Suspension Modeling in *Mathematica*™

e2e Group Meeting

10 March 2005

Motivation

- Wanted an AdvLIGO SUS design model to go beyond the Matlab model of Torrie, Strain et al.
- Desired features:
  - Full 3D with provision for asymmetries
  - Proper blade model
  - Wire bending elasticity
  - Arbitrary damping and consequent thermal noise
  - Export to other environments such as Matlab/Simulink and E2E.
- Mathematica code originally developed for modeling the X-pendulum was available -> reuse and extend.

Toolkit Features

- The toolkit is a Mathematica “package”, PendUtil.nb, for specifying different configurations (e.g., quad, triple etc) in a (relatively) user-friendly way
- Supported features:
  - » 6-DOF rigid bodies for masses (no internal modes)
  - » Springs described by an elasticity tensor and a vector of pre-load forces
  - » Massless wires (i.e., no violin modes) but detailed elasticity model from beam equation
  - » Arbitrary frequency-dependent damping on all sources of elasticity
  - » Symbolic up to the point of minimizing the potential to find the equilibrium position
  - » Calculates elasticity and mass matrices semi-numerically (symbolic partial derivatives of functions with mostly numeric coefficients)
  - » Eigenfrequencies and eigenmodes calculated numerically
  - » Reasonable runtime:
    - – 2 minutes for quad model with just wire longitudinal elasticity (adequate for most control theory purposes)
    - – 2 hours with wire bending elasticity (required for thermal noise estimates)
  - » Structured to make version control easy

Normal Mode Calculation (i)

- Express the potential energy of the system in terms of the coordinates:

$$E_P = E_P\left(x_1, \ldots x_n\right) = E_P\left(\mathbf{x}\right)$$

- Express the kinetic energy of the system in terms of the coordinates and coordinate velocities:

$$E_K = E_K\left(x_1, \ldots x_n, \dot{x}_1, \ldots \dot{x}_n\right)$$

- Minimize the potential energy to find the equilibrium values of the coordinates.

$$\mathbf{x}_{eq} = \left(x_{1(eq)}, \ldots x_{n(eq)}\right)^{\mathbf{T}}$$

LIGO

Normal Mode Calculation (ii)

- Create a matrix of second derivatives of the potential energy, a.k.a., the potential energy matrix or the stiffness matrix.

$$\mathbf{K}: K_{ij} = \left.\frac{\partial E_P}{\partial x_i \partial x_j}\right|_{\mathbf{x}=\mathbf{x}_{eq}} \qquad E_P = E_P\left(\mathbf{x}_{eq}\right) + \tfrac{1}{2}\left(\mathbf{x}-\mathbf{x}_{eq}\right)^{\mathbf{T}} \mathbf{K}\left(\mathbf{x}-\mathbf{x}_{eq}\right)$$

- Create a matrix of second derivatives w.r.t. velocity, a.k.a., the kinetic energy matrix or the mass matrix.

$$\mathbf{M}: M_{ij} = \left.\frac{\partial E_K}{\partial \dot{x}_i \partial \dot{x}_j}\right|_{\substack{\dot{\mathbf{x}}=0 \\ \mathbf{x}=\mathbf{x}_{eq}}} \qquad E_K = \tfrac{1}{2}\dot{\mathbf{x}}^{\mathbf{T}}\mathbf{M}\dot{\mathbf{x}}$$

Normal Mode Calculation (iii)

- Do a simultaneous diagonalization of the stiffness and mass matrices to obtain the eigenfrequencies and eigenmodes:

$$\mathbf{K}\mathbf{e}_i = \omega_i^2 \mathbf{M}\mathbf{e}_i \qquad \mathbf{x}_i(t) = \mathbf{x}_{eq} + \mathbf{e}_i e^{\omega_i t} \qquad f_i = \omega_i / 2\pi$$

- For a practical calculation potential matrix step needs to be considerably elaborated, partly for efficiency and partly to support additional calculations such as transfer functions and thermal noise estimates.

Other coordinates

- Also need to consider coordinates of structure - constant during normal mode motion but movable when injecting displacement inputs:

$$\mathbf{s} = (s_1, \ldots s_l)^{\mathrm{T}} \qquad \mathbf{s}_{nom} = (s_{1(nom)}, \ldots s_{l(nom)})^{\mathrm{T}}$$

- And “floats”, coordinates of things such as junctions between elastic elements in series - not independent of normal mode coordinates:

$$\mathbf{q} = (q_1, \ldots q_m)^{\mathrm{T}}$$

LIGO

Master Potential Matrix

- To work with all types of coordinates efficiently, define master potential matrix:

$$\mathbf{P} : E_P = E_P\left(\mathbf{x}_{eq}, \mathbf{q}_{eq}, \mathbf{s}_{nom}\right) + \tfrac{1}{2}\begin{pmatrix}\mathbf{x}^{\mathrm{T}} - \mathbf{x}^{\mathrm{T}}_{eq} & \mathbf{q}^{\mathrm{T}} - \mathbf{q}^{\mathrm{T}}_{eq} & \mathbf{s}^{\mathrm{T}} - \mathbf{s}^{\mathrm{T}}_{nom}\end{pmatrix}\mathbf{P}\begin{pmatrix}\mathbf{x} - \mathbf{x}_{eq} \\ \mathbf{q} - \mathbf{q}_{eq} \\ \mathbf{s} - \mathbf{s}_{nom}\end{pmatrix}$$

- It has a block structure with many useful submatrices:

$$\mathbf{P} = \begin{pmatrix}\mathbf{K} & \mathbf{C}_{XQ} & \mathbf{C}_{XS} \\ \mathbf{C}_{QX} & \mathbf{Q} & \mathbf{C}_{QS} \\ \mathbf{C}_{SX} & \mathbf{C}_{SQ} & \mathbf{S}\end{pmatrix} \qquad \mathbf{C}_{XQ} = \mathbf{C}^{\mathrm{T}}_{QX}$$

LIGO

Effective Potential and Coupling Matrices

- If there are any float coordinates, K submatrix of P is not appropriate to use in the normal mode analysis, since it assumes q=const whereas actually:

$$\mathbf{q} = \mathbf{q}_{eq} - \mathbf{Q}^{-1}\mathbf{C}_{QX}\left(\mathbf{x} - \mathbf{x}_{eq}\right)$$

- The effective potential matrix is

$$\mathbf{K}_{eff} = \mathbf{K} - \mathbf{C}_{XQ}\mathbf{Q}^{-1}\mathbf{C}_{QX}$$

- Similarly the effective coupling matrix converting displacement inputs of the structure to forces on the normal mode coordinates is

$$f_{xs} = \mathbf{C}_{XS(eff)}\left(\mathbf{s} - \mathbf{s}_{nom}\right) = \left(C_{XS} - \mathbf{C}_{XQ}\mathbf{Q}^{-1}\mathbf{C}_{QS}\right)\left(\mathbf{s} - \mathbf{s}_{nom}\right)$$

Damping

- Damping can be represented by a complex elastic modulus:

$$k \rightarrow k_0\left(\varepsilon'(\omega)+i\varepsilon''(\omega)\right)$$

- Strictly, the Kramers-Kronig relation applies:

$$\varepsilon'(\omega)-1=\frac{2}{\pi}PV\int_{-\infty}^{\infty}\frac{\varepsilon''(x)}{x-\omega}dx \qquad \varepsilon''(\omega)=-\frac{2}{\pi}PV\int_{-\infty}^{\infty}\frac{\varepsilon'(x)-1}{x-\omega}dx$$

- However often the variation in the real part can be ignored:

$$k \rightarrow k_0\left(1+i\phi(f)\right)$$

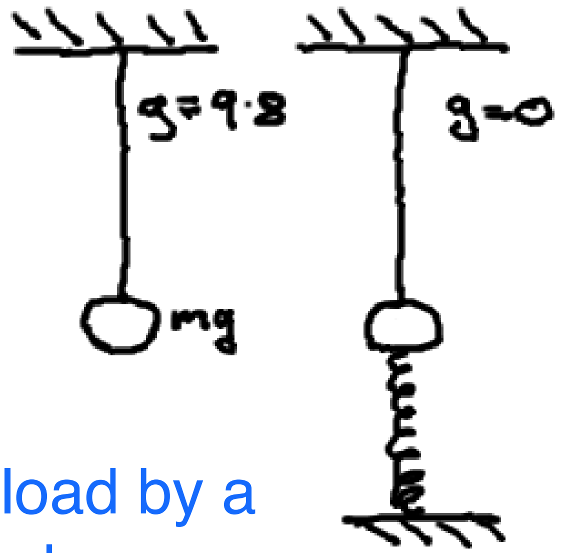
- Need to consider total potential as sum of terms, each with different damping:

$$\mathbf{P}=\sum \mathbf{P}_i\left(\varepsilon_i'(f)+i\varepsilon_i''(f)\right)$$

LIGO

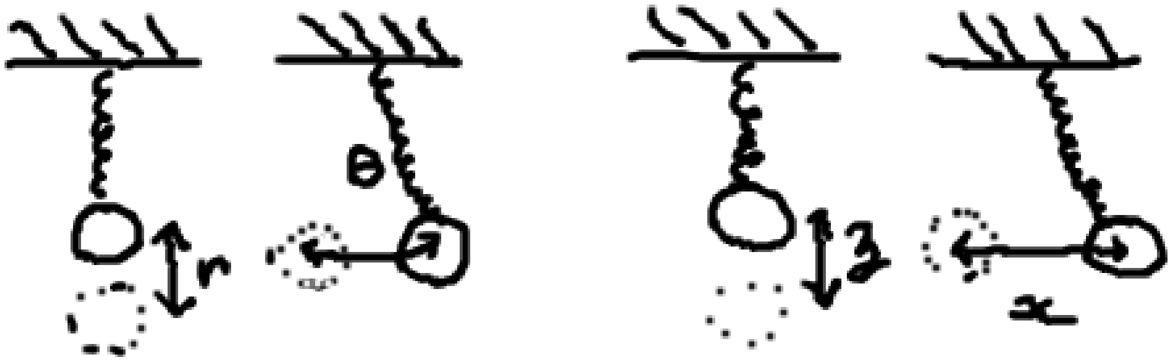
Dissipation Dilution and Pendulums (i)

- Two independent reasons why pendulums have low loss:
  - » Restoring force is gravitational
  - » Restoring force is sideways component of a tension
- Reason #2 would still apply if the tension were supplied by a second spring:
- Why? Because when a spring is used to create a restoring force by first generating a static force and then coupling that to the load by a variable mechanical advantage, the length change is only second order in amplitude.



Dissipation Dilution and Pendulums (ii)

- Why is it important to get this right? Because the normal mode formalism mixes up the two cases depending on the coordinates used and the stiffness of the wire:



- No stretch of spring for pendulum mode in polar coordinates vs. second order stretch in Cartesian coordinates.
- Solution: recompute potential matrix with tension zeroed out, then: $\mathbf{P} = \sum \left(\mathbf{P}_i \big|_{tension\_off} \left(\varepsilon_i'(f) + i\varepsilon_i''(f) \right) \right) + \sum \left(\mathbf{P}_i \big|_{tension\_on} - \mathbf{P}_i \big|_{tension\_off} \right)$

LIGO

Equations of Motion

- The net equation of motion is then,

$$\mathbf{K}_{eff}\left(\mathbf{x}-\mathbf{x}_{eq}\right)+\mathbf{M}\dot{\mathbf{x}}=\mathbf{f}_x+\mathbf{C}_{XS(eff)}\left(\mathbf{s}-\mathbf{s}_{nom}\right)$$

- Or in the frequency domain:

$$\mathbf{K}_{eff}\left(\mathbf{x}-\mathbf{x}_{eq}\right)-\left(2\pi f\right)^2\mathbf{M}\left(\mathbf{x}-\mathbf{x}_{eq}\right)=\mathbf{f}_x+\mathbf{C}_{XS(eff)}\left(\mathbf{s}-\mathbf{s}_{nom}\right)$$

- This can be solved for **x** for a sequence of different values of *f* to give force-input or displacement-input transfer functions as a function of frequency.
- Thermal noise is calculated in usual way from complex admittance.

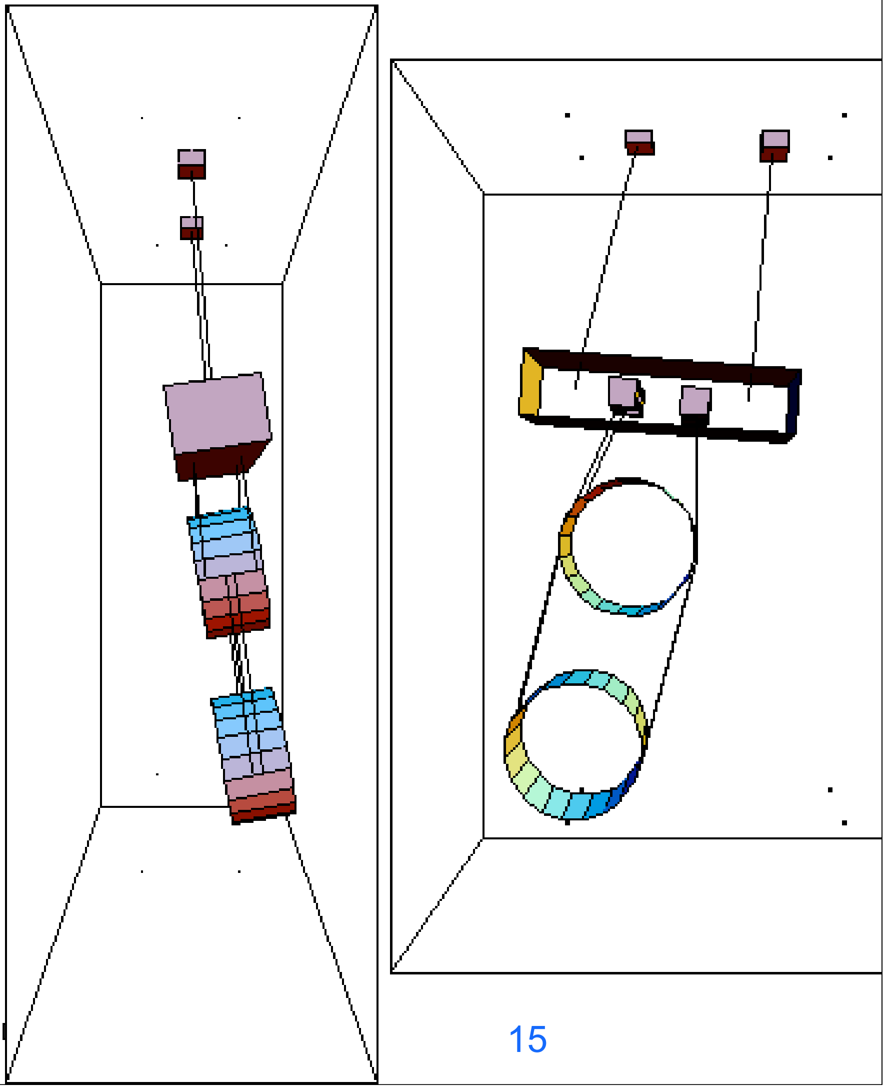
LIGO

Models

- Two major families of models have been defined:
  - » The triple models reflect a generic GEO-style pendulum with 3 masses, 6 blade springs and 10 wires.
  - » The quad models reflect a standard AdvLIGO quad pendulum, with 4 masses, 6 blade springs and 14 wires.
- Within each family there are three variants
  - » The “full” version, where the tips of the blade springs are modeled as 6-DOF rigid bodies attached to their bases by 6-DOF springs
  - » The “lite” version, where the tips of the blade springs are connected to their bases by geometrical constraints in 5 DOFs and elastically in 1 DOF.
  - » The xtra-lite version, where the tips of the blade springs are massless.
- The “xtra-lite” models are preferred for time-domain simulation because they have the smallest matrices and no high-frequency eigenmodes due to the blades.

Triple Pendulum Model

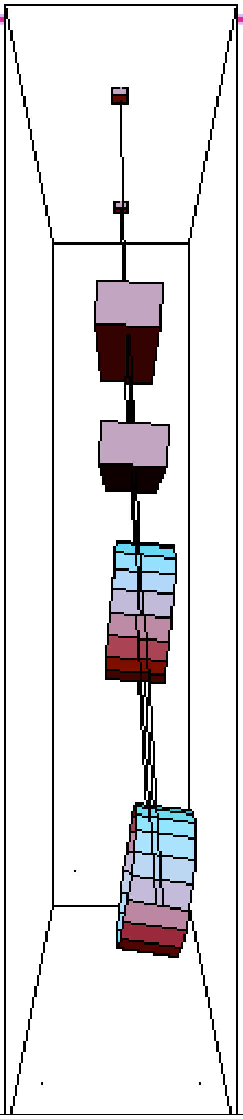
- 2 blade springs
- 2 wires
- “upper” mass
- 4 blade springs
- 4 wires
- “intermediate” mass
- 4 fibres
- optic

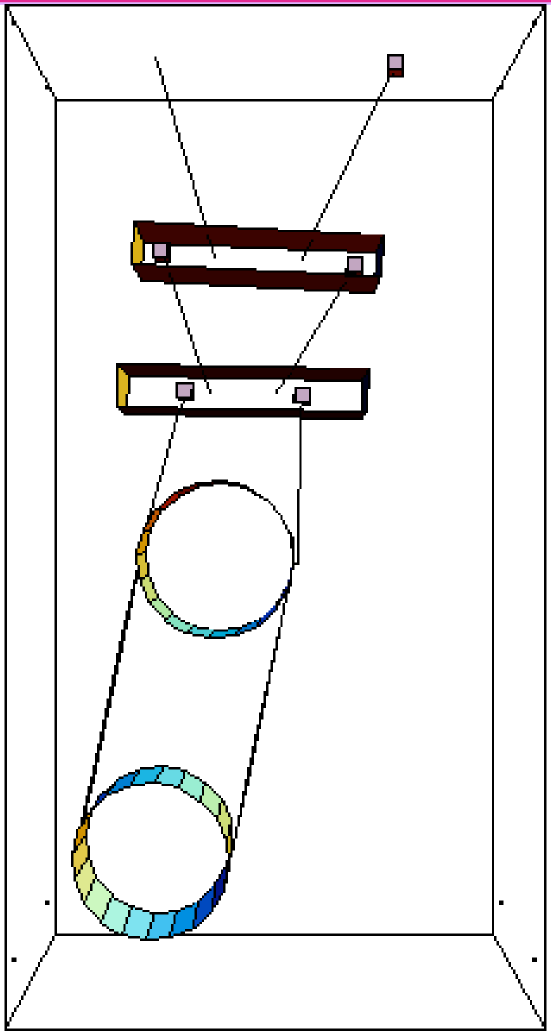


LIGO

Quad Pendulum

- 2 blade springs
- 2 wires
- “top” mass
- 2 blade springs
- 4 wires
- “upper” mass
- 2 blade springs
- 4 wires
- “intermediate” mass
- 4 fibres
- optic





LIGO

Defining a Model (i)

- Define the “variables” (cf. **x** in the theory - example from the xtra-lite triple):

```
• allvars = {
»     x1,y1,z1,yaw1,pitch1,roll1,
»     x2,y2,z2,yaw2,pitch2,roll2,
»     x3,y3,z3,yaw3,pitch3,roll3
• };
```

- Define the “floats” (cf. **q** in the theory):

```
»allfloats = {
      –qul,qur,qlf,qlb,qrf,qrb
»};
```

- Define the “parameters” (cf. **s** in the theory):

```
• allparams = {
»     x00, y00, z00, yaw00, pitch00, roll00
• };
```

LIGO

Defining a Model (ii)

- Define coordinate lists for rigid bodies of interest:

```
optic = {x3, y3, z3, yaw3, pitch3, roll3};
support = {x00, y00, z00, yaw00, pitch00, roll00};
```

- Define coordinate lists for points on rigid bodies

```
massUl={0,-n1,d0}; (* left wire attachment point on upper mass *)
```

- Define list of gravitational potential terms:

```
gravlist = {}; (* initialize list *)
AppendTo[gravlist, m3 g z3]; (* typical item *)
```

Defining a Model (iii)

- Define list of wires, each with the following format
- {
  - » *coordinate list defining first mass,*
  - » *attachment point for first mass (local coordinates),*
  - » *attachment vector for first mass,*
  - » *coordinate list defining second mass,*
  - » *attachment point for second mass (local coordinates),*
  - » *attachment vector for second mass,*
  - » *Young's modulus,*
  - » *unstretched length,*
  - » *longitudinal elasticity,*
  - » *vector defining principal axis 1,*
  - » *moment of area along principal axis 1,*
  - » *moment of area along principal axis 2,*
  - » *linear elasticity type,*
  - » *angular elasticity type,*
  - » *torsional elasticity type,*
  - » *shear modulus,*
  - » *cross sectional area for torsional calculations,*
  - » *torsional stiffness geometric factor*
- }

Defining a Model (iv)

- Define list of springs, each with following format:
- `{`
  - *coordinate list defining first mass,*
  - *attachment point for first mass (local coordinates),*
  - *attachment angles for first mass (yaw, pitch, roll),*
  - *coordinate list defining second mass,*
  - *attachment point for second mass (local coordinates),*
  - *attachment angles for second mass (yaw, pitch, roll),*
  - *damping type,*
  - *6x6 elasticity matrix,*
  - *1*6 pre-load force/torque vector*
- `}`
- Define kinetic energy
- `IM3 = {{I3x, 0, 0}, {0, I3y, 0}, {0, 0, I3z}}; (* typical MOI tensor)`
- `kinetic = (`
  - …
  - `+(1/2) m3 Plus@@(Dt[b2s[optic,COM],t]^2)`
  - `+(1/2) omegaB[yaw3, pitch3, roll3].IM3.omegaB[yaw3, pitch3, roll3]`
  - …
- `);`

LIGO

Defining a Model (v)

- Define default values of constants

```
defaultvalues = {
      g -> 9.81, (* value given numerically *)
      …
      m3 -> Pi*r3^2*t3, (* value given in terms of other constants *)
      …
      x00 -> 0, (* value for nominal position of structure *)
      y00 -> 0,
      z00 -> 0,
      …
      damping[imag,dampingtype] -> (phi&) (* value for frequency dependence of damping *)
      …
};
```

- Define starting point for finding equilibrium position:

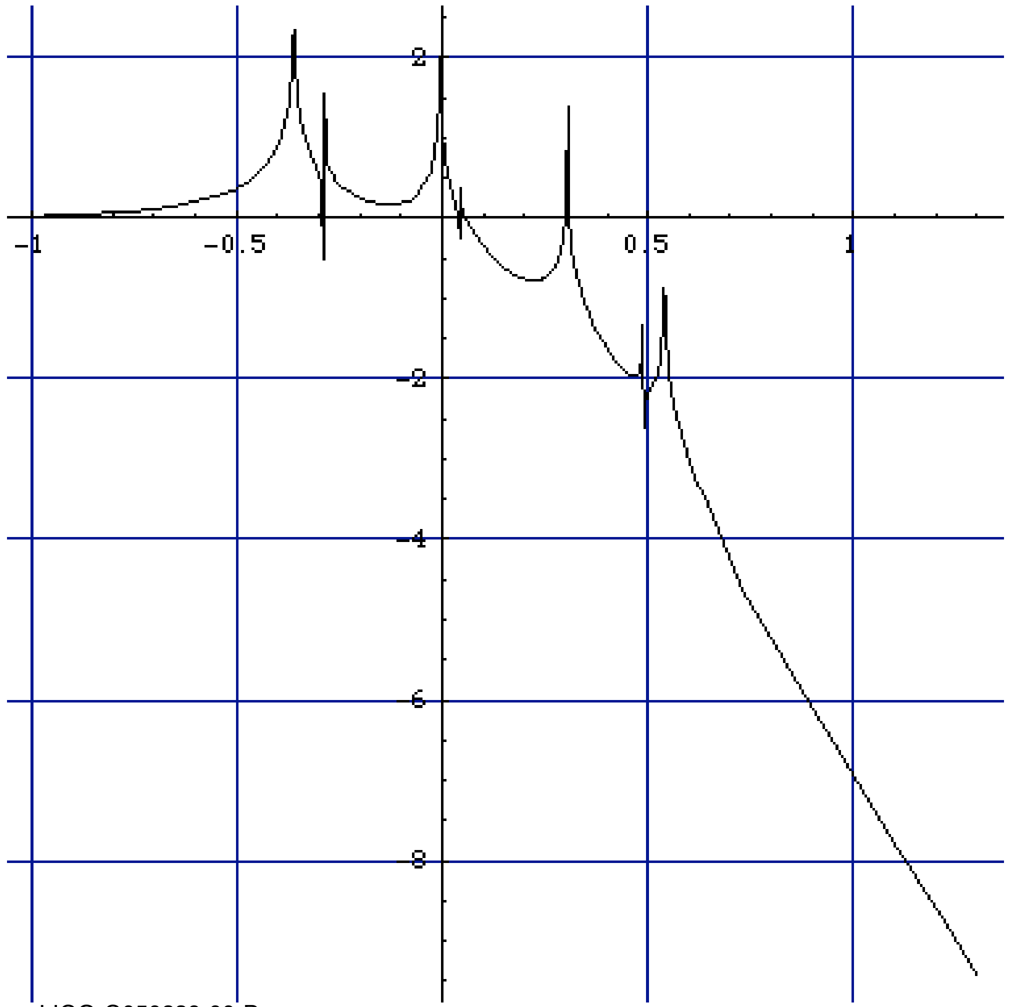
```
startpos = {
      x1 ->0,
      y1 ->0,
      …
};
```

Defining a Model (vi)

- Define model-specific utilities:
  - » A function to list eigenmodes in a table
  - » `pretty[eigenvector]`
  - » A function to plot eigenmode shapes
  - » `eigenplot[`*`eigenvector`*`, `*`amplitude`*`, {`*`viewpoint`*`}]`
  - » Vectors representing force and displacement inputs and displacement outputs of interest
  - » `structurerollinput = makeinputvector[roll00];`
  - » `opticxinput = makefinputvector[x3];`
  - » `opticx = makeoutputvector[x3];`
  - » Rotation matrices to put angle variables in a more easily interpretable basis:
  - » `e2ni;`

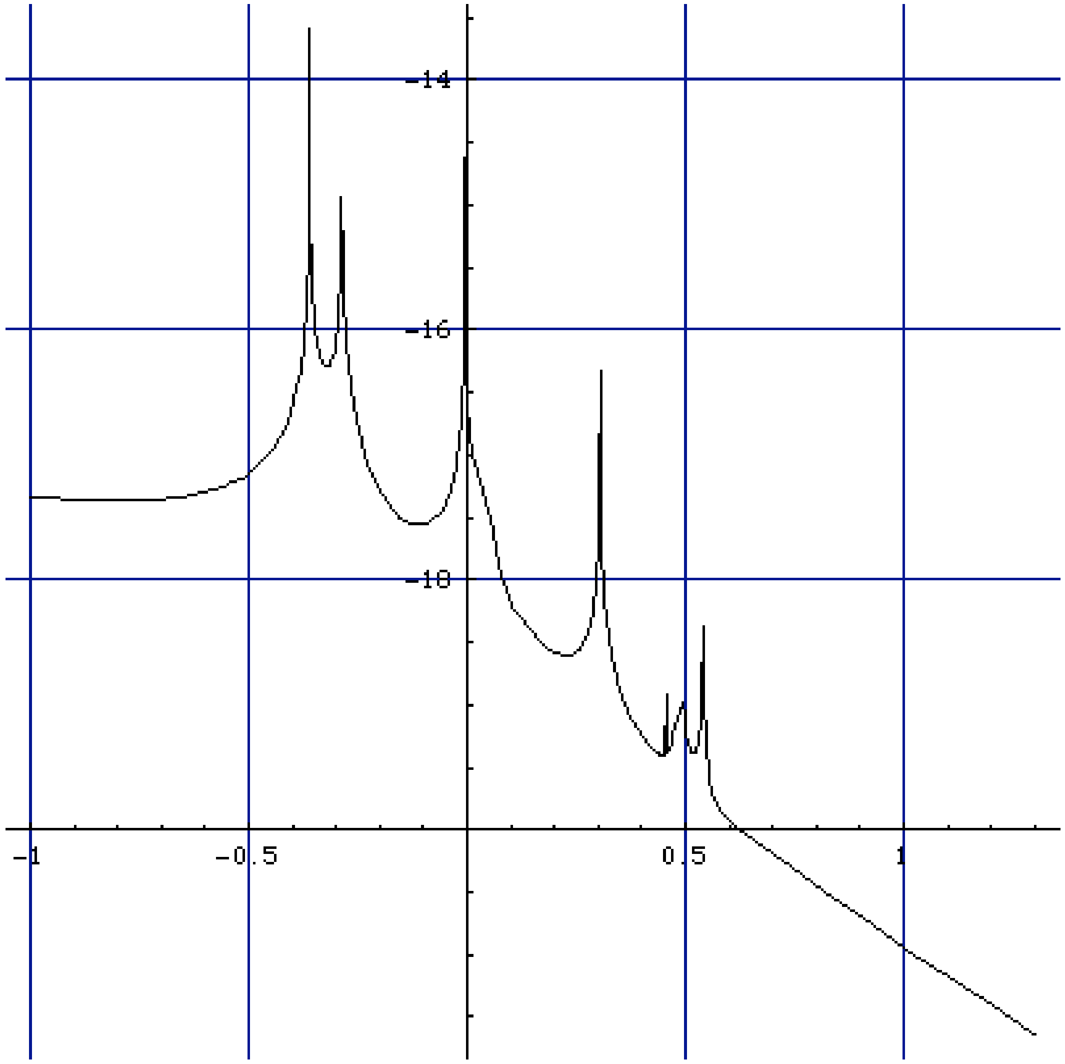
Sample Output (i)

- Transfer function from x displacement of support to x motion of optic (quad model, reference parameters of 20031114):

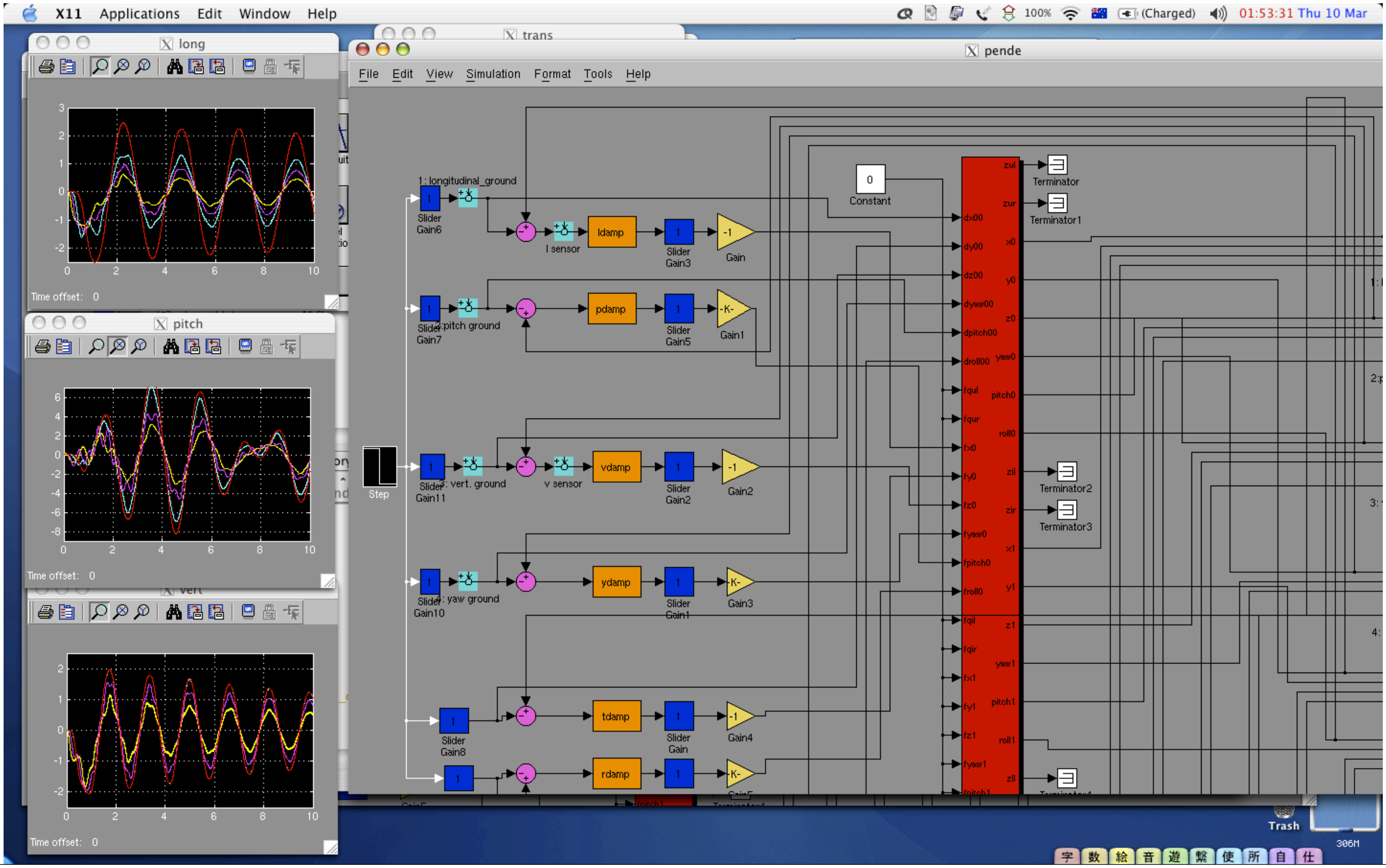


Sample Output (ii)

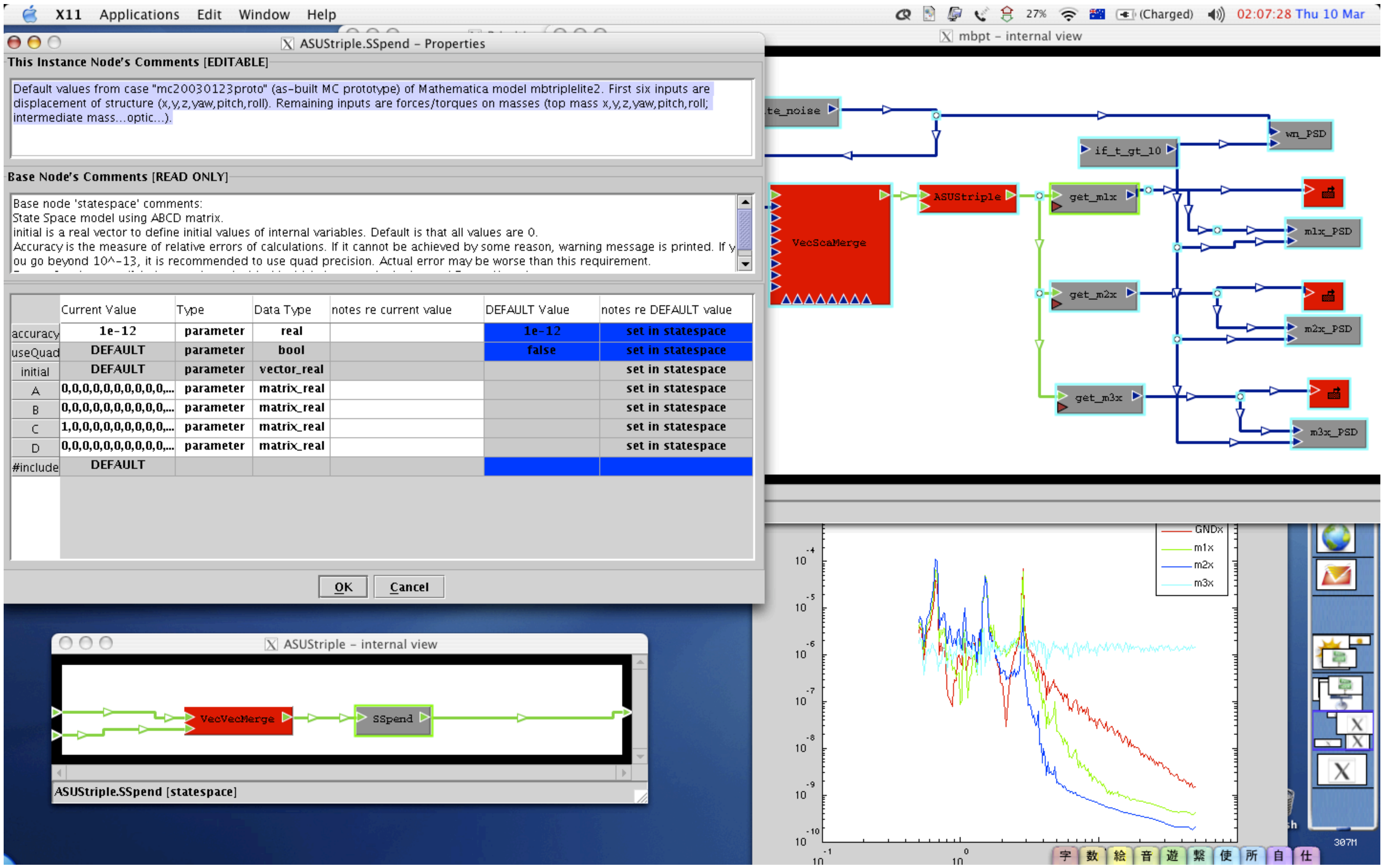
- Thermal noise in x motion of optic (quad model, reference parameters of 20031114):



Export to Matlab/Simulink



Export to E2E



ASUStriple.SSpend – Properties

This Instance Node's Comments [EDITABLE]

Default values from case "mc20030123proto" (as-built MC prototype) of Mathematica model mbtriplelite2. First six inputs are displacement of structure (x,y,z,yaw,pitch,roll). Remaining inputs are forces/torques on masses (top mass x,y,z,yaw,pitch,roll; intermediate mass...optic...).

Base Node's Comments [READ ONLY]

Base node 'statespace' comments:
State Space model using ABCD matrix.
initial is a real vector to define initial values of internal variables. Default is that all values are 0.
Accuracy is the measure of relative errors of calculations. If it cannot be achieved by some reason, warning message is printed. If you go beyond 10^-13, it is recommended to use quad precision. Actual error may be worse than this requirement.

	Current Value	Type	Data Type	notes re current value	DEFAULT Value	notes re DEFAULT value
accuracy	1e-12	parameter	real		1e-12	set in statespace
useQuad	DEFAULT	parameter	bool		false	set in statespace
initial	DEFAULT	parameter	vector_real			set in statespace
A	0,0,0,0,0,0,0,0,0,0,...	parameter	matrix_real			set in statespace
B	0,0,0,0,0,0,0,0,0,0,...	parameter	matrix_real			set in statespace
C	1,0,0,0,0,0,0,0,0,0,...	parameter	matrix_real			set in statespace
D	0,0,0,0,0,0,0,0,0,0,...	parameter	matrix_real			set in statespace
#include	DEFAULT					

Status

- Toolkit and quad and triple models defined and published:
  - http://www.ligo.caltech.edu/~mbarton/SUSmodels/indexMB.html
  - T020205-01
- Export of state-space matrices to E2E programmed.
- Simple E2E model using triple data is working.
- Need to do similar test model for quad data.
- Need to add local control systems.
- Need to define E2E blocks for all pendulum designs.
- Need to add global control (when finalized - ages off yet).
- Need to start on SEI.