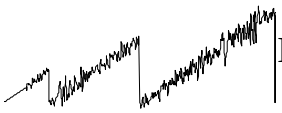
Draft

CU-NEES-06-3



NEES at CU Boulder

01000110 01001000 01010100

The George E Brown, Jr. Network for Earthquake Engineering Simulation

The CU-Boulder Fast Hybrid Test Desktop Platform

by

Dr. Gary Haussmann

Software Engineer

June 2006

Department of Civil Environmental and Architectural Engineering
University of Colorado
UCB 428
Boulder, Colorado 80309-0428

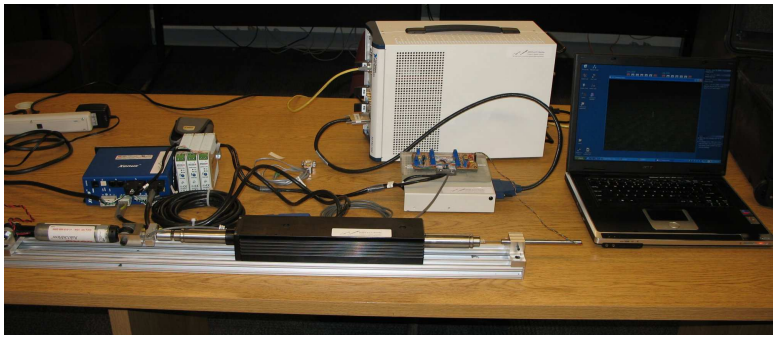


Figure 1: Desktop Platform

1 Introduction

While the CU-Boulder NEES site has implemented a complete FHT solution, the high cost and effort involved in setting up and running an FHT test may prohibitive for certain applications, including:

- Live demonstration of the FHT method in offsite locations.
- Interactive experiments or classroom presentations.
- Fast prototyping of new engineering ideas or concepts

To address these problems, a high performance desktop platform for realtime hybrid simulation is being developed at CU NEES. The hardware and software requirements will provide basic FHT functionality with relatively lower cost and more portability than full-scale hybrid test sites.

2 Overview

The desktop platform includes a full implementation of the fast hybrid method used in the CU production FHT test lab. A multiple-DOF simulation analyzes the structure while the actuators and instruments (available using LabView VI modules) drive the test specimen and inject measurements back into the simulation, Fig. 2. The simulation method uses an implicit scheme for stability and an iterative solution method to handle nonlinear responses.

The simulation and instrumentation are handled by a single computer platform, called the target platform. The target platform uses a real-time operating system to insure a deterministic response for the hybrid test. The target platform sends simulation state and other information over a network to a second computer, called the host computer. The host computer runs a constantly updated visualization to show the states of the simulation structure for debugging and demonstration. The host platform also allows for user interaction to start and stop the simulation, and reconfigure the instrumentation as accessed in LabView.

This system is composed of the following:

Hardware

- Two personal computers

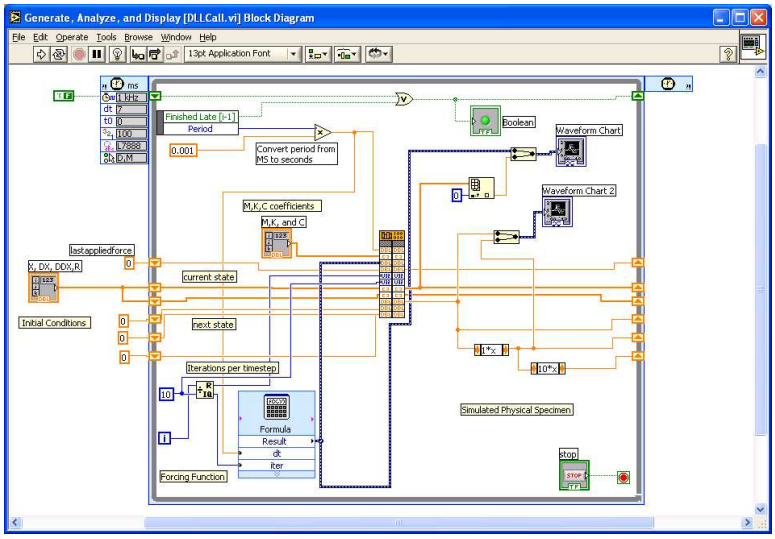


Figure 2: LabView Framework for the Desktop Platform System

- One multifunction DAQ board
- One or more actuators (each with a force and displacement measurement sensor)
- One test structure

Software

- National Instruments Labview & Real-time module
- Custom Realtime Hybrid Simulation Block or VI to be used in LabView
- Graphical user interface to facilitate an working understanding of what is happening during a realtime hybrid simulation.

3 Hardware Description

The FHT Desktop system includes:

- Two personal computers
- One multifunction DAQ board
- One or more actuators (each with a force and displacement measurement sensor)
- One test structure

The two personal computers are used for simulation and user interface; one computer is used for simulation/instrumentation and the other is used for graphical display and user control. The two computers are connected using an Ethernet connection, which is used to transfer simulation data between the two computers. The first computer, the simulation computer, contains a real-time operating system which executes the structural dynamics simulation and also manipulates the physical specimen using actuators and displacement/force measurements, Fig. 3.

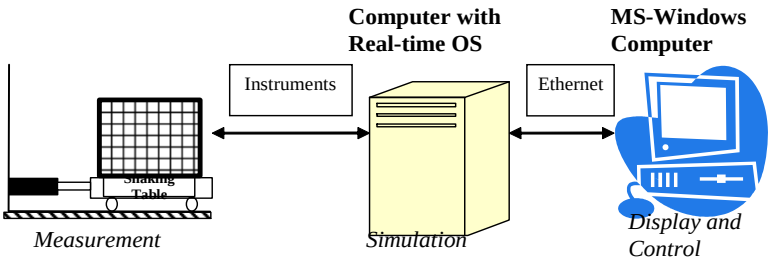


Figure 3: Hardware for the Desktop Platform

The simulation and manipulation are performed in real time, meaning that a complete simulation step and instrument reading happens every millisecond, without the delays or lag that may occur in a non-real-time environment. The results of simulation and measurement are transferred to a separately running program on the first computer which communicates with the second computer over an Ethernet network.

The second computer holds the program used for graphical display and user interaction. The current simulation state is displayed on this computer, as well as graphs and/or plots representing the physical specimen.

4 Software Description

The two computer platforms contain three separate computing processes among them. The first computing process is the real-time simulation/measurement process, which runs on the first computer and simulates the structure and interacts with the physical specimen. The second computing process also runs on the first computer and communicates directly with the real-time simulation process. This second computing process then sends data over the network to the third computing process which is used for graphical display and user interaction.

The simulation process is a finite-element simulation built on the second-order dynamics equation written in terms of the time-varying state, x, v and a:

$$\overline{M}a + s(v) + r(x) = f \tag{1}$$

Where $\overline{M}$ is the mass matrix, $s(v)$ is the damping force, $r(x)$ is the restorative force, and f is the excitation or external force. Here the quantity $\overline{M}$ is assumed constant and frequency-independent, while the damping force $s(x)$ and restorative force $r(x)$ are either general nonlinear functions, or linear responses of the form

$$s(v) = \overline{C}v \tag{2}$$
$$r(x) = \overline{K}x \tag{3}$$

where $\overline{C}$ and $\overline{K}$ are matrices representing the stiffness and damping matrices.

The equilibrium equations are discretized for a given timestep t_i

$$\overline{M}a_i + s_i + r_i = f_i \tag{4}$$

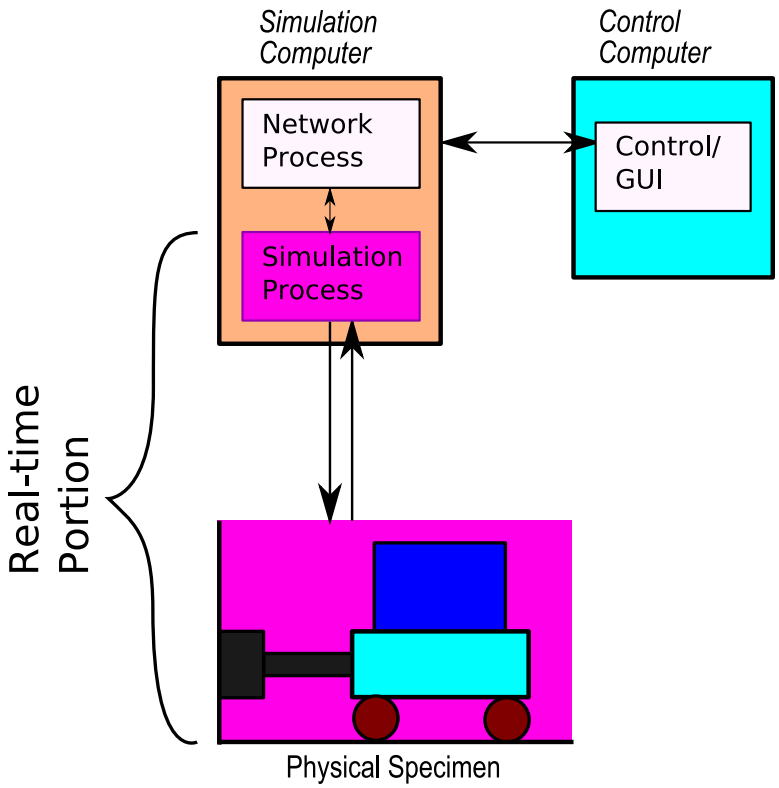


Figure 4: Software Configuration

The Fast Hybrid Testing (FHT) method at CU-Boulder uses a modified version of the alpha method, an implicit scheme, to compute the state variables for timestep t_{i+1}

$$\overline{M}a_{i+1} = (1+\alpha)f_{i+1} - \alpha f_i - \overline{C}\left[(1+\alpha)v_{i+1} + \alpha v_i\right] - (1+\alpha)r_{i+1} + \alpha r_i \quad (5)$$

$$d_{i+1} = d_i + \Delta t v_i + \Delta t^2 \left[\left(\frac{1}{2} - \beta\right) a_i + \beta a_{i+1}\right] \quad (6)$$

$$v_{i+1} = v_i + \Delta t \left[(1-\gamma)\, a_i + \gamma a_{i+1}\right] \quad (7)$$

If the restorative force is completely linear, i.e., $r(x) = \overline{K}x$ then the quantities for the next time step can be solved directly. However, if the restorative force is nonlinear, which occurs in a hybrid test, then the solution must be found by using iteration.

In a nonlinear simulation, the quantities d_{i+1} and r_{i+1} are solved using iteration. The explicit quantity

$$\begin{aligned}\tilde{M}\hat{d}_{i+1} = {} & \tilde{M}\left[d_i + \Delta t v_i + \Delta t^2 \left(\frac{1}{2} - \beta\right) a_i\right] \\ & + \Delta t^2 \beta \left[(1+\alpha)f_{i+1} - \alpha(f_i - r_i) - \overline{C}(v_i - \Delta t(1+\alpha)(1-\gamma)a_i\right] \end{aligned} \quad (8)$$

is computed, where $\tilde{M} = \overline{M} + \Delta t\gamma(1+\alpha)\overline{C}$, and the quantities d_{i+1} and r_{i+1} are then computed iteratively. These quantities at the next iteration $k+1$ are expressed in terms of values at the previous iteration k:

$$\Delta r^k_{i+1} = \left[\tilde{M}d^k_{i+1} - \tilde{M}\hat{d}_{i+1} + \Delta t^2\beta(1+\alpha)r^k_{i+1}\right] \quad (9)$$

$$\Delta d^k_{i+1} = K^{*-1}\Delta r^k_{i+1} \quad (10)$$

$$d^{k+1}_{i+1} = d^k_{i+1} + \Delta d^k_{i+1} \quad (11)$$

$$r^{k+1}_{i+1} = r^k_{i+1} + \Delta r^k_{i+1} \quad (12)$$

Here the tangent stiffness K^* represents the secant stiffness quantity; for a simulation this is computed, but for fast hybrid testing this value represents the stiffness of the physical specimen, which is typically not fully characterized. Therefore, for a FHT experiment the initial stiffness K of the test specimen is used.

For a hybrid test, the physical specimen must be integrated into the iteration and integration process. The FHT method shown here uses three quantities d^C, d^M, r^M to bridge the gap between computer simulation and physical experiment:

1. The command displacement d^C is produced by the simulation and used as a command signal to the actuator on the physical specimen
2. The feedback displacement d^M is measured on the physical specimen and transferred back to the simulation software.
3. The feedback force r^M is measured on the physical specimen and transferred back to the simulation software.

Thus during a hybrid experiment the iteration update process becomes

$$\Delta d^k_{i+1} = K^{*-1}\left[\tilde{M}d^k_{i+1} - \tilde{M}\hat{d}_{i+1} + \Delta t^2\beta(1+\alpha)r^k_{i+1}\right] \quad (13)$$

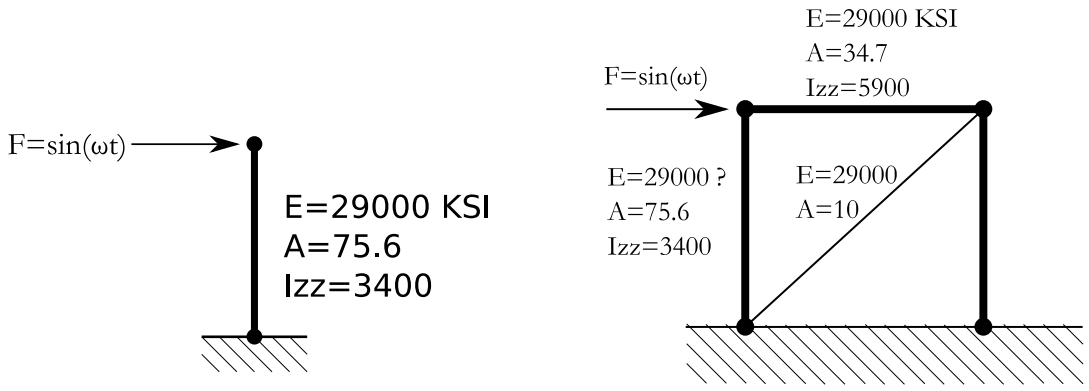
$$d^{C(k+1)}_{i+1} = d^{k+1}_{i+1} = d^k_{i+1} + \Delta d^k_{i+1} \quad (14)$$

$$r^{k+1}_{i+1} = r^{M(k+1)}_{i+1} + K^*\left(d^{C(k+1)}_{i+1} - d^{M(k+1)}_{i+1}\right) \quad (15)$$

where the iteration process has been updated to included the command displacement d^C that is sent to the physical actuator, and the measured feedback represented by d^M and r^M.

5 Comparison and Validation

Of key importance is the requirement that the FHT desktop produce simulation results resembling or identical to the simulations of the production system. Without a close correspondence, prototypes developed on the FHT desktop would not behave correctly when transferred over to the production system. In order to verify the correctness and resemblance of the FHT desktop and the production FHT system, two representative dynamics problems were simulated on both FHT systems and compare for differences.



(a) Case 1 (b) Case 2

Figure 5: Models Used for Simulation Validation

In both validation cases, the structure was simulated using an implicit alpha method for transient analysis, and the displacements at various locations were compared between the two simulation programs. In all cases, the difference in displacement values between the two programs was at or slightly larger than error explained by machine rounding error.

5.1 Single Cantilever Problem

The first problem used for validation was the simplest possible: a single cantilever, represented by an elastic beam fully constrained at one end. An excitation force is applied to the free end of the cantilever with the waveshape of a single frequency sinusoid. The model configuration is shown in figure 5(a) and the simulation results, displaying x-axis displacements, are show in figure 6.

5.2 Braced Single-Bay Problem

The second problem examined for validation purposes was a single-bay structure composed of beam elements, with a single bracing truss element (figure 5(b)) and the same excitation force as was used in the previous validation. It should be noted that the use of a non-zero beta value for Rayleigh damping produced an unexplained instability in OpenSEES, so the Rayleigh damping for this model uses different coefficients than Case 1. The displacement results are shown in figure 7.

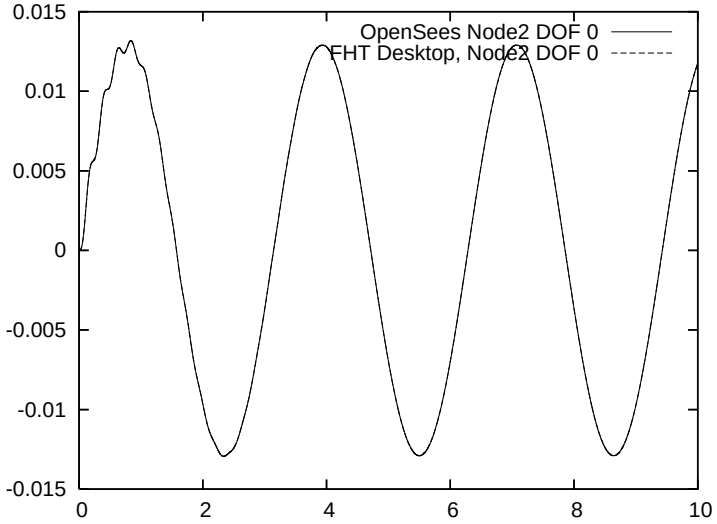


Figure 6: Displacement results for Case 1 simulation

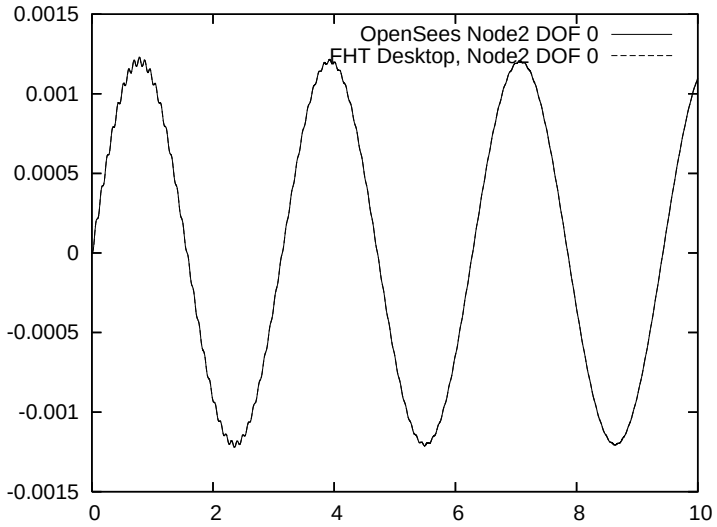


Figure 7: Displacement results for Case 2 simulation

6 Conclusion and Summary

The two computer platforms contain three separate computing processes among them. The first computing process is the real-time simulation/measurement process, which runs on the first computer and simulates the structure and interacts with the physical specimen. The second computing process also runs on the first computer and communicates directly with the real-time simulation process. This second computing process then sends data over the network to the third computing process which is used for graphical display and user interaction.

The FHT Desktop Platform provides a low-cost easily deployable implementation of the Fast Hybrid Test method. The small size and easy accessibility make the platform useful for outreach demonstrations and pedagogical purposes. Also, low cost and short setup times allow the platform to be used for prototyping of larger-scale FHT experiments.

7 Acknowledgements

The technical support of Dr. Eric Stauffer (CU-NEES Technical Director), and the financial support of the University of Colorado Chancellor's Fund for CU-NEES are gratefully acknowledged.

8 Appendix: Validation Case Information

Enclosed in this appendix are the text/code sequences used to generate the validation models. OpenSEES models are written in TCL, while FHT models are constructed using C++.

8.1 Validation Case 1

OpenSEES construction:

```
#
# modified 7-26-2006
#
# Test Case #1: Single column 3dof
#

model BasicBuilder -ndm 2 -ndf 3 node  1    0      0 node  2 0 156.0

fix     1    1    1    1

geomTransf PDelta 1
element elasticBeamColumn   1 1 2 75.6 29000 3400 1 mass 2 0.1  0.1  0.1

system UmfPack constraints Plain
algorithm Linear
numberer RCM
integrator  Newmark 0.5 0.25 0.4503 0.0 0.0023 0.0
analysis Transient
```

```
#                    time start stop  period
set SineTimeSeries "Sine 0.0  50.0  3.14159"
set XAmp 1
set YAmp 0
set ZAmp 0

#Apply sine varying force at node 2
pattern Plain 1 $SineTimeSeries {;
        load 2 $XAmp $YAmp $ZAmp;
}

recorder Node  -file case1node2disp.out  -time -node 2  -dof 1 2 3 disp
recorder Node  -file case1node1acc.out  -time -node 2  -dof 1 2 3 accel
```

FHT desktop construction:

```
FENode p1(1,0,0), p2(2,0,156);
p1.constrainDOF(0);
p1.constrainDOF(1);
p1.constrainDOF(2);
p2.addMass(0,0.1);
p2.addMass(1,0.1);

domain.addNode(p1);
domain.addNode(p2);

FEElement *e1;
e1 = new FE2DFrame(1, 29e3, 75.6, 3400 );
e1->setEndpoints(p1.getID(), p2.getID());
domain.addElement(e1);

stiffnessmatrix msm,ktt, mass, damper;
domain.assembleMatrices(msm, ktt, mass, damper);

// build the dynamics
const double alpha = 0.0;
const double beta = 0.25 * (1-alpha)*(1-alpha);
const double gamma = 0.5 - alpha;
const double dt = 0.001;
const double omega = 2.0;
const int freeDOF = (int)(mass.size1());
const int totalDOF = (int)(msm.size1());
AlphaDynamics sim(alpha,beta,gamma,dt);
sim.setDOF(freeDOF);
sim.setMatrices(mass, ktt, damper);
```

8.2 Validation Case 2

OpenSEES construction:

```
#
# modified 8-02-2006
#
# Test Case #2:  Single bay frame
#

model BasicBuilder -ndm 2 -ndf 3

node  1    0      0
node  2    0      156.0
node  3    360    156
node  4    360    0

fix     1    1    1    1
fix     4    1    1    1

geomTransf Linear 1
set pdelta off


# fields are ID, stiffness, tangent stiffness
uniaxialMaterial Elastic 1001 29000 0.0

# fields are id, node1, node2, A, E, Izz, ?
element elasticBeamColumn   1         1  2 75.6 29000 3400  1
element elasticBeamColumn   2         2  3 34.7 29000 5900  1
element elasticBeamColumn   3         3  4 75.6 29000 3400  1
element truss               4         1  3 10 1001

mass  2  0.1  0.1  0.1
mass  3  0.1  0.1  0.1

system UmfPack
constraints Plain

algorithm Linear

numberer RCM

integrator  Newmark 0.5 0.25 0.5 0.0 0.0 0.0

analysis Transient

#                   time start stop  period
```

```
set SineTimeSeries  "Sine 0.0  50.0  3.14159"
set XAmp 1
set YAmp 0
set ZAmp 0
#Apply sine varying force at node 2
pattern Plain 1 $SineTimeSeries {;
    load 2 $XAmp $YAmp $ZAmp;
}

recorder Node  -file case2node2disp.out  -time -node 2  -dof 1 2 3 disp
recorder Node  -file case2node3disp.out  -time -node 3  -dof 1 2 3 disp
```

FHT desktop construction:

```
FENode p1(1,0,0), p2(2,0,156), p3(3, 360,156), p4(4,360,0);
p1.constrainDOF(0);
p1.constrainDOF(1);
p1.constrainDOF(2);
p4.constrainDOF(0);
p4.constrainDOF(1);
p4.constrainDOF(2);
p2.addMass(0,0.1);
p2.addMass(1,0.1);
p2.addMass(2,0.1);
p3.addMass(0,0.1);
p3.addMass(1,0.1);
p3.addMass(2,0.1);

domain.addNode(p1);
domain.addNode(p2);
domain.addNode(p3);
domain.addNode(p4);

FEElement *e1, *e2, *e3, *e4;
e1 = new FE2DFrame(1, 29e3, 75.6, 3400 );
e1->setEndpoints(p1.getID(), p2.getID());
domain.addElement(e1);
e2 = new FE2DFrame(2, 29e3, 34.7, 5900);
e2->setEndpoints(p2.getID(), p3.getID());
domain.addElement(e2);
e3 = new FE2DFrame(3, 29e3, 75.6, 3400);
e3->setEndpoints(p3.getID(), p4.getID());
domain.addElement(e3);
e4 = new FE2DTruss(4, 29000, 10);
e4->setEndpoints(p1.getID(), p3.getID());
domain.addElement(e4);

stiffnessmatrix msm,ktt, mass, damper;
```

```
domain.assembleMatrices(msm, ktt, mass, damper);

// build the dynamics
const double alpha = 0.0;
const double beta = 0.25 * (1-alpha)*(1-alpha);
const double gamma = 0.5 - alpha;
const double dt = 0.001;
const double omega = 2.0;
const int freeDOF = (int)(mass.size1());
const int totalDOF = (int)(msm.size1());
AlphaDynamics sim(alpha,beta,gamma,dt);
sim.setDOF(freeDOF);
sim.setMatrices(mass, ktt, damper);
```