1 Device display

Calling sequence

```
function cho_display(net, q);
```

Display the mechanical structure described by a netlist.

Inputs

`net` Netlist structure returned from calling the compiled netlist

`q` (Optional) - the displacement of the original structure, as returned by the static analysis routine or the transient analysis routine. If `q` is unspecified, the undisplaced structure will be displayed.

Example

```
net = beamgap;          % Load netlist for a simple beam-gap system
cho_display(net);       % Display the undisplaced structure
```

Caveats

There is no display of electrical components.

2 Viewing displacements

Calling sequence

```
function [dqcoord] = cho_dq_coord(dq, net, node, coord);
```

Extract the displacement corresponding to a particular coordinate from a displacement vector q (as output by `cho_dc`), or an array of displacement vectors (as output by `cho_ta`).

Note that a node “node” completely internal to an instance “foo” of some subnet would be named “node.foo”.

Inputs

`dq` : Displacement vector or array of displacement vectors.

`net` : Netlist structure returned from calling the compiled netlist

`node` : Name of node.

`coord` : Name of coordinate at indicated node.

Outputs

`dqcoord` : Value (or vector of values) from `dq` associated with the indicated coordinate.

Example

```
net = beamgap;                        % Load netlist
dq = cho_dc(net);                     % Perform static analysis
dy = cho_dq_view(Q, net, 'c', 'y');  % Get the y component at c
```

3 Static analysis

Calling sequence

```
function [q] = cho_dc(net, q0, is_sp)
```

Finds a solution of the equilibrium equations

$$Kx = F(x)$$

using a Newton-Raphson method.

Inputs

`net` : Netlist structure returned from calling the compiled netlist

`q0` : (Optional) Starting guess at the displacements from starting position for an equilibrium position. If no `q0` is provided, or if `q0 = []` the search will start at `q0` of 0.

`is_sp` : (Optional) If true, the code will use sparse solvers. The current default is true (use sparse solvers).

Outputs

`q` : The computed equilibrium state, expressed as displacements from the initial position.

Example

```
net = beamgap;         % Load netlist for a simple beam-gap system
dq = cho_dc(net);      % Perform static analysis
cho_display(net, dq); % Display the deflected structure
```

Caveats

The zero-finder currently used by SUGAR is very simple, and may fail to converge for some problems. If the function fails to converge after 40 iterations, it will exit and print a warning diagnostic:

```
Warning: DC solution finder did not converge after 40 iterations
```

In this case, the result `q` returned by the routine is likely to be useless. Failure to find equilibrium occurs in such cases as an electrostatically actuated gap operating near pull-in voltage.

When SUGAR finds an equilibrium point, it may not always be the equilibrium point desired. In the case of an electrostatically actuated gap, for example, there are two equilibria below pull-in voltage: one stable and one unstable. When the equilibria are close together, especially with respect to the distance from the starting point `q0`, the solver may move to an unstable equilibrium.

Good initial guesses for an equilibrium point can often be attained by finding the equilibrium point for a "nearby" problem. For example, in trying to find the equilibrium point for a electrostatically actuated gap operating near pull-in, a good initial guess `q0` would be the output of a static analysis for the same device at a lower voltage.

4 Modal analysis

Analysis routine

Calling sequence

```
function [freq, egv, q0] = cho_mode(net, nmodes, q0, find_dc);
```

Find the resonating frequencies and corresponding mode shapes (eigenvalues and eigenvectors) for the linearized system about an equilibrium point.

Inputs

`net` : Netlist structure returned from calling the compiled netlist.

`nmodes` : (Optional) If `nmodes > 0`, use sparse solvers to get nmodes modes If `nmodes == 0`, use the usual dense solver to get all the modes If `nmodes < 0`, solve with `eig(K \ M)` rather than `eig(M,K)`. This last option can potentially cause trouble (for instance, if K is singular), but it is faster.

`q0` : (Optional) Equilibrium operating point, or initial guess for a search for an equilibrium operating point. If not supplied, of if `q0 = []`, the routine will search for an equilibrium point near 0.

`find_dc` : (Optional) If true, search for an equilibrium point near the supplied q0.

Outputs

freq : Vector of resonating frequencies (eigenvalues)

egv : Array of corresponding mode shapes (eigenvectors)

q : Equilibrium point about which the system was linearized

Display routine

Calling sequence

```
function cho_modeshape(net, freq, egv, q0, s, num);
```

Display the shape of a resonating mode of the mechanical structure.

Inputs

`net` : Netlist structure returned from calling the compiled netlist.

`freq` : Vector of resonant frequencies from `cho_mode`.

`egv` : Array of mode shape vectors from `cho_mode`.

`q0` : Equilibrium point from `cho_mode`.

`s` : Scale factor. Eigenvectors from `cho_mode` are normalized to be unit length; for eigenvectors with significant components (within a few orders of magnitude of 1) in directions corresponding to components which normally move a few microns, a scale factor of 10^{-4} often makes the display of the mode more comprehensible.

`num` : Number of the mode to display. Modes are numbered in order of decreasing frequency.

Example

```
% Show the first (lowest-frequency) mode shape for the system,
% scaled by a factor of 0.1
net = beamgap;
[freq, egv, q] = cho_mode(net);
cho_modeshape(net, freq, egv, q0, 0.1, 1);
```

Caveats

While SUGAR will attempt to find an appropriate linearization point, it is not guaranteed to converge to one. See the caveats for static analysis.

The modal analysis routine neglects damping forces.

As noted above, using `cho_modeshape` with a too-large scaling factor often results in the displayed device being stretched to incomprehensible proportions.

Currently, trial-and-error guesses at an appropriate scale factor seem to work best.

5 Steady state analysis

Calling sequence

```
function find_ss(net, q0, in_node, in_var, out_node, out_var)
```

Make Bode plots of the frequency response of the linearized system about an equilibrium point `q0`.

Inputs

`net` : Netlist structure returned from calling the compiled netlist

`q0` : Equilibrium position for the system, as determined via the static analysis routine `cho_dc`

`in_node` : Name of the node at which an input signal is to be applied.

`in_var` : Name of the nodal variable to be excited.

`out_node` : Name of the node where the response is to be observed.

`out_var` : Name of the nodal variable to be observed.

Example

```
net = multimode;
dq = cho_dc(net);
find_ss(net, dq, 'node10', 'y', 'node14', 'y');
```

Caveats

Steady-state frequency response analysis currently fails for devices involving purely algebraic constraint. Such devices include electrical resistor networks with no inductances or capacitances considered, for example.

The steady-state analysis routines currently use functions from Matlab's Control Toolbox, which may be unavailable to some Matlab users.

6 Transient analysis

Calling sequence

```
function [T,Q,C,G] = cho_ta(net,tspan,q0)
```

Simulate the behavior of the device over some time period.

Inputs

`net` : Netlist structure returned from calling the compiled netlist

`tspan` : Two-element vector `[tstart tend]` indicating the start and end times for the simulation.

`q0` : (Optional) Initial state at `tstart`. If `q0` is not provided, the default is zero.

Outputs

`T` : Time points where the solution was sampled

`Q` : Array of state vectors sampled at the times in `T` (i.e. `Q(i,:)` is the state vector at time `T(i)`)

Example

```
net = beamgap;                         % Load the netlist
[T,Q] = cho_ta(net,[0 1e-3]);          % Simulate 1 ms behavior
dy = cho_dq_view(Q, net, 'c', 'y');    % Get the y component at c, and
plot(T, dy);                           %  plot how it moves over time
```

Caveats

The transient analysis routines currently take an impractically long time to simulate even some simple examples over modest time spans (like a millisecond). Mixed electrical-mechanical simulations are particularly problematic.

Like frequency-response analysis, the transient analysis routine fails completely for devices involving purely algebraic constraints.