limit x p(x)", "limit x^2 q(x)"}}, m],
 Frame –> All]
 {{–1, 1, –(mu^2/4)}, {1, 1, –(mu^2/4)}, {∞, 0, ∞}}
```

Another example

```
ClearAll[c, a, b, x];
p=–2 x/(1 – x^2);
q=(lam + 40 (1 – x^2) – (mu^2)/(1 – x^2))/(1 – x^2);
at={0,1,Infinity};

m = checkForSingularity[p,q ,at, x];
```

{{–1, 1, –(mu^2/4)}, {1, 1, –(mu^2/4)}, {∞, 0, ∞}}

butterd

In[63]:= `? nma`butterd`

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

design a low pass IIR digital filter from Butterwoth 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, cuttoff,digitalFilter, digitalFilterPoles, digitalFilterzeros, a, b},

order: Butterworth filter order found from the specifications
cuttoff: Butterworth cuttoff 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*);
{filterOrder,Ωc,hz,poles,zeros,a,b}=nma`butterd[fs,z,pass,stop,apass,astop,form→"quadratic"];
TraditionalForm@hz
```

$$\left(-0.87866\, z^2-0.374587\, z+0.504073\right)\Big/\left(1.\, z^2-1.01058\, z+0.358271\right)+\left(0.0532783\, z^2-0.126001\, z-0.17928\right)\Big/\left(1.\, z^2-1.26865\, z+0.705128\right)+\left(0.826119\, z^2+0.580613\, z-0.245506\right)\Big/\left(1.\, z^2-0.904366\, z+0.215516\right)$$

```
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}
```

Example 2
––––––––––

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

$$\left(12.6193\, z^5+202.07\, z^4+322.868\, z^3+82.0139\, z^2+2.06504\, z\right)\Big/\left(z^6-3.36352\, z^5+5.06842\, z^4-4.27586\, z^3+2.10662\, z^2-0.570649\, z+0.0660743\right)$$

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

{∞,∞,∞,∞,∞,∞}

Example 3

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

TraditionalForm@hz

$$\frac{3128.12-5418.06\,i}{1-\frac{0.60556-0.553065\,i}{z}}-\frac{23348.6}{1-\frac{0.498385+0.299972\,i}{z}}-\frac{23348.6}{1-\frac{0.498385-0.299972\,i}{z}}+\frac{20220.5-35022.9\,i}{1-\frac{0.467706+0.0939884\,i}{z}}+\frac{20220.5+35022.9\,i}{1-\frac{0.467706-0.0939884\,i}{z}}+\frac{3128.12+5418.06\,i}{1-\frac{0.60556+0.553065\,i}{z}}$$

zeros

{−1,−1,−1,−1,−1,−1}

butterworth digital filter design

test l

In[64]:=
```
Clear[z];
fs = 20000 (*hz*);
pass = 2000 (* hz*);
stop = 3000 (*hz*);
apass = -1 (*db*);
astop = -15 (*db*);
{filterOrder, Ωc, hz, poles, zeros, a, b} =
  nma`butterd[fs, z, pass, stop, apass, astop, form → "quadratic"];
TraditionalForm@
 hz
```

Out[71]//TraditionalForm=

$$-\frac{0.0584099\,z}{z\,(1.\,z-1.26865)+0.705128}-\frac{1.26254\,z}{z\,(1.\,z-1.01058)+0.358271}+\frac{1.32773\,z}{z\,(1.\,z-0.904366)+0.215516}-\frac{0.216848}{z\,(1.\,z-1.26865)+0.705128}+\frac{0.818872}{z\,(1.\,z-1.01058)+0.358271}-\frac{0.423548}{z\,(1.\,z-0.904366)+0.215516}+0.00073782$$

In[72]:=
```
poles
```

Out[72]=
```
{0.634323 + 0.550238 i, 0.505289 + 0.320864 i, 0.452183 + 0.105101 i,
 0.452183 - 0.105101 i, 0.505289 - 0.320864 i, 0.634323 - 0.550238 i}
```

In[73]:=
```
zeros
```

Out[73]=
```
{-1, -1, -1, -1, -1, -1}
```

test2

In[74]:=
```
{filterOrder, Ωc, hz, poles, zeros, a, b} =
  nma`butterd[fs, z, pass, stop, apass, astop, method → "impulse", form → "quadratic"];
TraditionalForm@
 hz
```

Out[75]//TraditionalForm=

$$\frac{5741.65\,z^2}{(z-1.29716)\,z+0.694887}-\frac{42\,856.2\,z^2}{(z-1.06911)\,z+0.369915}+\frac{37\,114.6\,z^2}{(z-0.997252)\,z+0.257049}-\frac{8931.73\,z}{(z-1.29716)\,z+0.694887}+\frac{22\,909.\,z}{(z-1.06911)\,z+0.369915}-\frac{12\,607.1\,z}{(z-0.997252)\,z+0.257049}$$

In[76]:=
```
poles
```

Out[76]=
```
{0.64858 + 0.523671 𝕚, 0.534554 + 0.290116 𝕚, 0.498626 + 0.0917669 𝕚,
 0.498626 - 0.0917669 𝕚, 0.534554 - 0.290116 𝕚, 0.64858 - 0.523671 𝕚}
```

In[77]:=
```
zeros
```

Out[77]=
```
{∞, ∞, ∞, ∞, ∞, ∞}
```

In[78]:=
```
{filterOrder, Ωc, hz, poles, zeros, a, b} =
  nma`butterd[fs, z, pass, stop, apass, astop, form → "linear"];
TraditionalForm@
 hz
```

Out[79]//TraditionalForm=

$$\frac{3128.12-5418.06\,i}{1-\frac{0.60556-0.553065\,i}{z}}-\frac{23\,348.6}{1-\frac{0.498385+0.299972\,i}{z}}-\frac{23\,348.6}{1-\frac{0.498385-0.299972\,i}{z}}+\frac{20\,220.5-35\,022.9\,i}{1-\frac{0.467706+0.0939884\,i}{z}}+\frac{20\,220.5+35\,022.9\,i}{1-\frac{0.467706-0.0939884\,i}{z}}+\frac{3128.12+5418.06\,i}{1-\frac{0.60556+0.553065\,i}{z}}$$

---

butterToBandPass

In[80]:=
```
? nma`butterToBandPass
```

butterToBandPass[Hs,Ω_1,Ω_2,variable];
convert Butterworth normalized low pass analog filter Hs to band pass analog filter with low
corner frequency Ω_1 and high corner frequency Ω_2. Variable is the symbolic variable of the polynomial.

test1

make a normalized butter low pass analog filter of order 4

In[81]:=
```
Clear[s, omega];
{poles, hs} = nma`getButterworthPolynomial[4, 1, s];
TraditionalForm@hs
```

Out[83]//TraditionalForm=

$$\frac{1.}{1.\,s^4 + 2.61313\,s^3 + 3.41421\,s^2 + 2.61313\,s + 1.}$$

convert to band pass

In[84]:=
```
gamma1 = 2 Pi / (0.25 * 10^(-3));
gamma2 = 4 Pi / (0.25 * 10^(-3));
hsBandPass = nma`butterToBandPass[hs, gamma1, gamma2, s];
TraditionalForm@hsBandPass
```

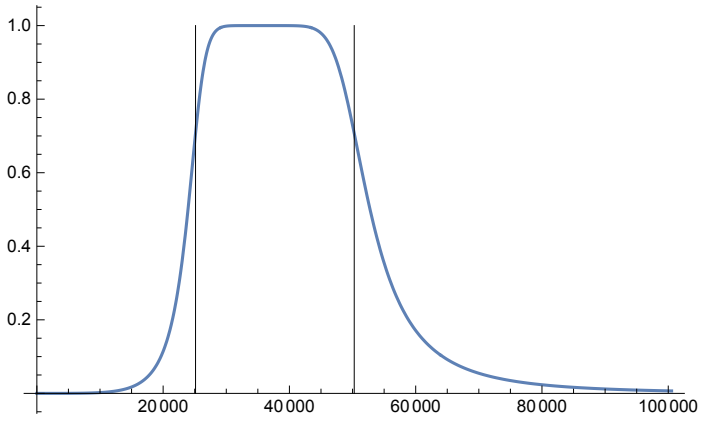
Out[87]//TraditionalForm=

$$1.\Bigg/\left(\frac{5.40519\times10^{-9}\left(s^2+1.26331\times10^9\right)^2}{s^2}+\frac{0.000103973\left(s^2+1.26331\times10^9\right)}{s}+\frac{2.50634\times10^{-18}\left(s^2+1.26331\times10^9\right)^4}{s^4}+\frac{1.64604\times10^{-13}\left(s^2+1.26331\times10^9\right)^3}{s^3}+1.\right)$$

Plot the magnitude spectrums

In[88]:=
```
ftNew = hsBandPass /. s → (I omega);
Plot[Abs[ftNew], {omega, 0, 2 gamma2}, PlotRange → All, AxesOrigin → {0, 0},
 Epilog → {Line[{{gamma1, 0}, {gamma1, 1}}], Line[{{gamma2, 0}, {gamma2, 1}}]}]
```

Out[89]=



butterToBandStop

In[90]:=
```
? nma`butterToBandStop
```

butterToBandStop[Hs,Ω_1,Ω_2,variable];
convert Butterworth normalized low pass analog filter Hs to band stop analog filter with low
corner frequency Ω_1 and high corner frequency Ω_2. Variable is the symbolic variable of the polynomial.

test1

make a normalized butter low pass analog filter of order 4

In[91]:=
```
Clear[s, omega];
{poles, hs} = nma`getButterworthPolynomial[4, 1, s];
TraditionalForm@hs
```

Out[93]//TraditionalForm=

$$\frac{1.}{1.\, s^4 + 2.61313\, s^3 + 3.41421\, s^2 + 2.61313\, s + 1.}$$

convert to band stop

In[94]:=
```
gamma1 = 2 Pi / (0.25 * 10^(-3));
gamma2 = 4 Pi / (0.25 * 10^(-3));
hsBandStop = nma`butterToBandStop[hs, gamma1, gamma2, s];
TraditionalForm@hsBandStop
```

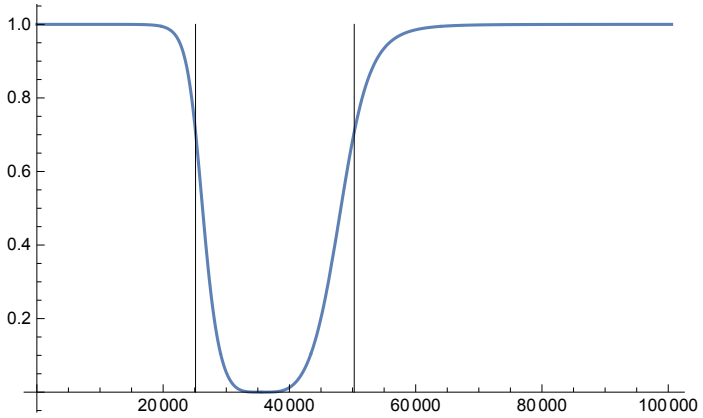
Out[97]//TraditionalForm=

$$1. \Bigg/ \left(\frac{2.1566\times 10^9\, s^2}{\left(s^2 + 1.26331\times 10^9\right)^2} + \frac{65\,675.\, s}{s^2 + 1.26331\times 10^9} + \frac{3.98988\times 10^{17}\, s^4}{\left(s^2 + 1.26331\times 10^9\right)^4} + \frac{4.14839\times 10^{13}\, s^3}{\left(s^2 + 1.26331\times 10^9\right)^3} + 1. \right)$$

Plot the magnitude spectrums

In[98]:=
```
ftNew = hsBandStop /. s → (I omega);
Plot[Abs[ftNew], {omega, 0, 2 gamma2}, PlotRange → All, AxesOrigin → {0, 0},
 Epilog → {Line[{{gamma1, 0}, {gamma1, 1}}], Line[{{gamma2, 0}, {gamma2, 1}}]}]
```

Out[99]=



butterToHighPass

In[100]:=
```
? nma`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

In[101]:=
```
Clear[s, omega];
{poles, hs} = nma`getButterworthPolynomial[4, 1, s];
TraditionalForm@hs
```

Out[103]//TraditionalForm=

$$\frac{1.}{1.\,s^4 + 2.61313\,s^3 + 3.41421\,s^2 + 2.61313\,s + 1.}$$

convert to high pass

In[104]:=
```
gammaCutoff = 2 Pi / (0.25 * 10^(-3))
hsHigh = nma`butterToHighPass[hs, gammaCutoff, s];
TraditionalForm@hsHigh
```

Out[104]= 25 132.7

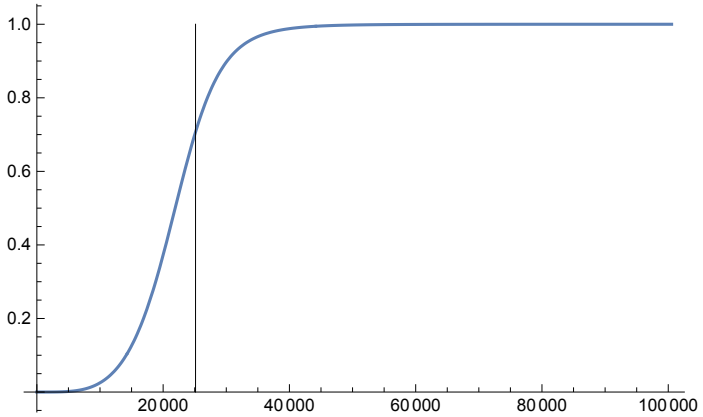
Out[106]//TraditionalForm=

$$\frac{1.}{1. + \frac{3.98988\times10^{17}}{s^4} + \frac{4.14839\times10^{13}}{s^3} + \frac{2.1566\times10^{9}}{s^2} + \frac{65\,675.}{s}}$$

Plot the magnitude spectrums

In[107]:=
```
ftNew = hsHigh /. s → (I omega);
Plot[Abs[ftNew], {omega, 0, 4 gammaCutoff}, PlotRange → All,
 AxesOrigin → {0, 0}, Epilog → Line[{{gammaCutoff, 0}, {gammaCutoff, 1}}]]
```

Out[108]=



butterToLowPass

In[109]:=
```
? nma`butterToLowPass
```

butterToLowPass[Hs,Ω_c,variable];
convert Butterworth normalized low pass analog filter Hs to low pass
analog filter with cutoff frequency Ω_c. Variable is the symbolic variable of the polynomial.

test1

make a normalized butter low pass analog filter of order 4

In[110]:=
```
Clear[s, ω];
{poles, hs} = nma`getButterworthPolynomial[4, 1, s];
TraditionalForm@hs
```

Out[112]//TraditionalForm=

$$\frac{1.}{1.\,s^4 + 2.61313\,s^3 + 3.41421\,s^2 + 2.61313\,s + 1.}$$

convert the cut off frequency to a new cutoff frequency

In[113]:=
```
Ωc = 2 Pi / (0.25 * 10^(-3));
hsLow = nma`butterToLowPass[hs, Ωc, s];
TraditionalForm@hsLow
```

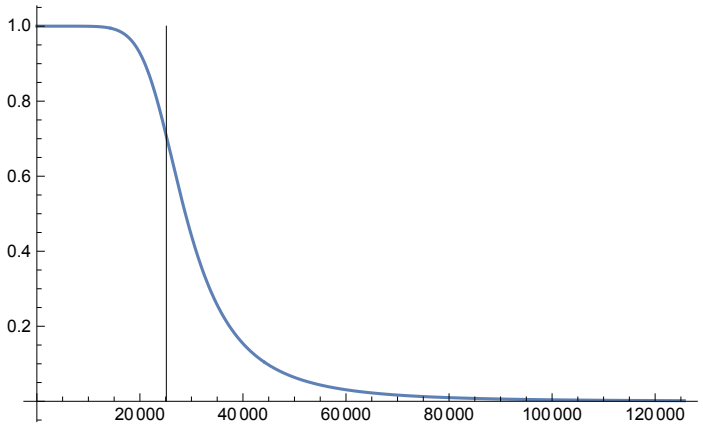
Out[115]//TraditionalForm=

$$1.\Big/\left(2.50634\times10^{-18}\,s^4 + 1.64604\times10^{-13}\,s^3 + 5.40519\times10^{-9}\,s^2 + 0.000103973\,s + 1.\right)$$

Plot the magnitude spectrums

In[116]:=
```
ftNew = hsLow /. s → (I ω);
Plot[Abs[ftNew], {ω, 0, 5 Ωc}, PlotRange → All,
 AxesOrigin → {0, 0}, Epilog → Line[{{Ωc, 0}, {Ωc, 1}}]]
```

Out[117]=



centerImageData

In[118]:=
```
? nma`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.

In[119]:=
```
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]
```

Out[120]= {150, 116}

Out[121]= 3

Out[123]= {116, 150, 3}

Out[124]=

Out[125]=

In[126]:=
```
(*center Image first*)
d = nma`centerImageData[d];
```

In[127]:=
```
Dimensions[d]
```

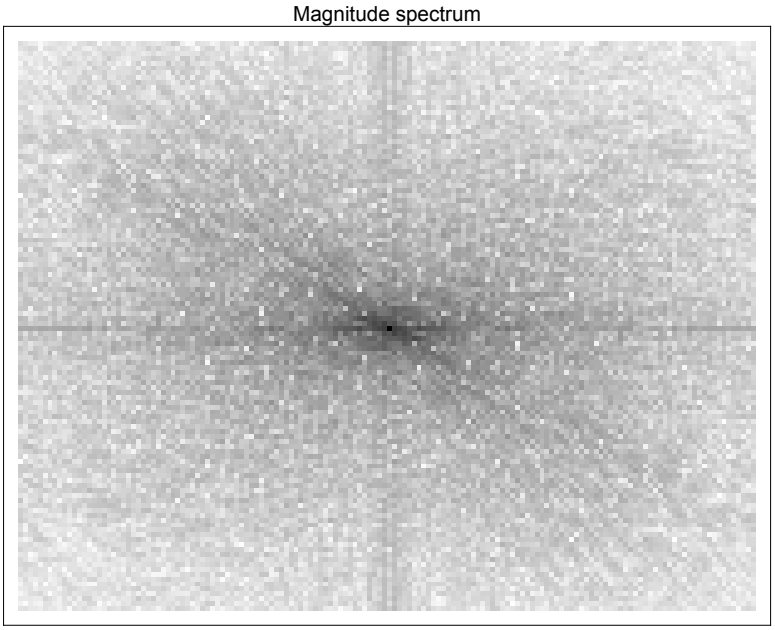
Out[127]= {116, 150}

In[128]:=
```
d[[2, 2]]
```

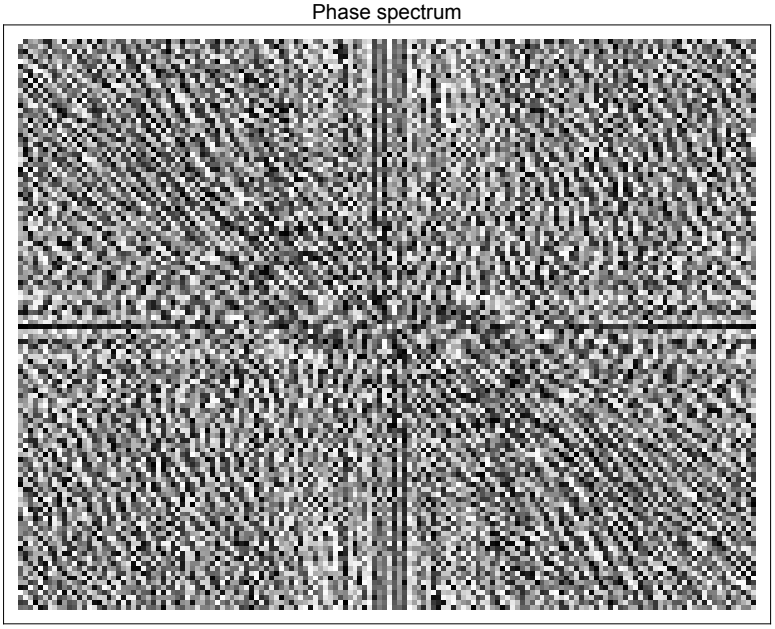
Out[128]=
```
0.654902
```

In[129]:=
```
(*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]
```

Out[130]=



Out[131]=



convertToLCD

In[132]:=
```
? nma`convertToLCD
```

convertToLCD[mat]

convert matrix to least common denominator form.

Examples:

convertToLCD[{{5/6, 1/4}, {1/7, 1/8}}]
convertToLCD[{{5/6, 1/4}}]

{{140/168,42/168},{24/168,21/168}}
{{10/12,3/12}}

Note that the result contains a HoldForm in it hence can't be used
for computation as is without first releasing the hold

In[133]:=
```
convertToLCD[{{5 / 6, 1 / 4}, {1 / 7, 1 / 8}}]
```

Out[133]=
$$\left\{\left\{\frac{140}{168}, \frac{42}{168}\right\}, \left\{\frac{24}{168}, \frac{21}{168}\right\}\right\}$$

In[134]:=
```
convertToLCD[{{5 / 6, 1 / 4}}]
```

Out[134]=
$$\left\{\left\{\frac{10}{12}, \frac{3}{12}\right\}\right\}$$

diagonalBlockMatrix

In[135]:=
```
? nma`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@nma`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}}

In[136]:=
```
(block = {{1, 2}, {3, 4}}) // MatrixForm
nma`diagonalBlockMatrix[block, 4];
MatrixForm[Normal[%]]
```

Out[136]//MatrixForm=

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

Out[138]//MatrixForm=

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

diagonalDominantMatrixQ

In[139]:=
```
? nma`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 >= for dominance of diagonal.

stict–>True, the check is made using > for dominance of diagonal

Examples:

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

nma`diagonalDominantMatrixQ[mat,strict–>False]
Out[27]= True

nma`diagonalDominantMatrixQ[mat,strict–>True]
Out[31]= False

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

In[140]:=
```
mat = {{4, 4}, {4, 4}};
nma`diagonalDominantMatrixQ[mat]
```

Out[141]=
```
True
```

In[142]:=
```
nma`diagonalDominantMatrixQ[mat, strict → True]
```

Out[142]=
```
False
```

displayHurwitz

In[143]:=
```
? nma`displayHurwitz
```

displayHurwitz[p_List,s_]

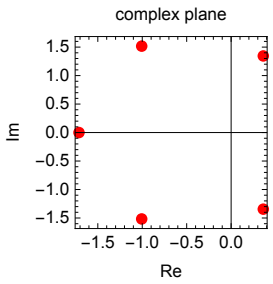
p is list of polynomials in s.
Generates Hurwitz stability Matrix and displays it.

Example:

displayHurwitz[{11 + 9*s + 8*s^2 + 6*s^3 + 3*s^4 + s^5}, s]

In[144]:=
```
nma`displayHurwitz[{11 + 9 * s + 8 * s^2 + 6 * s^3 + 3 * s^4 + s^5}, s]
```

Out[144]=

Hurwitz Matrix	$\triangle_i$	root locations	Real part of roots
$s^5 + 3\,s^4 + 6\,s^3 + 8\,s^2 + 9\,s + 11$ $\begin{pmatrix} 9 & 6 & 1 & 0 & 0 \\ 11 & 8 & 3 & 0 & 0 \\ 0 & 9 & 6 & 1 & 0 \\ 0 & 11 & 8 & 3 & 0 \\ 0 & 0 & 9 & 6 & 1 \end{pmatrix}$	$\triangle_1$ 9 $\triangle_2$ 6 $\triangle_3$ −108 $\triangle_4$ −196 $\triangle_5$ −196		−1.71185 −1.00624 −1.00624 0.362161 0.362161

fcPulseTrain

In[145]:=
```
? nma`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,nma`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

In[146]:=
```
h = 1; width = 1; period = 2;
Table[{n, nma`fcPulseTrain[n, h, width, period]}, {n, -4, 4}] // TableForm
```

Out[147]//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

formatBinary

In[148]:= `? nma`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
nma`formatBinary[4,4]
0100

nma`formatBinary[4,8]
0000 0100

In[149]:= `nma`formatBinary[4, 4]`

Out[149]= 0100

In[150]:= `nma`formatBinary[4, 8]`

Out[150]= 0000 0100

In[151]:= `nma`formatBinary[127, 8]`

Out[151]= 0111 1111

In[152]:= `nma`formatBinary[127, 16]`

Out[152]= 0000 0000 0111 1111

`nma`formatBinary[30, 16]`

0000 0000 0001 1110

generateLTVControllabilityMatrix

In[153]:=
```
? nma`generateLTVControllabilityMatrix
```

generateLTVControllabilityMatrix[A,B,t]

Generate the Linear time varying controllability matrix M defined as

M0 = B(t)
M(k+1) = –A(t) M(k) + d/dt M(k) for k=0..n–2

If the rank of the matrix is n at t>t0, where n is the size of the square matrix A,
then the system is controllable at t0.

Example

a={{Cos[t],t},{Exp[t],2}};
b={{1},{t}};
r=generateLTVControllabilitymatrix[a,b,t]

{{1, –t^2 – Cos[t]}, {t, 1 – E^t – 2*t}}

Det[r]

1 – E^t – 2*t + t^3 + t*Cos[t]

Now evaluate the about for any t>t0 where t0 is where you want to check contrability at.

If the matrix r is not square, then use MatrixRank[r/.t→1] and check for rank=n

In[154]:=
```
a = {{Cos[t], t}, {Exp[t], 2}};
b = {{1}, {t}};
r = nma`generateLTVControllabilityMatrix[a, b, t]
```

Out[156]=
$$\left\{\left\{1,\ -t^2-\text{Cos}[t]\right\},\ \left\{t,\ 1-e^t-2\,t\right\}\right\}$$

In[157]:=
```
Det[r]
```

Out[157]=
$$1-e^t-2\,t+t^3+t\,\text{Cos}[t]$$

In[158]:=
```
% /. t → 1
```

Out[158]=
$$-e+\text{Cos}[1]$$

generateLTVObservabilityMatrix

In[159]:=
```
? nma`generateLTVObservabilityMatrix
```

generateLTVObservabilityMatrix[A,C,t]

Generate the Linear time varying observability matrix M defined as

M0=C
M(k+1)= M(k) A(t) + d/dt M(k) for k=0..n−2

where n is the size of the square A matrix.

If the rank of the matrix is n at t>t0, where n is the size of the square matrix A,
then the system is observable at t0.

Example:

a={{0,1,t},{t^2,−t,Exp[t]},{1,−2,1}};
c={{1,t,Exp[−t]},{1,1,2}};
(r=generateLTVObservabilityMatrix[a,c,t])//MatrixForm
MatrixRank[r/.t→1]

In[160]:=
```
a = {{0, 1, t}, {t^2, -t, Exp[t]}, {1, -2, 1}};
c = {{1, t, Exp[-t]}, {1, 1, 2}};
(r = nma`generateLTVObservabilityMatrix[a, c, t]) // MatrixForm
MatrixRank[r /. t → 1]
```

Out[162]//MatrixForm=

$$\begin{pmatrix} 1 & t \\ 1 & 1 \\ e^{-t}+t^3 & 2-2e^{-t}-t^2 \\ 2+t^2 & -3-t \\ -e^{-t}+t+e^t t+3t^2+t^2\left(2-2e^{-t}-t^2\right) & 3e^{-t}-2t+t^3-2\left(t+e^t t\right)-t\left(2-2e^{-t}-t^2\right) \\ 2+e^t+3t+\left(-3-t\right)t^2 & 1-\left(-3-t\right)t+t^2-2\left(2+e^t+t\right) \end{pmatrix} \quad \begin{matrix} 1+e^t+t+2e^t \\ 3+2e^t \end{matrix}$$

Out[163]= 3

getButterworthPolynomial

In[164]:=
```
? nma`getButterworthPolynomial
```

getButterworthPolynomial[order, cuttoff, variable, options];

order: filter order
cuttoff: butterworth cuttoff frequency

returns {poles,polynomial}

poles: list of stable poles of the Butterworth H(s)
polynomial: Buttworth polynomial in symbolic expression
expressed in the variable supplied. The form of the polynomial depends on options used.

Possible values for options are
form–>"linear" returns H(s) in the form linear terms in the variable
form–>"quadratic" returns H(s) in quadratic terms in the variable
If this option is not selected, then H(s) will be returned as one polynomial.

Example usage

Example 1
––––––––––

```
Clear[s,form];
order=2;
cuttoff=.2 Pi;
{poles,hs}=nma`getButterworthPolynomial[order,cuttoff,s];
poles
```

{–0.44428829381583657`+0.44428829381583657` ⅈ,–0.44428829381583657`–0.44428829381583657` ⅈ}

```
TraditionalForm@hs
```

$$\frac{0.394784}{s^2+0.888577\,s+0.394784}$$

Example 2
––––––––––

```
{poles,hs}=nma`getButterworthPolynomial[order,cuttoff,s,form→"linear"];

poles
```

{–0.44428829381583657`+0.44428829381583657` ⅈ,–0.44428829381583657`–0.44428829381583657` ⅈ}

```
TraditionalForm@hs
```

$$\frac{0.444288\,i}{s+(0.444288+0.444288\,i)}-\frac{0.444288\,i}{s+(0.444288-0.444288\,i)}$$

Example 3
––––––––––

```
{poles,hs}=nma`getButterworthPolynomial[order,cuttoff,s,form→"quadratic"];
poles
```

{–0.44428829381583657`+0.44428829381583657` ⅈ,–0.44428829381583657`–0.44428829381583657` ⅈ}

```
TraditionalForm@hs
```

$$\frac{0.394784}{s^2+0.888577\,s+0.394784}$$

Generate butterworth H(s) given order and cuttoff

test1

In[165]:=
```
Clear[s, form];
order = 2;
cuttoff = .2 Pi;
{poles, hs} = nma`getButterworthPolynomial[order, cuttoff, s];
poles
TraditionalForm@hs
```

Out[169]= $\{-0.444288 + 0.444288\,i,\ -0.444288 - 0.444288\,i\}$

Out[170]//TraditionalForm=

$$\frac{0.394784}{s^2 + 0.888577\,s + 0.394784}$$

test2

In[171]:=
```
{poles, hs} = nma`getButterworthPolynomial[order, cuttoff, s, form → "linear"];
poles
TraditionalForm@hs
```

Out[172]= $\{-0.444288 + 0.444288\,i,\ -0.444288 - 0.444288\,i\}$

Out[173]//TraditionalForm=

$$\frac{0. + 0.444288\,i}{s + (0.444288 + 0.444288\,i)} - \frac{0. + 0.444288\,i}{s + (0.444288 - 0.444288\,i)}$$

test3

In[174]:=
```
{poles, hs} = nma`getButterworthPolynomial[order, cuttoff, s, form → "quadratic"];
poles
TraditionalForm@hs
```

Out[175]= $\{-0.444288 + 0.444288\,i,\ -0.444288 - 0.444288\,i\}$

Out[176]//TraditionalForm=

$$\frac{0.394784}{s^2 + 0.888577\,s + 0.394784}$$

test4 find normalized Butterworth

In[177]:=
```
{poles, hs} = nma`getButterworthPolynomial[5, 1, s, form → "quadratic"];
poles
TraditionalForm@hs
```

Out[178]=
$$\left\{\frac{1}{4}\left(1-\sqrt{5}\right)+i\sqrt{\frac{5}{8}+\frac{\sqrt{5}}{8}},\ \frac{1}{4}\left(-1-\sqrt{5}\right)+i\sqrt{\frac{5}{8}-\frac{\sqrt{5}}{8}},\right.$$
$$\left.-1,\ \frac{1}{4}\left(-1-\sqrt{5}\right)-i\sqrt{\frac{5}{8}-\frac{\sqrt{5}}{8}},\ \frac{1}{4}\left(1-\sqrt{5}\right)-i\sqrt{\frac{5}{8}+\frac{\sqrt{5}}{8}}\right\}$$

Out[179]//TraditionalForm=
$$\frac{-0.276393\,s-0.894427}{1.\,s^2+0.618034\,s+1.}-\frac{1.61803\,s}{1.\,s^2+1.61803\,s+1.}+\frac{1.89443}{s+1}$$

test 5, even

In[180]:=
```
{poles, hs} = nma`getButterworthPolynomial[4, 1, s, form → "quadratic"];
poles
TraditionalForm@hs
```

Out[181]=
$$\left\{i\,\text{Cos}\left[\frac{\pi}{8}\right]-\text{Sin}\left[\frac{\pi}{8}\right],\ -\text{Cos}\left[\frac{\pi}{8}\right]+i\,\text{Sin}\left[\frac{\pi}{8}\right],\ -\text{Cos}\left[\frac{\pi}{8}\right]-i\,\text{Sin}\left[\frac{\pi}{8}\right],\ -i\,\text{Cos}\left[\frac{\pi}{8}\right]-\text{Sin}\left[\frac{\pi}{8}\right]\right\}$$

Out[182]//TraditionalForm=
$$\frac{-0.92388\,s-0.707107}{1.\,s^2+0.765367\,s+1.}+\frac{0.92388\,s+1.70711}{1.\,s^2+1.84776\,s+1.}$$

---

getControllableForm

In[183]:=
```
? nma`getControllableForm
```

getControllableForm[A,B]

Takes A,B state space matrices and returns the controllable canonical form.
Example

a={{0,0,1,0},{0,0,0,1},{−2,−1,0,0},{1,−1,0,0}};
b={{0},{0},{1},{0}};
{A0,B0}=getControllableForm[a,b]

In[184]:=
```
a = {{0, 0, 1, 0}, {0, 0, 0, 1}, {-2, -1, 0, 0}, {1, -1, 0, 0}};
MatrixForm[a]
```

Out[185]//MatrixForm=
$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -2 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{pmatrix}$$

In[186]:=
```
b = {{0}, {0}, {1}, {0}};
MatrixForm[b]
```

Out[187]//MatrixForm=

$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

In[188]:=
```
{A0, B0} = getControllableForm[a, b];
MatrixForm[A0]
```

Out[189]//MatrixForm=

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 0 & -3 & 0 \end{pmatrix}$$

In[190]:=
```
MatrixForm[B0]
```

Out[190]//MatrixForm=

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

getMagnitudeOfPulseTrainFourierCoeff

In[191]:=
```
? nma`getMagnitudeOfPulseTrainFourierCoeff
```

getMagnitudeOfPulseTrainFourierCoeff[t0,period,range,dutyCycle,nCoeff,h]

Example

nma`getMagnitudeOfPulseTrainFourierCoeff[0,1,10,1,5,3]

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

In[192]:=
```
nma`getMagnitudeOfPulseTrainFourierCoeff[0, 1, 10, 1, 5, 3]
```

Out[192]=
```
{{-4, 0}, {-3, 0}, {-2, 0}, {-1, 0}, {0, 3}, {1, 0}, {2, 0}, {3, 0}, {4, 0}}
```

getMaxPadding

In[193]:=
```
? nma`getMaxPadding
```

getMaxPadding[p_List]

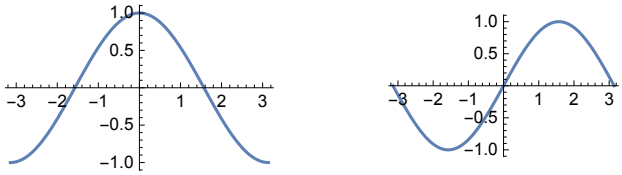
finds maximum image padding for
a list of plots.

writtenby Hypnotoad from stackoverflow
http://mathematica.stackexchange.com/questions/8241/generating-a-broken-or-snipped-axis-in-listplot/8245#8245

p1=Plot[Sin[x],{x,-Pi,Pi},ImagePadding->30]
p2=Plot[Cos[x],{x,-Pi,Pi},ImagePadding->20]
Grid[{{p2, p1}} /. Graphics[x__] :>
  Graphics[x, ImagePadding -> getMaxPadding[{p1, p2}]]]

In[194]:=
```
p1 = Plot[Sin[x], {x, -Pi, Pi}, ImagePadding -> 30];
p2 = Plot[Cos[x], {x, -Pi, Pi}, ImagePadding -> 20];
Grid[{{p2, p1}} /. Graphics[x__] :>
  Graphics[x, ImagePadding -> getMaxPadding[{p1, p2}]]]
```

Out[196]=



getPhaseOfPulseTrainFourierCoeff

In[197]:=
```
? nma`getPhaseOfPulseTrainFourierCoeff
```

getPhaseOfPulseTrainFourierCoeff[delay,period,range,dutyCycle,nCoeff]

Example

  nma`getPhaseOfPulseTrainFourierCoeff[0,1,10,1,5]

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

In[198]:=
```
nma`getPhaseOfPulseTrainFourierCoeff[0, 1, 10, 1, 5]
```

Out[198]=
```
{{-4, 0}, {-3, 0}, {-2, 0}, {-1, 0}, {0, 0}, {1, 0}, {2, 0}, {3, 0}, {4, 0}}
```

getPositiveAndNegativeTerms

In[199]:=
```
? nma`getPositiveAndNegativeTerms
```

{neg,pos}=getPositiveAndNegativeTerms[expr]
Takes an expression expr and returns the terms in it which have negative sign and the terms which have postive 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.

In[200]:=
```
Clear[c, b, d, h]
expr = 3 - c + b + d - h
{neg, pos} = nma`getPositiveAndNegativeTerms[expr]
```

Out[201]= 3 + b − c + d − h

Out[202]= {{−c, −h}, {3, b, d}}

getStateGainVector

In[203]:=
```
? nma`getStateGainVector
```

getStateGainVector[A,B,designEigenvalues,debugFlag]

Takes state space A,B, as in x'=Ax+Bu and list of design eigenvalues
and returns state feedback gain vector k, such that A+B*k have the desired eigenvalues.

If debugFlag is True, then design steps are printed.

Example:

a={{0,0,1,0},{0,0,0,1},{−2,1,0,0},{1,−1,0,0}};
b={{0},{0},{1},{0}};
desiredEigenValues={−1+I,−1+I,−1−I,−1−I};
k = getStateGainVector[a,b,desiredEigenValues,True]
{−5,2,−4,−4}

In[204]:=
```
a = {{0, 0, 1, 0}, {0, 0, 0, 1}, {-2, 1, 0, 0}, {1, -1, 0, 0}};
b = {{0}, {0}, {1}, {0}};
desiredEigenValues = {-1 + I, -1 + I, -1 - I, -1 - I};
nma`getStateGainVector[a, b, desiredEigenValues, False]
```

Out[207]= $\{-5, 2, -4, -4\}$

In[208]:=
```
a = {{0, 0, 1, 0}, {0, 0, 0, 1}, {-2, 1, 0, 0}, {1, -1, 0, 0}};
b = {{0}, {0}, {1}, {0}};
desiredEigenValues = {-1 + I, -1 + I, -1 - I, -1 - I};
nma`getStateGainVector[a, b, desiredEigenValues, True]
```

Out[211]= $\{-5, 2, -4, -4\}$

---

hurwitz

In[212]:=
```
? nma`hurwitz
```

hurwitz[p_,s_]

Take one polynomial in s and reurns the Hurwitz stability matrix.

Example:
hurwitz[11 + 9*s + 8*s^2 + 6*s^3 + 3*s^4 + s^5, s];
Grid[%, Frame -> All]

In[213]:=
```
nma`hurwitz[11 + 9 * s + 8 * s^2 + 6 * s^3 + 3 * s^4 + s^5, s];
Grid[%, Frame -> All]
```

Out[214]=

9	6	1	0	0
11	8	3	0	0
0	9	6	1	0
0	11	8	3	0
0	0	9	6	1

kharitonovPoly

In[215]:= `? nma`kharitonovPoly`

kharitonovPoly[w_List,s_]

Take list of intervals and returns the four Kharitonov Polynomials in s.
Example,

(nma`kharitonovPoly[{{1,2},{9,10}},s])//TableForm

$1+9\,s$
$2+10\,s$
$2+9\,s$
$1+10\,s$

Notice that the order for input is like this: limits of a0 followed by limits of a1 followed by limits of a2 and so on.
Hence to find Kharitonov polynomials for [1,2]+[9,10]s we enter it as shown above.

Example 2: For polynomial [1,2]+[9,10]s+[−1,3]s^2 the call is

(nma`kharitonovPoly[{{1,2},{9,10},{−1,3}},s])//TableForm

$1+9\,s+3\,s^2$
$2+10\,s-s^2$
$2+9\,s-s^2$
$1+10\,s+3\,s^2$

In[216]:=
```
p = nma`kharitonovPoly[{{11, 12}, {9, 10}, {7, 8}, {5, 6}, {3, 4}, {1, 2}}, s];
TableForm[p]
```

Out[217]//TableForm=

$11 + 9\,s + 8\,s^2 + 6\,s^3 + 3\,s^4 + s^5$
$12 + 10\,s + 7\,s^2 + 5\,s^3 + 4\,s^4 + 2\,s^5$
$12 + 9\,s + 7\,s^2 + 6\,s^3 + 4\,s^4 + s^5$
$11 + 10\,s + 8\,s^2 + 5\,s^3 + 3\,s^4 + 2\,s^5$

leadingDet

In[218]:=
```
? nma`leadingDet
```

leadingDet[h_]

Take a Hurwitz matrix returned by hurwitz[polynomial,s] and returns the leading minors determinants. They must all be postive for stable polynomial.

Example

hurwitz[11 + 9*s + 8*s^2 + 6*s^3 + 3*s^4 + s^5, s]
leadingDet[%]
{9,6,−108,−196,−196}

In[219]:=
```
nma`hurwitz[11 + 9 * s + 8 * s^2 + 6 * s^3 + 3 * s^4 + s^5, s]
nma`leadingDet[%]
```

Out[219]= {{9, 6, 1, 0, 0}, {11, 8, 3, 0, 0}, {0, 9, 6, 1, 0}, {0, 11, 8, 3, 0}, {0, 0, 9, 6, 1}}

Out[220]= {9, 6, −108, −196, −196}

linspace

In[221]:=
```
? nma`linspace
```

generate linearly spaced list.
linespace(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.

In[222]:=
```
nma`linspace[2, 3, 5]
```

Out[222]= $\left\{2, \frac{9}{4}, \frac{5}{2}, \frac{11}{4}, 3\right\}$

In[223]:=
```
nma`linspace[2 + I, 3, 5]
```

Out[223]= $\left\{2 + i, \frac{9}{4} + \frac{3\,i}{4}, \frac{5}{2} + \frac{i}{2}, \frac{11}{4} + \frac{i}{4}, 3\right\}$

In[224]:=
```
nma`linspace[2 + I, 3, 5] // N
```

Out[224]= {2. + 1. i, 2.25 + 0.75 i, 2.5 + 0.5 i, 2.75 + 0.25 i, 3.}

In[225]:=
```
nma`linspace[2, 3, 2] // N
```

Out[225]= {2., 3.}

In[226]:=
```
nma`linspace[2, 2.1, 10] // N
```

Out[226]=
```
{2., 2.01111, 2.02222, 2.03333, 2.04444, 2.05556, 2.06667, 2.07778, 2.08889, 2.1}
```

makePulseTrain

In[227]:=
```
? nma`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*)
]
```

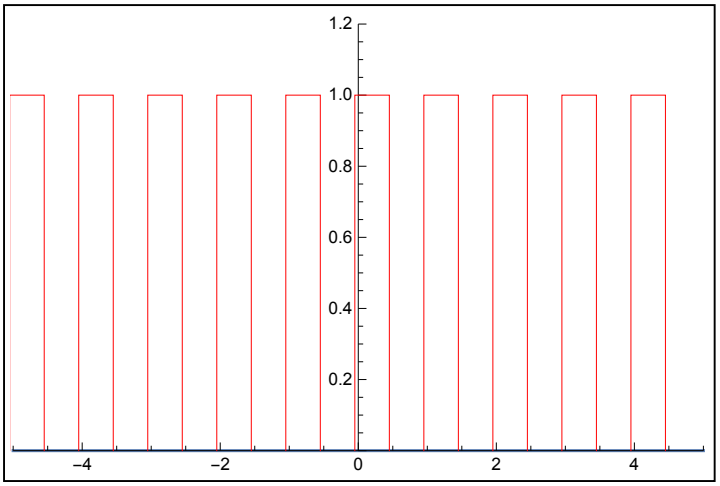
Returns List of Lines.

EXAMPLE USAGE:

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

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

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

In[228]:=
```
delay = .2; period = 1; range = 5.; tao = .5;  h = 1;
currentPulses = nma`makePulseTrain[delay, period, range, tao, h];
Framed@Plot[0, {x, -range, range},
  PlotRange → {{-range, range}, {0, 1.2}}, Epilog → {Thin, Red, currentPulses}]
```

Out[230]=



matrix2DLexOrdering

In[231]:=
```
? nma`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 sequentional Lexicographical order

Example

a=Table[0,{i,3},{j,3}];
nma`matrix2DLexOrdering[a]
Out[310]= {{{1,3},{2,3},{3,3}},{{1,2},{2,2},{3,2}},{{1,1},{2,1},{3,1}}}

In[232]:=
```
(c = Table[0, {i, 3}, {j, 1}]) // MatrixForm
```

Out[232]//MatrixForm=

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

```
{A0, B0} = nma`matrix2DLexOrdering[c];
A0 // MatrixForm
```

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

```
B0 // MatrixForm
```

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

```
(c = Table[0, {i, 3}, {j, 2}]) // MatrixForm
{A0, B0} = nma`matrix2DLexOrdering[c];
A0 // MatrixForm
```

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

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

```
B0 // MatrixForm
```

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

```
(c = Table[0, {i, 3}, {j, 3}]) // MatrixForm
{A0, B0} = nma`matrix2DLexOrdering[c];
A0 // MatrixForm
```

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

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

```
B0 // MatrixForm
```

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

matrix2DredBlackPositions

In[233]:=
```
? nma`matrix2DredBlackPositions
```

matrix2DredBlackPositions[nRow,nCol,opt] 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.

opt: internal–>True, is the default, which says to apply red/black only to the
internal nodes, and not to the boundaries.

Example, put a '1' where a red should be, and put a '–1' where a black should be

```
AnRow = 3;
nCol  = 3;
a     = Table[0,{i,nRow},{j,nCol}];
{r,b}=nma`matrix2DredBlackPositions[nRow,nCol];
(a[[Sequence@@#]]=1)&/@r;
(a[[Sequence@@#]]=-1)&/@b;
MatrixForm[a]
```

In[234]:=
```
nRow = 3;
nCol = 3;
A0 = Table[0, {i, nRow}, {j, nCol}];
{r, b} = nma`matrix2DredBlackPositions[nRow, nCol, internal → True];
(A0[[Sequence @@ #]] = 1) & /@ r;
(A0[[Sequence @@ #]] = -1) & /@ b;
MatrixForm[A0]
```

Out[240]//MatrixForm=

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

In[241]:=
```
{r, b} = nma`matrix2DredBlackPositions[nRow, nCol, internal → False];
(A0[[Sequence @@ #]] = 1) & /@ r;
(A0[[Sequence @@ #]] = -1) & /@ b;
MatrixForm[A0]
```

Out[244]//MatrixForm=

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

In[245]:=
```
nRow = 2;
nCol = 5;
A0 = Table[0, {i, nRow}, {j, nCol}];
{r, b} = nma`matrix2DredBlackPositions[nRow, nCol, internal → False];
(A0[[Sequence @@ #]] = 1) & /@ r;
(A0[[Sequence @@ #]] = -1) & /@ b;
MatrixForm[A0]
```

Out[251]//MatrixForm=

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

To apply the 2D Laplacian on a matrix using red-black

In[252]:=
```
mask = {{0, 1, 0}, {1, 0, 1}, {0, 1, 0}};
nRow = 5; nCol = 5;
A0 = Table[RandomInteger[10], {i, nRow}, {j, nCol}];
MatrixForm[A0]
```

Out[255]//MatrixForm=

$$\begin{pmatrix} 8 & 3 & 6 & 1 & 3 \\ 10 & 0 & 2 & 8 & 1 \\ 0 & 0 & 0 & 5 & 8 \\ 7 & 10 & 3 & 8 & 0 \\ 1 & 8 & 10 & 3 & 1 \end{pmatrix}$$

In[256]:=
```
{r, b} = nma`matrix2DredBlackPositions[nRow, nCol, internal → True];
Table[A0[[Sequence @@ r[[i]] ]] =
    1./4 Total[mask * A0[[ r[[i, 1]] - 1 ;; r[[i, 1]] + 1, r[[i, 2]] - 1 ;; r[[i, 2]] + 1 ]], 2],
  {i, 1, Length[r]}];
MatrixForm[A0]
Table[A0[[Sequence @@ b[[i]] ]] =
    1./4 Total[mask * A0[[ b[[i, 1]] - 1 ;; b[[i, 1]] + 1, b[[i, 2]] - 1 ;; b[[i, 2]] + 1 ]], 2],
  {i, 1, Length[b]}];
MatrixForm[
 A0]
```

Out[258]//MatrixForm=

$$\begin{pmatrix} 8 & 3 & 6 & 1 & 3 \\ 10 & 3.75 & 2 & 2.25 & 1 \\ 0 & 0 & 2.5 & 5 & 8 \\ 7 & 4.5 & 3 & 2.75 & 0 \\ 1 & 8 & 10 & 3 & 1 \end{pmatrix}$$

Out[260]//MatrixForm=

$$\begin{pmatrix} 8 & 3 & 6 & 1 & 3 \\ 10 & 3.75 & 3.625 & 2.25 & 1 \\ 0 & 2.6875 & 2.5 & 3.875 & 8 \\ 7 & 4.5 & 4.9375 & 2.75 & 0 \\ 1 & 8 & 10 & 3 & 1 \end{pmatrix}$$

Try to see how to make this below work (currently I get error, for later....)

```
r
(A0[[Sequence @@ #]] =
    1/4 Total[mask * A0[[#[[1]] - 1 ;; #[[1]] + 1, #[[2]] - 1 ;; #[[1]] + 1]], 2]) & /@ r ;
MatrixForm[
 A0]
```

```
{{2, 2}, {2, 4}, {3, 3}, {4, 2}, {4, 4}}
```

```
$Aborted
```

$$\begin{pmatrix} 3 & 4 & 3 & 9 & 8 \\ 4 & 3.92188 & 3.4375 & 4.5 & 8 \\ 6 & 4.25 & 2.25 & 2.3125 & 0 \\ 9 & 4.75 & 3.625 & 2.5 & 3 \\ 9 & 2 & 5 & 6 & 4 \end{pmatrix}$$

numIt

In[261]:=
```
? nma`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 formating*)

EXAMPLE USAGE:

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

23.3

In[262]:=
```
nma`numIt[23.321, 5, 1, 14]
nma`numIt[23.321, 5, 2, 10]
```

Out[262]=
```
23.3
```

Out[263]=
```
23.32
```

numItse

In[264]:=
```
? nma`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
nma`numItse[234.4,6,4,10,12]

2.3440e2

In[265]:=
```
RandomReal[1]
nma`numItse[%, 5, 3, 10]
```

Out[265]=
```
0.0895692
```

Out[266]=
```
8.957e-2
```

In[267]:=
```
RandomReal[10]
nma`numItse[%, 5, 3, 10]
```

Out[267]=
```
5.3585
```

Out[268]=
```
5.358e
```

plotFourierTransform

In[269]:=
```
? nma`plotFourierTransform
```

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

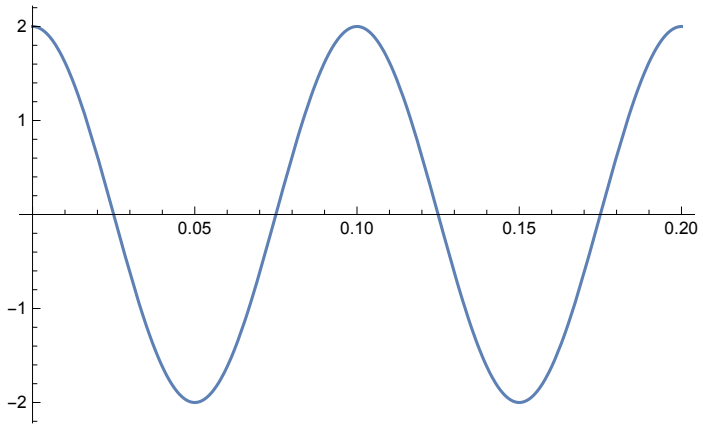
EXAMPLE

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

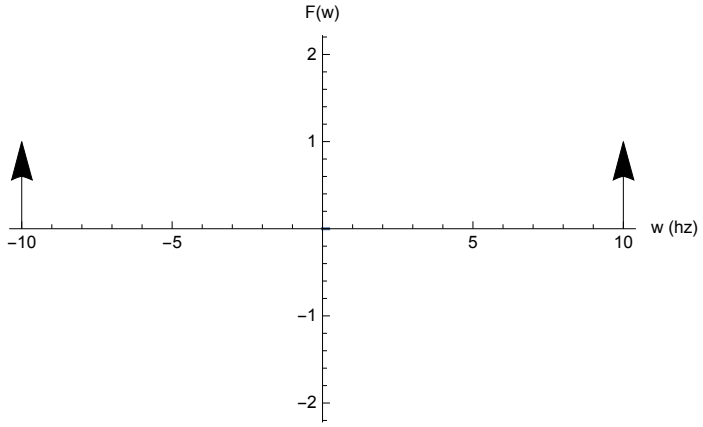
nma`plotFourierTransform[%,t,0,2 period,−2,2,Small]

In[270]:=
```
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@nma`plotFourierTransform[%, w, 0, 2 period, -2, 2, Large],
 AxesLabel → {"w (hz)", "F(w)"}]
```

Out[273]=



Out[275]=



poissonMatrixOnSquareDirichlet

In[276]:=
```
? nma`poissonMatrixOnSquareDirichlet
```

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

Example

nma`poissonMatrixOnSquareDirichlet[3,5]//MatrixForm

In[277]:=
```
nma`poissonMatrixOnSquareDirichlet[3, 5] // MatrixForm
```

Out[277]//MatrixForm=

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

In[278]:=
```
nma`poissonMatrixOnSquareDirichlet[2, 5] // MatrixForm
```

Out[278]//MatrixForm=

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

In[279]:=
```
nma`poissonMatrixOnSquareDirichlet[4, 5] // MatrixForm
```

Out[279]//MatrixForm=

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

polarForm

In[280]:=
```
? nma`polarForm
```

polarForm wrapper to make polar form for complex number
written by Xerxes. see

http://mathematica.stackexchange.com/questions/20570/wrapper-for-inexact-numeric-complex-numbers-that-maintains-polar-form

Sqrt[5] E^(I ArcTan[2]) // N // polarForm
(* 2.23607E^(I1.10715) *)

In[281]:=
```
nma`polarForm[3.0 + 2 I]
```

Out[281]=
$$3.60555\, e^{i\, 0.588003}$$

quantizationLevel

In[282]:=
```
? nma`quantizationLevel
```

quantizationLevel[x,mp,nBits]

Find the quantization level for an analog value x. mp is the
maximum allowed level and nBits is the number of bits available for encoding x.

Example

nBits=8;
mp=16;
x=-6.5;

quantizationLevel[-8.7,16,nBits]

58

In[283]:=
```
nBits = 8;
mp = 16;
x = -6.5;

nma`quantizationLevel[-8.7, 16, nBits]
```

Out[286]=
58

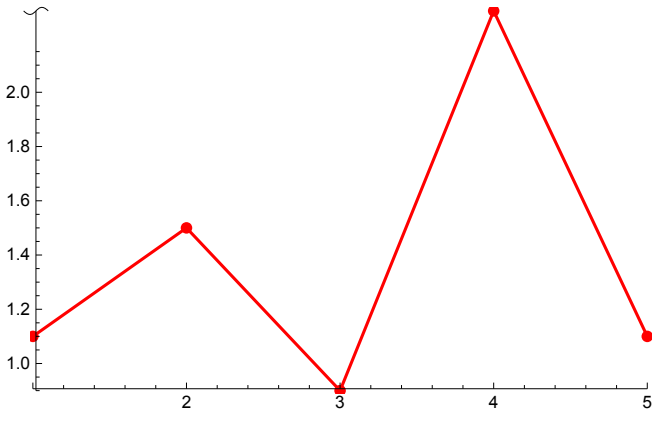
snip

In[287]:= `? nma`snip

```
snip[pos_]

writtenby Hypnotoad
http://mathematica.stackexchange.com/questions/8241/generating-a-broken-or-snipped-axis-in-listplot/8245#8245

The function snip[x] places the mark on the axes at relative
position x (0 and 1 being the ends)

example:

data1 = {{1, 1.1}, {2, 1.5}, {3, 0.9}, {4, 2.3}, {5, 1.1}};
data2 = {{1, 1001.1}, {2, 1001.5}, {3, 1000.9}, {4, 1002.3}, {5,
    1001.1}};
p1 = ListPlot[data1, PlotRange -> All, Joined -> True, Mesh -> Full,
   PlotStyle -> Red, AxesStyle -> {None, snip[1]},
   PlotRangePadding -> None, ImagePadding -> 40];
p2 = ListPlot[data2, PlotRange -> All, Joined -> True, Mesh -> Full,
   PlotStyle -> Blue, Axes -> {False, True},
   AxesStyle -> {None, snip[0]}, PlotRangePadding -> None,
   ImagePadding -> 40];

Column[{p2, p1} /.
  Graphics[x__] :>
   Graphics[x, ImagePadding -> getMaxPadding[{p1, p2}],
    ImageSize -> 400]]
```

In[288]:=
```
data1 = {{1, 1.1}, {2, 1.5}, {3, 0.9}, {4, 2.3}, {5, 1.1}};
data2 = {{1, 1001.1}, {2, 1001.5}, {3, 1000.9}, {4, 1002.3}, {5, 1001.1}};
p1 = ListPlot[data1, PlotRange -> All, Joined -> True, Mesh -> Full,
  PlotStyle -> Red, AxesStyle -> {None, snip[1]},
  PlotRangePadding -> None, ImagePadding -> 40];

p2 = ListPlot[data2, PlotRange -> All, Joined -> True, Mesh -> Full,
  PlotStyle -> Blue, Axes -> {False, True},
  AxesStyle -> {None, snip[0]}, PlotRangePadding -> None,
  ImagePadding -> 40];

Column[{p2, p1} /. Graphics[x__] :>
   Graphics[x, ImagePadding -> getMaxPadding[{p1, p2}], ImageSize -> 400]]
```

Out[292]=



str

In[293]:=
```
? nma`str
```

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

In[296]:=
```
test := Module[{a = 4}, Print["a=", nma`str[a]]]
test
```

stripGraphMouseAppearance

In[298]:=
```
? nma`stripGraphMouseAppearance
```

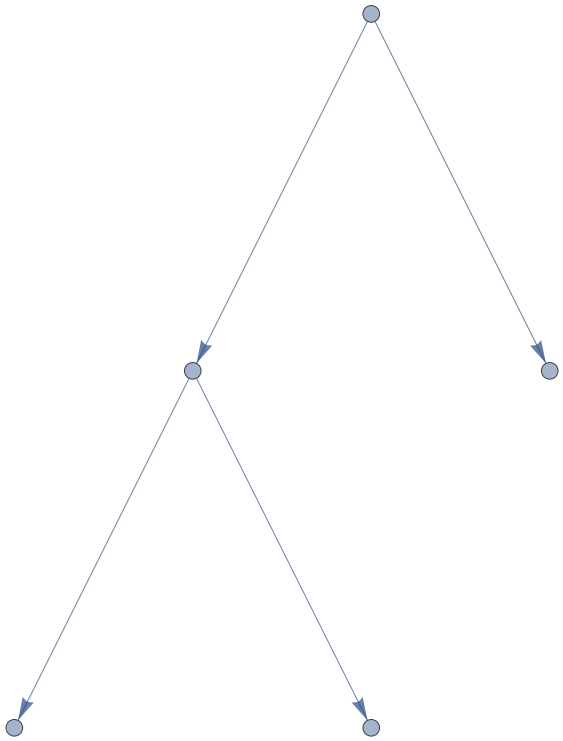
stripGraphMouseAppearance[x_Graph]
puts normal arrow looking mouse appearance on Grph plots.

written by John Fultz, see
see http://mathematica.stackexchange.com/questions/1039/mouseappearance-and-cursor-problems
example:

```
stripGraphMouseAppearance[
 TreeGraph[{1 -> 2, 1 -> 3, 2 -> 4, 2 -> 5},
 VertexLabels -> {5 -> Placed["five", Tooltip]}, ImagePadding -> 20]
]
```

In[299]:=
```
nma`stripGraphMouseAppearance[
 TreeGraph[{1 -> 2, 1 -> 3, 2 -> 4, 2 -> 5},
 VertexLabels -> {5 -> Placed["five", Tooltip]}, ImagePadding -> 20]
]
```

Out[299]=



toOffsetBinary

In[300]:=
```
? nma`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 reprentation, use the funtion formatBinary[] in this package.

Example

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

In[301]:=
```
mp = 16;
nBits = 8;
x = -8.7
{delta, quantized, code} = nma`toOffsetBinary[x, mp, nBits]
formatBinary[code, nBits]
```

Out[303]= -8.7

Out[304]= {0.125, -8.75, 58}

Out[305]= 0011 1010

In[306]:=
```
mp = 1;
nBits = 4;
x = 1 / 15.
{delta, quantized, code} = nma`toOffsetBinary[x, mp, nBits]
formatBinary[code, nBits]
```

Out[308]= 0.0666667

Out[309]= {0.125, 0.125, 9}

Out[310]= 1001

In[311]:=
```
mp = 1;
nBits = 4;
x = -100;
{delta, quantized, code} = nma`toOffsetBinary[x, mp, nBits]
formatBinary[code, nBits]
```

Out[314]= {0.125, -1., 0}

Out[315]= 0000

toOnesComplement

In[316]:=
```
? nma`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 reprentation, use the funtion formatBinary[] in this package.

In[317]:=
```
mp = 10;
nBits = 8;
x = N[-6.5]
{delta, quantized, code} = nma`toOnesComplement[x, mp, nBits]
formatBinary[code, nBits]
```

Out[319]= -6.5

Out[320]= {0.078125, -6.48438, 172}

Out[321]= 1010 1100

In[322]:=
```
mp = 1;
nBits = 4;
x = -1 / 17.
{delta, quantized, code} = nma`toOnesComplement[x, mp, nBits]
nma`formatBinary[code, nBits]
```

Out[324]= -0.0588235

Out[325]= {0.125, 0., 15}

Out[326]= 1111

In[327]:=
```
mp = 1;
nBits = 4;
x = 13 / 17.
{delta, quantized, code} = nma`toOnesComplement[x, mp, nBits]
formatBinary[code, nBits]
```

Out[329]= 0.764706

Out[330]= {0.125, 0.75, 6}

Out[331]= 0110

In[332]:=
```
mp = 16;
nBits = 8;
x = -8.7;
{delta, quantized, code} = nma`toOnesComplement[x, mp, nBits]
formatBinary[code, nBits]
```

Out[335]= {0.125, -8.75, 185}

Out[336]= 1011 1001

In[337]:=
```
mp = 1;
nBits = 4;
x = 100;
{delta, quantized, code} = nma`toOnesComplement[x, mp, nBits]
formatBinary[code, nBits]
```

Out[340]= {0.125, 0.875, 7}

Out[341]= 0111

In[342]:=
```
mp = 1;
nBits = 4;
x = -100;
{delta, quantized, code} = nma`toOnesComplement[x, mp, nBits]
formatBinary[code, nBits]
```

Out[345]= {0.125, -0.875, 8}

Out[346]= 1000

In[347]:=
```
delta = 1 / 16.;
data = Table[{n * delta, formatBinary[nma`toOnesComplement[n * delta, 1, 4][[3]], 4]},
   {n, 2^4 - 1, -2^4, -1}] // TableForm
```

Out[348]//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.	1111
-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

toSignMagnitude

In[349]:=
```
? nma`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 reprentation, use the funtion formatBinary[] in this package.

Example

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

In[350]:=
```
mp = 16;
nBits = 8;
x = -8.7
{delta, quantized, code} = nma`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]
```

Out[352]= -8.7

Out[353]= {0.125, -8.75, 198}

Out[354]= 1100 0110

In[355]:=
```
mp = 10;
nBits = 8;
x = -6.5;
{delta, quantized, code} = nma`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]
```

Out[358]= {0.078125, -6.48438, 211}

Out[359]= 1101 0011

In[360]:=
```
mp = 10;
nBits = 8;
x = 6.5;
{delta, quantized, code} = nma`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]
```

Out[363]= {0.078125, 6.48438, 83}

Out[364]= 0101 0011

In[365]:=
```
mp = 10;
nBits = 8;
x = 600;
{delta, quantized, code} = nma`toSignMagnitude[x, mp, nBits]
formatBinary[code, nBits]
```

Out[368]= {0.078125, 9.92188, 127}

Out[369]= 0111 1111

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

{0.078125, -9.92188, 255}

1111 1111

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

{0.125, -8.75, 198}

1100 0110

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

{0.078125, 9.92188, 127}

0111 1111

toTwoComplement

In[370]:=
```
? nma`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 reprentation, use the funtion formatBinary[] in this package.

In[371]:=
```
mp = 16;
nBits = 8;
x = -8.7
{delta, quantized, code} = nma`toTwoComplement[x, mp, nBits]
formatBinary[code, nBits]
```

Out[373]= -8.7

Out[374]= {0.125, -8.75, 186}

Out[375]= 1011 1010

In[376]:=
```
mp = 1;
nBits = 4;
x = -8.7
{delta, quantized, code} = nma`toTwoComplement[x, mp, nBits]
formatBinary[code, nBits]
```

Out[378]= -8.7

Out[379]= {0.125, -1., 8}

Out[380]= 1000

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

In[381]:= `? nma`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.

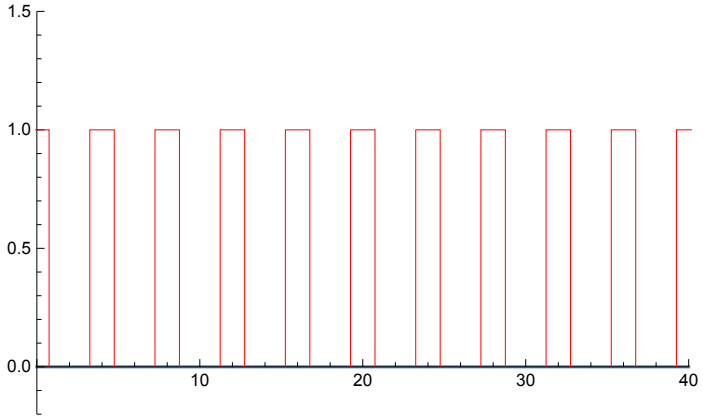
EXAMPLE USAGE:

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

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

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

test on a square pulse

In[382]:=
```
h = 1; width = 1.5; period = 4;  n = 10; delay = 0; range = 10 * period;
Clear[t, x];

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

Show[{Plot[0, {x, 0, range},
   PlotRange → {{0, range}, {-(.2 h), 1.5 h}}, Epilog → {Thin, Red, currentPulses}]}]
```

Out[385]=



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

In[389]:=
```
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

In[392]:=
```
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
]
```

Out[392]= $x^{-2} + x^{-3}$

Out[393]//TraditionalForm=
$$x^{-2} + x^{-3} + 1$$

Out[394]//TraditionalForm=
$$x^{-1} + x^{-2} + x^{-3} + x^{4}$$

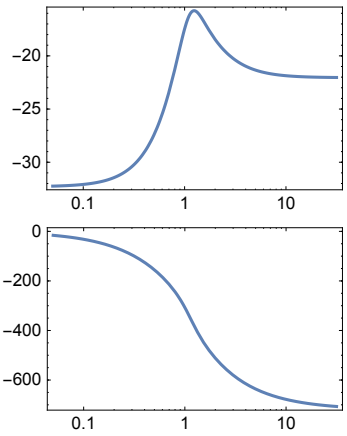
Out[395]= z^{-1}

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

test

In[396]:=
```
Clear[z, hz, alpha, cutOffFrequency, wp]
hz[z_] := (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_] :=
  (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; (*rad per second*)
wp = .6 Pi; (*rad per second*)
alpha = -Cos[(wp+cutOffFrequency)/2] / Cos[(wp-cutOffFrequency)/2];

foundHz[z_] := hz[(-(z^-1 + alpha)/(1 + alpha z^-1))];
BodePlot[foundHz[z]]
```

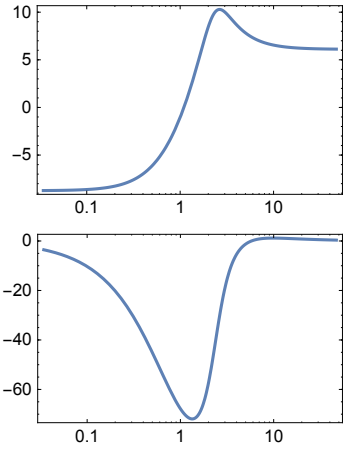
Out[403]=



test2

In[404]:=
```
Clear[z, hz, x, expectedHz, foundHz]
hz[z_] := 0.175 z^-1 / (1 - 1.318 z^-1 + 0.493 z^-2);
expectedHz[z_] := (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 = -Cos[(wp+cutOffFrequency)/2] / Cos[(wp-cutOffFrequency)/2];

foundHz[z_] := hz[(-(z^-1 + alpha)/(1 + alpha z^-1))];

BodePlot[foundHz[z]]
```

Out[411]=

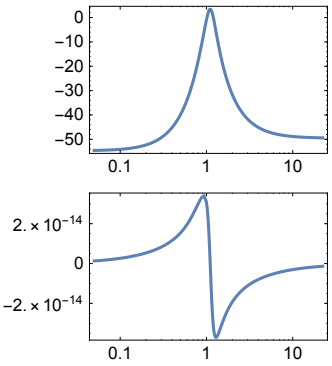


Function to convert low pass digital filter to bandpass digital filter using frequency transformation

test

In[412]:=
```
Clear[z, hz, alpha, cutOff, w1, w2, k]
hz[z_] := (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; (*rad per second*)
w1 = .4 Pi;(*rad per second*)
w2 = .6 Pi;(*rad per second*)
alpha = Cos[(w2+w1)/2] / Cos[(w2-w1)/2];
k = Cot[(w2 - w1)/2] Tan[cutOff/2];
foundHz[z_] := Module[{}, hz[-N[((z^-2 - (2 alpha k)/(k+1) z^-1 + (k-1)/(k+1)) / ((k-1)/(k+1) z^-2 - (2 alpha k)/(k+1) z^-1 + 1))]]];

BodePlot[foundHz[z]]
```

Out[420]=

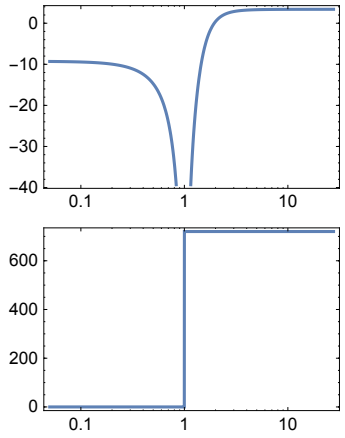


Function to convert low pass digital filter to bandstop digital filter by frequency transformation

test

In[421]:=
```
Clear[z, hz, alpha, cutOff, w1, w2, k]
hz[z_] :=
  (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 = Cos[(w2+w1)/2] / Cos[(w2-w1)/2];
k = Tan[(w2 - w1)/2] Tan[cutOff/2];

foundHz[z_] := hz[ (z^(-2) - (2 * alpha) / (k + 1) * z^(-1) + (1 - k) / (k + 1)) /
    ((1 - k) / (k + 1) * z^(-2) - 2 * alpha / (k + 1) * z^(-1) + 1)]

BodePlot[foundHz[z]]
```

Out[429]=



checkForSingularity

In[4]:=
```
ClearAll[c, a, b, x];
m = nma`checkForSingularity[
    (c - (a + b + 1) x) / (x (1 - x)), (-a b) / (x (1 - x)), {0, 1, Infinity}, x];
Grid[Join[{{"point", "limit x p(x)", "limit x^2 q(x)"}}, m], Frame → All]
```

Out[6]=

point	limit x p(x)	limit x^2 q(x)
0	c	0
1	1 + a + b - c	0
∞	1 - a - b	a b

In[7]:=
```
ClearAll[c, a, b, x];
m = nma`checkForSingularity[-2 x / (1 - x^2),
    (lam + 40 (1 - x^2) - (mu^2) / (1 - x^2)) / (1 - x^2), {-1, 1, Infinity}, x];
Grid[Join[{{"point", "limit x p(x)", "limit x^2 q(x)"}}, m], Frame → All]
```

Out[9]=

point	limit x p(x)	limit x^2 q(x)
-1	1	$-\frac{mu^2}{4}$
1	1	$-\frac{mu^2}{4}$
∞	0	∞