Parametric Model Reduction Tutorial

Evgenii B. Rudnyi, http://Evgenii.Rudnyi.Ru/

Introduction

This notebook allows you to repeat computations described in the paper

E. B. Rudnyi, L. H. Feng, M. Salleras, S. Marco, J. G. Korvink. Error Indicator to Automatically Generate Dynamic Compact Parametric Thermal Models. THERMINIC 2005, 11th International Workshop on Thermal Investigations of ICs and Systems, 27 - 30 September 2005, Belgirate, Lake Maggiore, Italy, p. 139 - 145. Preprint is at http://www.imtek.uni-freiburg.de/simulation/mor4ansys/pdf/rudnyi05therminic.pdf

The goal was to preserve three film coefficients for a simple thermal problem during model reduction process by means of multivariate expansion. The engineering problem is described in

E. B. Rudnyi and J. G. Korvink. Boundary Condition Independent Thermal Model. In: Benner, P., Mehrmann, V., Sorensen, D. (eds) Dimension Reduction of Large-Scale Systems, Lecture Notes in Computational Science and Engineering (LNCSE). Springer-Verlag, Berlin/Heidelberg, Germany, v. 45, p. 345-348, 2005. Preprint is at http://www.imtek.uni-freiburg.de/simulation/benchmark/

The system matrices are available from Oberwolfach Benchmark Collection (for convenience they are included in the distribution along with this notebook).

The original dynamic model after the discretization by the finite element method is as follows

E dx/dt + (K + ht Kt + hs Ks + hb Kb) x = Bu

The goal is to find a low-dimensional model in the similar form that approximates the dynamic properties of the original system in a wide range of film coefficients, ht, hs, hb.

It is assumed that you browse the papers above before starting the tutorial.

From a computational viewpoint, several Krylov subspaces are computed and then merged to form a projection basis. Several points in the parameter space of the transfer function are employed to choose an optimal dimension for each Krylov subspace.

Setting up

Make the directory with system matrices current. You may need to change the next expression.

```
SetDirectory["."]
```

Load the functions. They are included for convenience but the latest versions as well as documentation are available from my site at http://evgenii.rudnyi.ru/soft/Post4MOR.tar.gz. The content of the archive is also available online at http://evgenii.rudnyi.ru/soft/Post4MOR/.

```
<< Post4MOR.m
```

```
<< ModelReduction.m
```

The first set of functions that is in the file Post4MOR.m is well documented. The goal of the second set was to add new functionality with the minimal development cost. As a result, my design was to reuse functions from Post4MOR.m as much as possible. Documentation for ModelReduction.m is in the beginning of the file.

Turn off some especially annoying *Mathematica* warning messages.

```
In[4]:= Off[General::"spell"];
        Off[General::"spell1"];
```

Reading in and Preparing the Original System

The model in the benchmark was written as

E dx/dt = (A - hbottom Abottom -htop Atop -hside Aside)x + Bu

I convert it to the form used in the paper

E dx/dt + (K + ht Kt + hs Ks + hb Kb) x = Bu
K = -A
Kt = Atop
Ks = Aside
Kb = Abottom

Also, I leave only one output to reduce the number of plots. However, the procedure will work for many outputs as well.

```
In[6]:= matE = Import["T2DAL_BCI.E", "MTX"];
        matK = -Import["T2DAL_BCI.A", "MTX"];
        matB = Import["T2DAL_BCI.B", "MTX"];
        matKt = Import["T2DAL_BCI.Atop", "MTX"];
        matKs = Import["T2DAL_BCI.Aside", "MTX"];
        matKb = Import["T2DAL_BCI.Abottom", "MTX"];
        matC = Take[Import["T2DAL_BCI.C", "MTX"], 1, Length[matK]];
```

Function MakeParametricSystem takes as an input a list of system matrices but we also have to define three functions to convert this list (mat) to three system matrices of DynamicSystem for a given list of parameters (par).

```
In[13]:= getM = Function[{mat, par}, Null];
         getE = Function[{mat, par}, mat[[1]]];
         getK = Function[{mat, par},
            mat[[2]] + par[[1]] * mat[[3]] + par[[2]] * mat[[4]] + par[[3]] * mat[[5]]];

In[16]:= psys = MakeParametricSystem[{matE, matK, matKt, matKs, matKb},
            matB, matC, {"heater"}, {getM, getE, getK}];
```

For better readability of the code below, I define several additional function to access system matrices from Parametric-System.

```
In[17]:= MatrixE[sys_ParametricSystem] := Matrix[sys, 1]
         MatrixK[sys_ParametricSystem] := Matrix[sys, 2]
         MatrixKt[sys_ParametricSystem] := Matrix[sys, 3]
         MatrixKs[sys_ParametricSystem] := Matrix[sys, 4]
         MatrixKb[sys_ParametricSystem] := Matrix[sys, 5]
```

Computing Transfer Function for Several Points in the Parameter Space

The question how to choose the points to check the local error is open. In the paper, the points along diagonals in the parameter space were used with an assumption that provided the approximation error is small for these points it is also small for internal points.

We compute

H{s = 100, ht = 10, hb = 10, hs = 10}
H{s = 100, ht = 10^6, hb = 10, hs = 10}
H{s = 100, ht = 10^6, hb = 10^6, hs = 10}
H{s = 100, ht = 10^6, hb = 10, hs = 10^6}

YSeries returns a matrix with the number of rows equal to the number of outputs and the number of columns equal to the number of simulation points (frequencies). As *Mathematica* stores matrices by rows and we need a column, I use Transpose. First takes the first row from the matrix.

```
In[22]:= maxfreq = 100;
         p1 = {10, 10, 10};
         exact1 = First[Transpose[YSeries[
              HarmonicSolution[{maxfreq}, MakeDynamicSystem[psys, p1]]
             ]]];
         p2 = {1*^6, 10, 10};
         exact2 = First[Transpose[YSeries[
              HarmonicSolution[{maxfreq}, MakeDynamicSystem[psys, p2]]
             ]]];
         p3 = {1*^6, 1*^6, 10};
         exact3 = First[Transpose[YSeries[
              HarmonicSolution[{maxfreq}, MakeDynamicSystem[psys, p3]]
             ]]];
         p4 = {1*^6, 1*^6, 1*^6};
         exact4 = First[Transpose[YSeries[
              HarmonicSolution[{maxfreq}, MakeDynamicSystem[psys, p4]]
             ]]];
```

It could be good to take more points but in the general case, the computations with the original system are quite expensive.

Preparing Common Things

An expansion point for the Laplace variable is zero, as this allows us to preserve the stationary solution. However, the expansion point for each parameter is 10. As a result, it is necessary to convert MatrixK to a new matrix.

Transformation to hb0 = 10, hs0 = 10, hb0 = 10.

```
In[31]:= matKnew = MatrixK[psys] + 10. * (MatrixKt[psys] + MatrixKs[psys] + MatrixKb[psys])

Out[31]= SparseArray[<37465>, {4257, 4257}]
```

Note that the dimension of the original model is equal to 4257.

Matrix is factorized only once. Later on this factor is used in order to quickly solve a system of equations matKnew x = b.

```
In[32]:= invKnew = LinearSolve[matKnew, Method -> "Cholesky"];
```

The starting vector will be the same for all Krylov subspaces.

```
In[33]:= start = invKnew[MatrixB[psys].{1}];
```

I multiply the input matrix by {1} in order to convert it from a matrix to a vector. In *Mathematica* a matrix consisting from a single column is different from a vector: a matrix 3x1 is {{1}, {2}, {3}} and a vector is {1, 2, 3}.

We will make a similar plots several times and below there are options to make it look nicer.

```
In[34]:= plotoptions = {SymbolShape →
             {PlotSymbol[Triangle, Filled → False], PlotSymbol[Box, Filled → False],
              PlotSymbol[Diamond, Filled → False], PlotSymbol[Star, Filled → False]},
           SymbolStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0], RGBColor[0, 0, 1],
             RGBColor[0, 1, 1]}, FrameLabel → {"Model Dimension", ""},
           PlotLegend → {"10,10,10", "1e6,10,10", "1e6,1e6,10", "1e6,1e6,1e6"},
           LegendBorder → {}, LegendPosition → {-0.7, -0.4}, LegendSize → {0.8, 0.28},
           Evaluate[Imtek`Post4MOR`Private`defaultPlotOptions]};
```

Generating Krylov Subspace Along the Laplace variable

The function to compute the next vector is as follows

next = Knew^(-1) MatrixE prev

and it is defined first. ArnoldiProcess uses this function and the starting vector defined above in order to compute the orthogonalized Krylov subspace

Krylov{Knew^(-1) MatrixE v, Knew^(-1) f}

The projection matrix is returned in the transposed form as *Mathematica* stores matrices by rows. The maximum dimension 30 is taken based on previous experience.

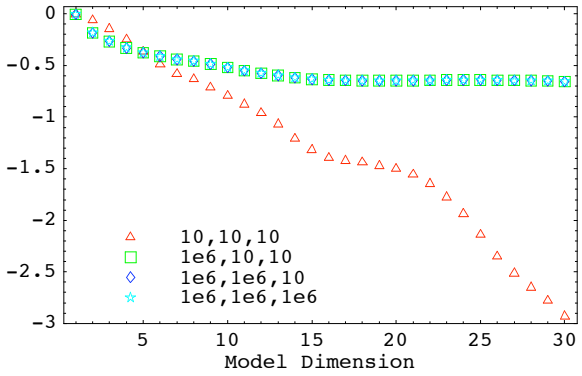
```
In[35]:= fun = Function[{prev}, invKnew[MatrixE[psys].prev]];
         V = ArnoldiProcess[fun, start, 30];
         Dimensions[V]

Out[37]= {30, 4257}
```

Now we project a parametric system onto the basis V

```
In[38]:= pred = ProjectSystem[psys, V];
```

to make a reduced parametric system. Then we compute the relative error between the original system and the reduced system for the four different parameter sets defined above as a function of the system dimension. MakeDynamicSystem converts ParametricSystem to DynamicSystem for given set of parameters, FrequencyConvergence runs HarmonicSimulation for maxfreq for reduced systems with different dimensions from 1 to Length[MatrixK[red]]. Local error computes the relative error between results from reduced systems and the original system.

```
In[39]:= red1 = MakeDynamicSystem[pred, p1];
         er1 = LocalError[FrequencyConvergence[maxfreq, red1], exact1];
         red2 = MakeDynamicSystem[pred, p2];
         er2 = LocalError[FrequencyConvergence[maxfreq, red2], exact2];
         red3 = MakeDynamicSystem[pred, p3];
         er3 = LocalError[FrequencyConvergence[maxfreq, red3], exact3];
         red4 = MakeDynamicSystem[pred, p4];
         er4 = LocalError[FrequencyConvergence[maxfreq, red4], exact4];

In[47]:= MultipleListPlot[{er1, er2, er3, er4}, Evaluate[plotoptions]];
```



We see that the relative error for the set p1 decreases while the relative error for other sets (p2, p3, p4) stays high. This can be easily explained because we generate vectors for the Laplace variable and they do not approximate the system behavior well when film coefficients change considerably.

In the paper, the local approximation error of 1% was considered to be good enough and the dimension 28 was chosen as the optimal for this subspace.

```
In[48]:= V = Take[V, 28];
         Dimensions[V]

Out[49]= {28, 4257}
```

Generating Krylov Subspace Along the parameter ht (film coefficient at the top).

The function to compute the next vector now is as follows

next = Knew^(-1) MatrixKt prev

The maximum dimension 30 is taken rather arbitrary.

```
In[50]:= fun = Function[{prev}, invKnew[MatrixKt[psys].prev]];
         W = ArnoldiProcess[fun, start, 30];
         Dimensions[W]

Out[52]= {30, 4257}
```

We merge subspaces V and W

```
In[53]:= V = Orthogonalize[V, W];
         Dimensions[V]

        Vector is deflated: 1 7.51801×10^-16

Out[54]= {57, 4257}
```

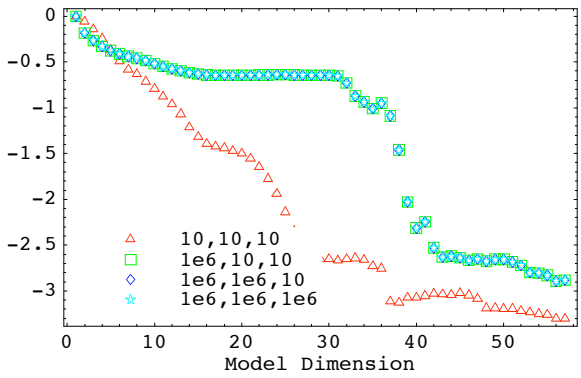
One vector is deflated as the starting vector was the same for both subspaces.

Now we project a parametric system onto the new basis V and compute relative errors again

```
In[55]:= pred = ProjectSystem[psys, V];
```

```
In[56]:= red1 = MakeDynamicSystem[pred, p1];
         er1 = LocalError[FrequencyConvergence[maxfreq, red1], exact1];
         red2 = MakeDynamicSystem[pred, p2];
         er2 = LocalError[FrequencyConvergence[maxfreq, red2], exact2];
         red3 = MakeDynamicSystem[pred, p3];
         er3 = LocalError[FrequencyConvergence[maxfreq, red3], exact3];
         red4 = MakeDynamicSystem[pred, p4];
         er4 = LocalError[FrequencyConvergence[maxfreq, red4], exact4];
```

```
In[64]:= MultipleListPlot[{er1, er2, er3, er4}, Evaluate[plotoptions]];
```



We see that now the local error defined for the film coefficient ht goes down. Surprisingly, the local error for other two film coefficients goes down simultaneously. This can be explained that in our case study the film coefficient at the top plays the major role.

Based on the convergence plot, we take the dimension of the reduced system of 41.

```
In[65]:= V = Take[V, 41];
         Dimensions[V]

Out[66]= {41, 4257}
```

Generating Krylov Subspace Along the parameter hs (film coefficient at the side).

The function to compute the next vector now is as follows

next = Knew^(-1) MatrixKs prev

The maximum dimension 30 is taken rather arbitrary.

```
In[67]:= fun = Function[{prev}, invKnew[MatrixKs[psys].prev]];
         W = ArnoldiProcess[fun, start, 30];
         Dimensions[W]

Out[69]= {30, 4257}
```

We merge subspaces V and W

```
In[70]:= V = Orthogonalize[V, W];
         Dimensions[V]

         Vector is deflated: 1 7.51769×10^-16

Out[71]= {70, 4257}
```

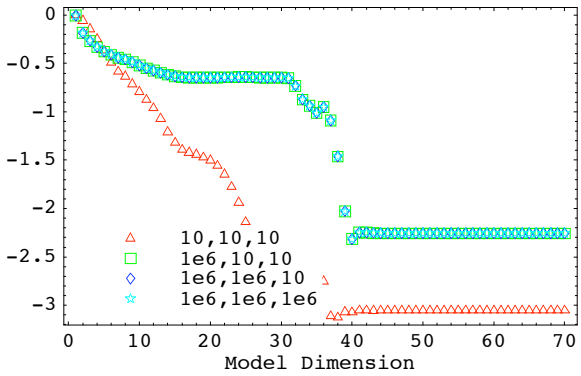
One vector is deflated as the starting vector was the same for both subspaces.

Now we project a parametric system onto the new basis V and compute relative errors again

```
In[72]:= pred = ProjectSystem[psys, V];
```

```
In[73]:= red1 = MakeDynamicSystem[pred, p1];
         er1 = LocalError[FrequencyConvergence[maxfreq, red1], exact1];
         red2 = MakeDynamicSystem[pred, p2];
         er2 = LocalError[FrequencyConvergence[maxfreq, red2], exact2];
         red3 = MakeDynamicSystem[pred, p3];
         er3 = LocalError[FrequencyConvergence[maxfreq, red3], exact3];
         red4 = MakeDynamicSystem[pred, p4];
         er4 = LocalError[FrequencyConvergence[maxfreq, red4], exact4];
```

```
In[81]:= MultipleListPlot[{er1, er2, er3, er4}, Evaluate[plotoptions]];
```



We can see that the new subspace does not reduce the local error and we leave the dimension of the reduced system of 41.

```
In[82]:= V = Take[V, 41];
         Dimensions[V]

Out[83]= {41, 4257}
```

The expansion along hb (the file coefficient at the bottom) produces the same result, that is, no further reduction in the local error. As a result, the basis of the dimension 41 is considered to be the optimal one. The optimal parametric reduced is system is

```
In[84]:= pred = ProjectSystem[psys, V];
```

Comparing the Reduced and Original systems

The procedure above is empirical by its nature. There were two main assumptions made:

1) One can ignore mixed moments.
2) The relative error at a few border points in the parameter space is enough to choose the dimension of the reduced model.

In our experience, this was working okay for the problem in question but it is unclear whether this will work for other systems. Unfortunately, mathematical proofs to estimate errors in the case of parametric model reduction are missing.

Below, an empirical check of the quality of the reduced system is performed.

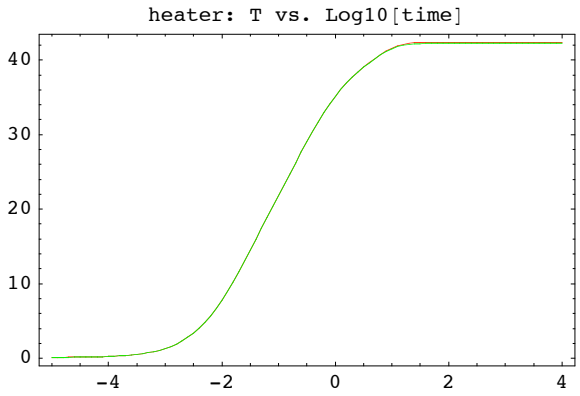
Function that compares the static, harmonic response and transient solution of the original and reduced systems. We compute relative error in per cent as follows Norm[full-red]/Norm[full]*100 for each output and then the maximum values is taken to the final table.

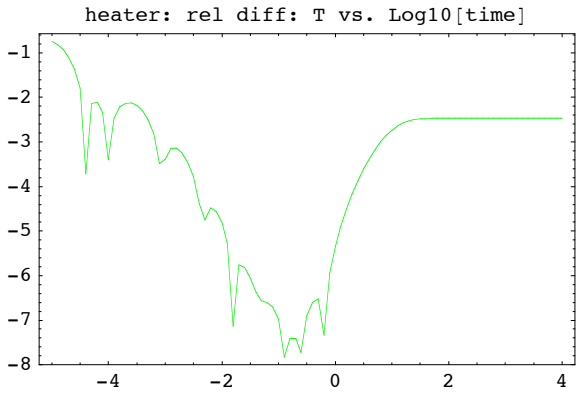
```
In[85]:= error[r2_SimulationResult, r1_SimulationResult] :=
          MapThread[Norm[#1 - #2] / Norm[#2] &, {YSeries[r2], YSeries[r1]}]

         compareSystems[psys_ParametricSystem, pred_ParametricSystem, par_List] :=
          Module[{sys, red, r1, r2, relS, relT, relH,
            time = Table[10^i, {i, -5, 4, 0.1}], freq = Table[10^i, {i, -4, 3, 0.5}]},
           Print[par];
           sys = MakeDynamicSystem[psys, par];
           red = MakeDynamicSystem[pred, par];
           r1 = StationarySolution[red];
           r2 = StationarySolution[sys];
           relS = Abs[r1 - r2] / r2 * 100;
           Print["Relative Error in % for Static ", relS];
           r1 = AnsysTransientSolution[time, red];
           r2 = AnsysTransientSolution[time, sys];
           PlotResult[{r2, r1}, PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0]},
            FunctionX → Log10, CommonTitle -> "T vs. Log10[time]"];
           PlotResult[Difference[r2, r1, ErrorFunction → (Log10[Abs[#1 - #2] / Abs[#1]] &)],
            PlotStyle → {RGBColor[0, 1, 0]}, FunctionX → Log10,
            CommonTitle -> "rel diff: T vs. Log10[time]"];
           relT = error[r2, r1] * 100;
           Print["Relative Error in % for Transient ", relT];
           r1 = HarmonicSolution[freq, red]; r2 = HarmonicSolution[freq, sys];
           PlotResult[{r2, r1}, FunctionY → (Log10[Abs[#]] &),
            FunctionX → Log10, PlotStyle → {RGBColor[1, 0, 0], RGBColor[0, 1, 0]},
            CommonTitle -> "Log10[Abs[H]] vs. Log10[freq]"];
           PlotResult[Difference[r2, r1, ErrorFunction → (Log10[Abs[#1 - #2] / Abs[#1]] &)],
            FunctionX → Log10, PlotStyle → {RGBColor[0, 1, 0]}, CommonTitle ->
             "rel diff: Log10[Abs[H]] vs. Log10[freq]"]; relH = error[r2, r1] * 100;
           Print["Relative Error in % for Harmonic ", relH];
           Flatten[{par, Max[relS], Max[relT], Max[relH]}]]
```

Below there is a check the function above for some parameter values. It takes some time as it makes simulation for the original system as well. First plot contains two curves: a red line for the original system and a green line for the reduced system. As the difference is small, one can see red just a little bit. The second plot gives the relative difference between the two curves.

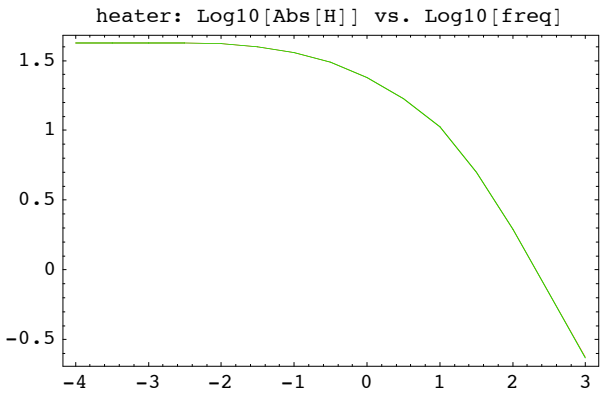
```
In[87]:= compareSystems[psys, pred, {1, 100, 1000}]

         {1, 100, 1000}
```

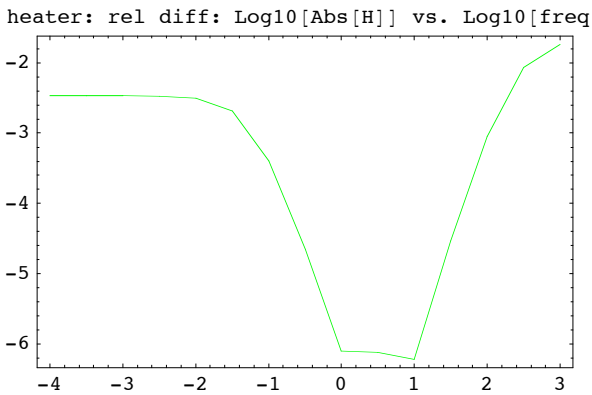
```
Relative Error in % for Static {0.337146}
```






```
Relative Error in % for Transient {0.272574}
```



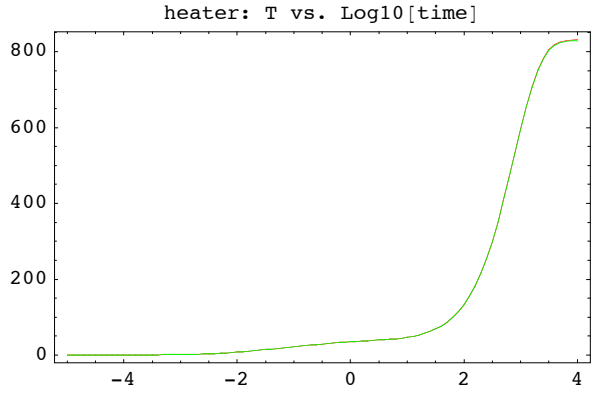


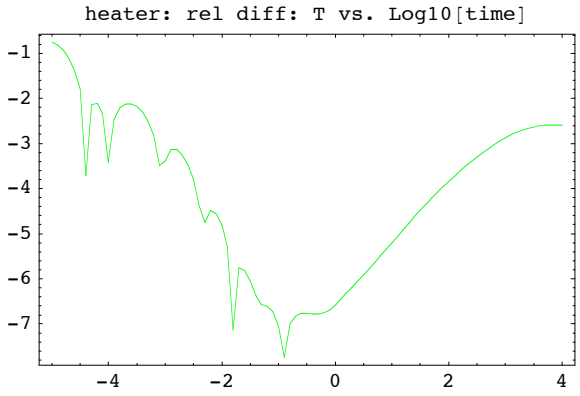

```
Relative Error in % for Harmonic {0.278135}

Out[87]= {1, 100, 1000, 0.337146, 0.272574, 0.278135}
```

Now we make a loop over different combinations of parameters. Computational time is equal to that above multiplied by Length[krange]^3.

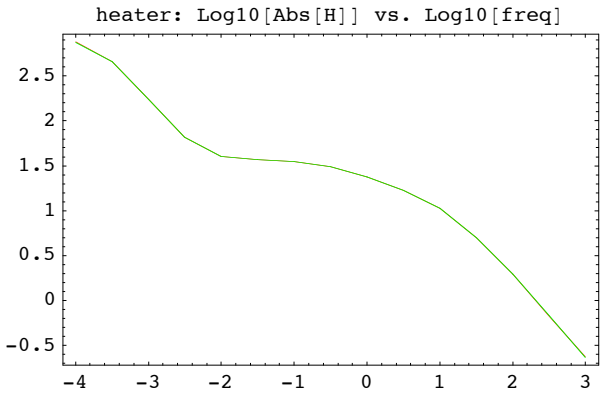
```
In[88]:= krange = {1, 10000};

In[89]:= table = Outer[compareSystems[psys, pred, {#1, #2, #3}] &, krange, krange, krange];
```

```
{1, 1, 1}

Relative Error in % for Static {0.259052}
```

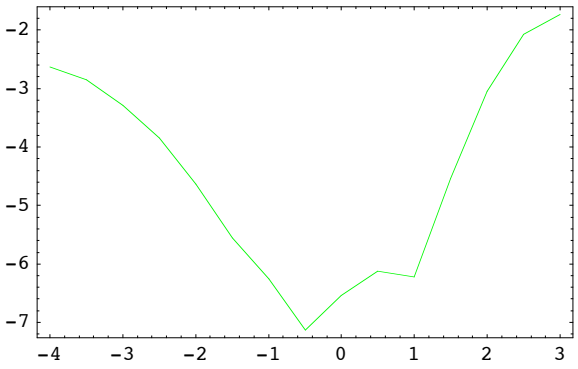





```
Relative Error in % for Transient {0.212959}
```



heater: rel diff: Log10[Abs[H]] vs. Log10[freq]

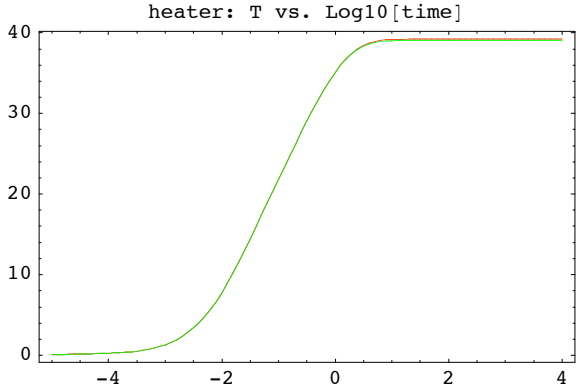


Relative Error in % for Harmonic {0.207698}

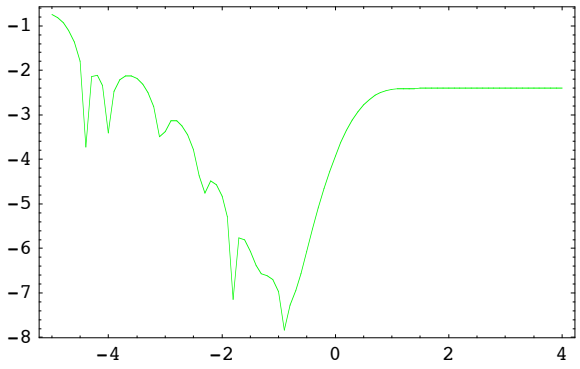
{1, 1, 10000}

Relative Error in % for Static {0.391372}

heater: T vs. Log10[time]

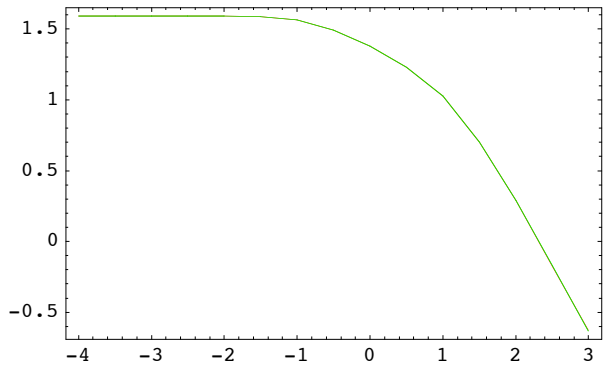


heater: rel diff: T vs. Log10[time]

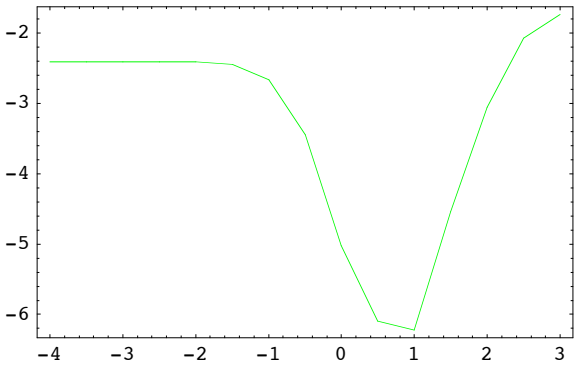


Relative Error in % for Transient {0.331023}

heater: Log10[Abs[H]] vs. Log10[freq]



heater: rel diff: Log10[Abs[H]] vs. Log10[freq]

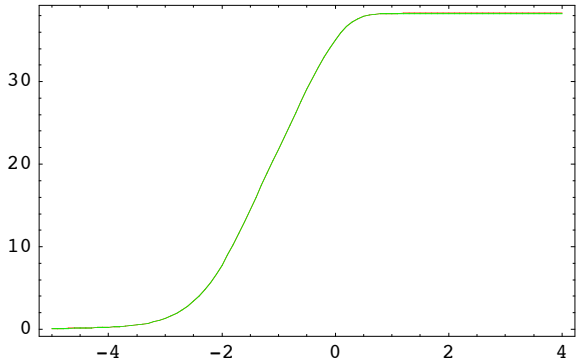


Relative Error in % for Harmonic {0.340043}

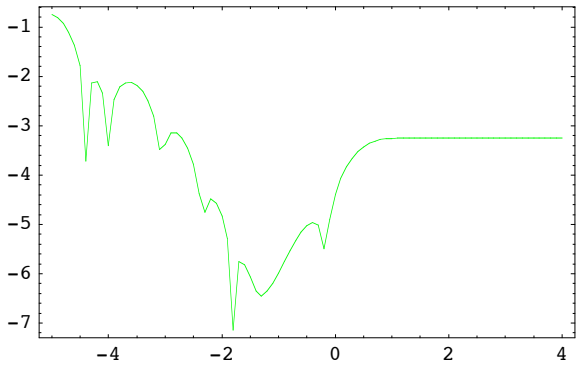
{1, 10000, 1}

Relative Error in % for Static {0.0560827}

heater: T vs. Log10[time]

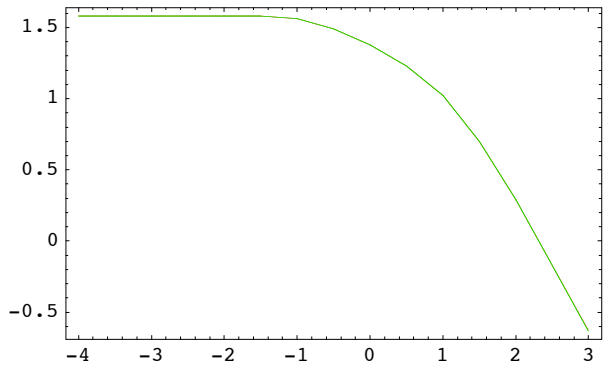


heater: rel diff: T vs. Log10[time]

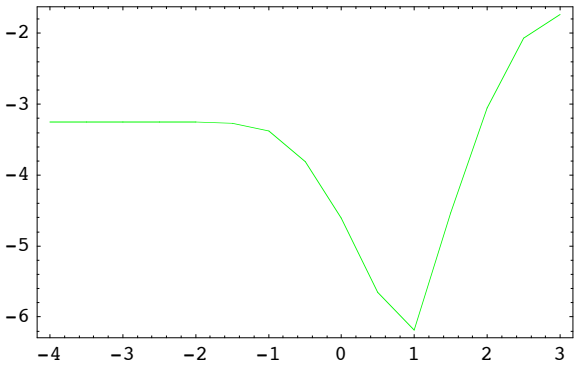


Relative Error in % for Transient {0.0496018}

heater: Log10[Abs[H]] vs. Log10[freq]



heater: rel diff: Log10[Abs[H]] vs. Log10[freq]

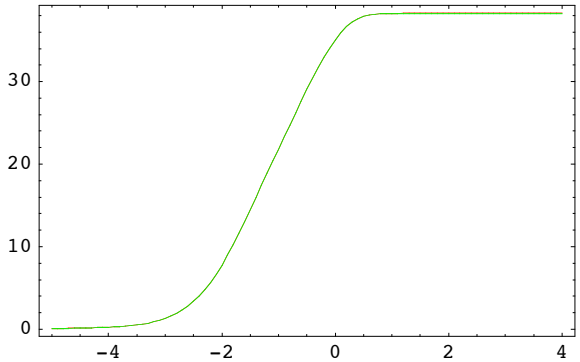


Relative Error in % for Harmonic {0.050184}

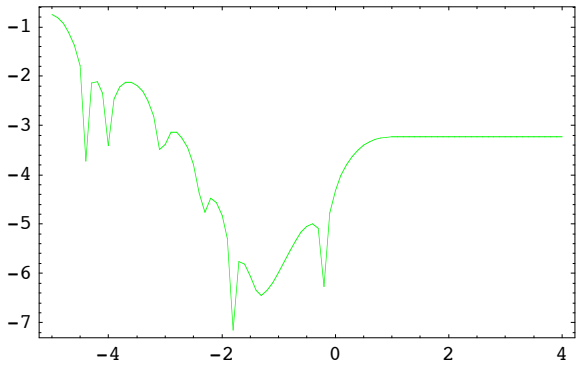
{1, 10000, 10000}

Relative Error in % for Static {0.0586923}

heater: T vs. Log10[time]

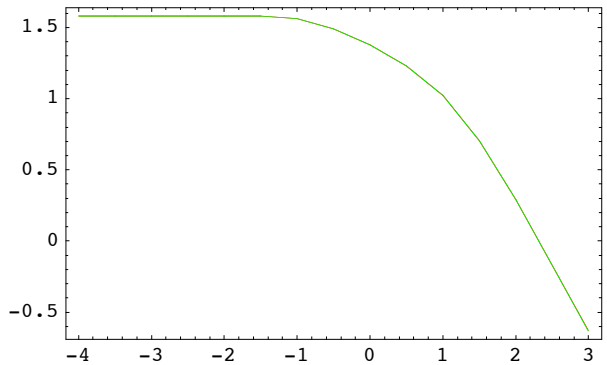


heater: rel diff: T vs. Log10[time]

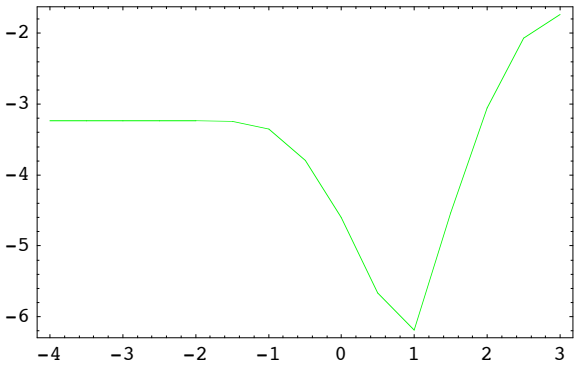


Relative Error in % for Transient {0.0518569}

heater: Log10[Abs[H]] vs. Log10[freq]



heater: rel diff: Log10[Abs[H]] vs. Log10[freq]

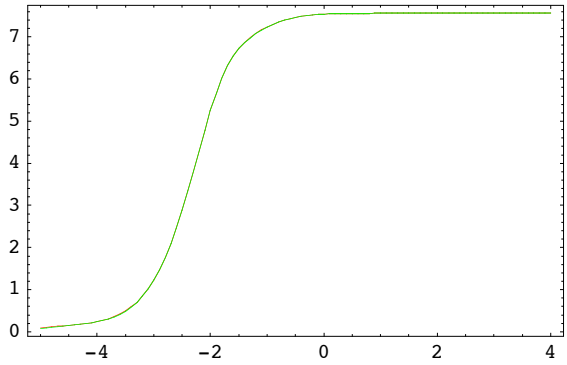


Relative Error in % for Harmonic {0.0525566}

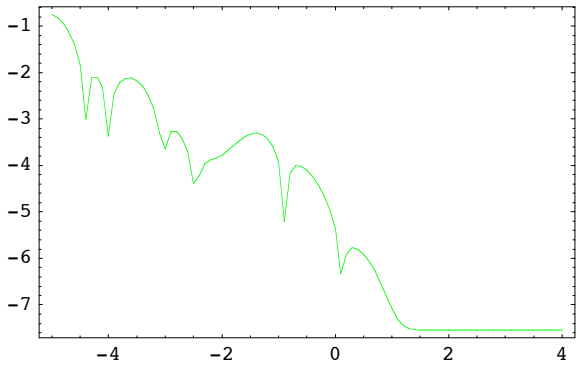
{10000, 1, 1}

Relative Error in % for Static $\{2.86708 \times 10^{-6}\}$

heater: T vs. Log10[time]

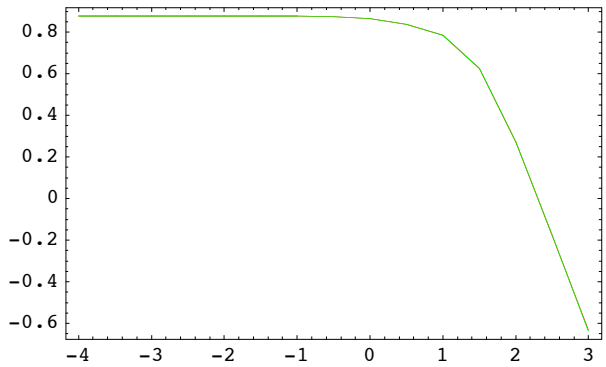


heater: rel diff: T vs. Log10[time]

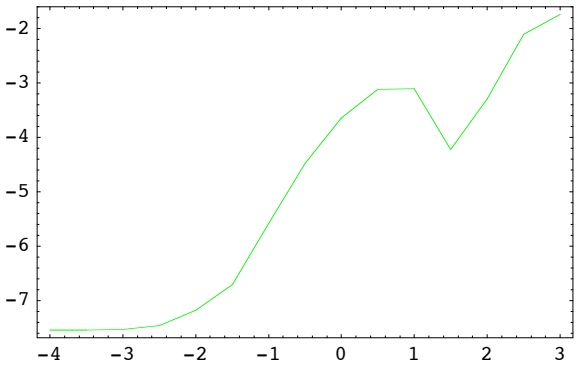


Relative Error in % for Transient {0.0540516}

heater: Log10[Abs[H]] vs. Log10[freq]



heater: rel diff: Log10[Abs[H]] vs. Log10[freq]

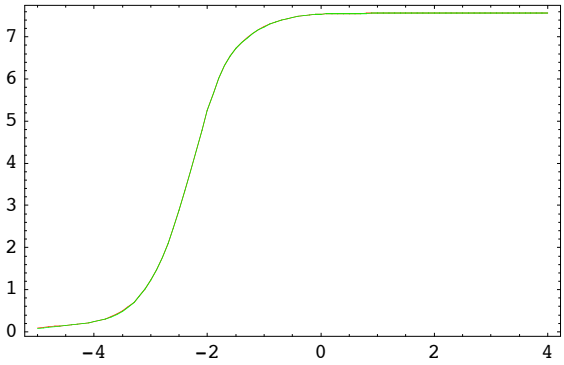


Relative Error in % for Harmonic {0.0403429}

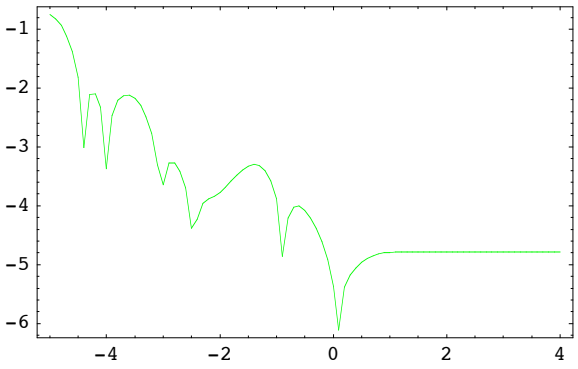
{10000, 1, 10000}

Relative Error in % for Static {0.00163634}

heater: T vs. Log10[time]

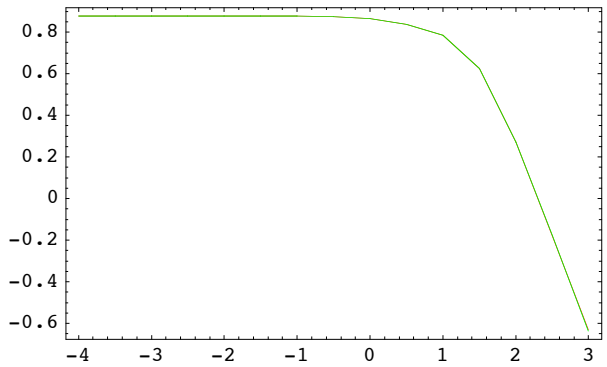


heater: rel diff: T vs. Log10[time]

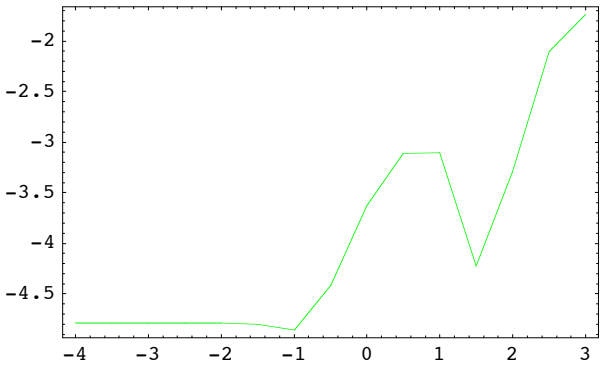


Relative Error in % for Transient {0.0541043}

heater: Log10[Abs[H]] vs. Log10[freq]



heater: rel diff: Log10[Abs[H]] vs. Log10[freq]

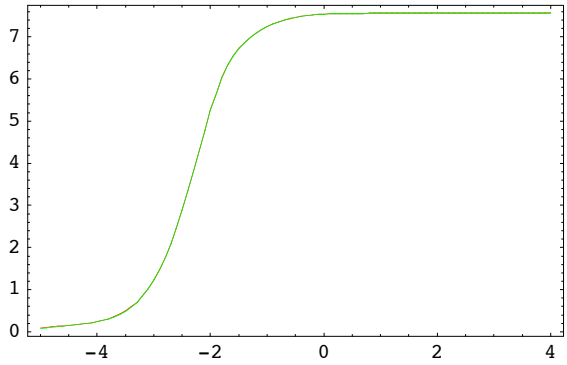


Relative Error in % for Harmonic {0.040447}

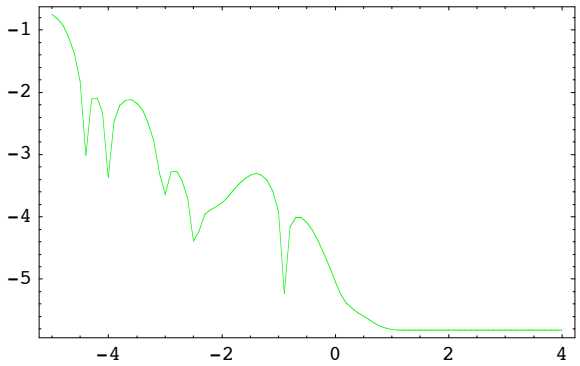
{10000, 10000, 1}

Relative Error in % for Static {0.000152372}

heater: T vs. Log10[time]

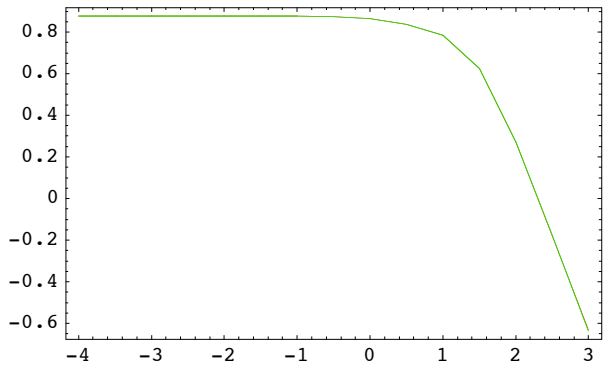


heater: rel diff: T vs. Log10[time]

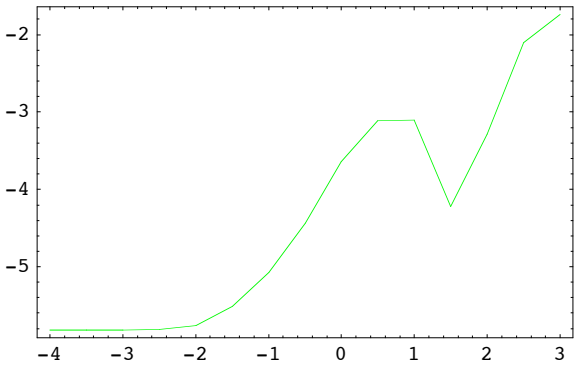


Relative Error in % for Transient {0.0540634}

heater: Log10[Abs[H]] vs. Log10[freq]



heater: rel diff: Log10[Abs[H]] vs. Log10[freq]

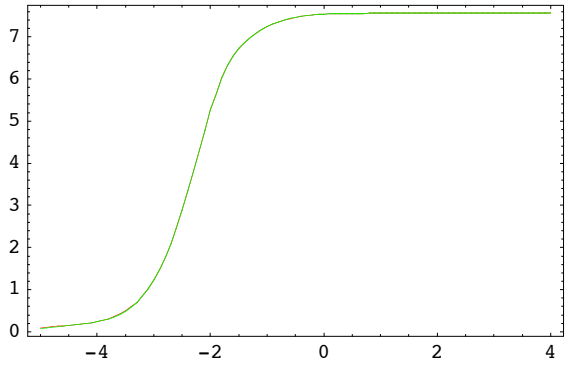


Relative Error in % for Harmonic {0.0403663}

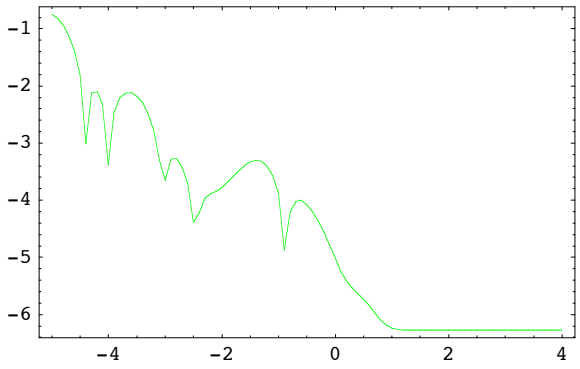
{10000, 10000, 10000}

Relative Error in % for Static {0.000053794}

heater: T vs. Log10[time]

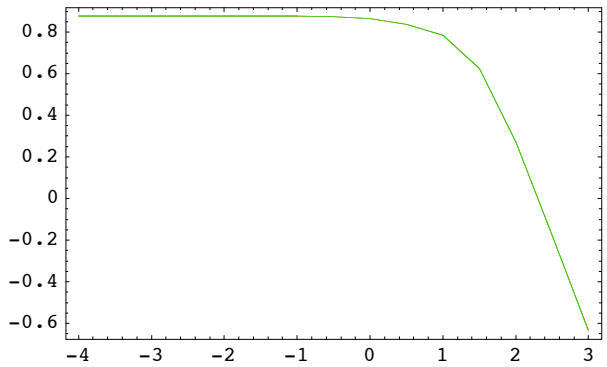


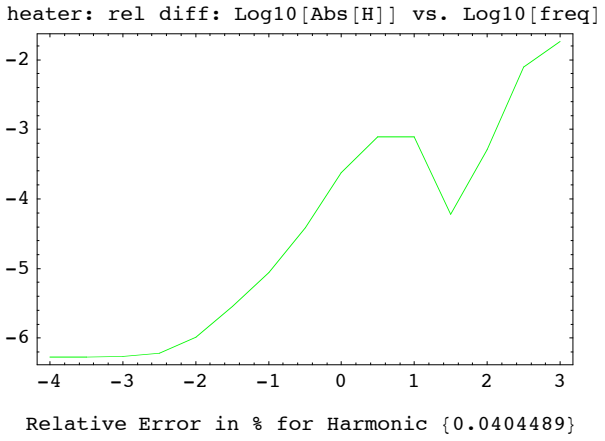
heater: rel diff: T vs. Log10[time]



Relative Error in % for Transient {0.0540968}

heater: Log10[Abs[H]] vs. Log10[freq]





In[94]:=
```
TableForm[Flatten[table, 2]]
```

Out[94]//TableForm=

1	1	1	0.259052	0.212959	0.207698
1	1	10000	0.391372	0.331023	0.340043
1	10000	1	0.0560827	0.0496018	0.050184
1	10000	10000	0.0586923	0.0518569	0.0525566
10000	1	1	2.86708×10^{-6}	0.0540516	0.0403429
10000	1	10000	0.00163634	0.0541043	0.040447
10000	10000	1	0.000152372	0.0540634	0.0403663
10000	10000	10000	0.000053794	0.0540968	0.0404489

We can see that the low-dimensional system of dimension 41 does approximate the behavior of the original system of dimension 4257 quite well.

Bringing *Mathematica* Properties into the Default State

In[91]:=
```
On[General::"spell"];
On[General::"spell1"];
```