Design of a Scripting Language for the Energy-Based Analysis of Nonlinear Structural Systems

Mark A. Austin,

Institute for Systems Research,

University of Maryland, College Park

E-mail : austin@isr.umd.edu

WWW : http://www.isr.umd.edu/~austin/

Co-workers : Wane-Jang Lin, Xiaoguang Chen.

Acknowledgements : National Science Foundation, Federal Highway Adminstration.

INSTITUTE FOR SYSTEMS RESEARCH

Overview of Talk

Aladdin. A new computational toolkit for the interactive matrix and finite element analysis of engineering structures.

Target applications. Static and dynamic analysis of building and highway bridge structures. The response of structures due to earthquake loadings is of particular interest.

Overview. This talk will cover:

- Part 1 : Motivation for work;
- Part 2 : Systems design for Aladdin;
- Part 3 : Features of the language;
- Part 4 : Matrix and engineering examples;
- Part 5 : Conclusions and future work;

Part 1 : Motivation for Work

Some observations on current trends

- Engineering systems are rapidly becoming more complex (e.g. number of system components; different types of interacting system components and subsystems).
- Complex engineering systems have time-varying and user-specific requirements, implying the need for flexibility in implementation.
- Design Teams are becoming geographically dispersed.
- Design Teams are looking to computing technology for assistance in management of project complexity (e.g. software simulators, Internet, WWW).
- New simulation/optimization tools are needed for understanding system behavior, and thorough exploration of the design space, a benefit for both large and small designs.
- Information services needed for end-to-end project development : project specification, behavior, evaluation, verification, maintenance, retirement.

Part 1 : Motivation for Work

State-of-the-art matrix and finite element analysis

- **Finite element programs.** Most finite element programs offer a restricted, but well implemented set of numerical procedures for static structural analysis and linear/nonlinear time-history calculations.
- **Finite element programs.** Are mostly stand-alone packages. They have not been designed for interaction with programs in other disciplines (e.g., control, optimization, design rule checking).
- **Problem description files.** are notoriously difficult to read ... what is the problem being solved?
- **Problem description files.** Physical units are neither an integral part of the problem decription, nor its solution procedure.
- **Matrix environments.** MATLAB-type environments are not well suited to F.E. computations.

Part 1 : Project Objectives

Project goals

- **Design.** High-level problem description language for matrix and finite element computations.
- **Design.** Low-level data structures and algorithms to incorporate physical units in matrix/finite element computations.
- **Verify.** Systems performance by applying environment to solution of a wide variety of engineering system simulations.

Observation

Although the `C Programming Language` has only 32 keywords and a handful of control structures, its judicious combination with external libraries has resulted in the language being used in a wide range of problem solving domains.

INSTITUTE FOR SYSTEMS RESEARCH

Part 2 : Systems Design

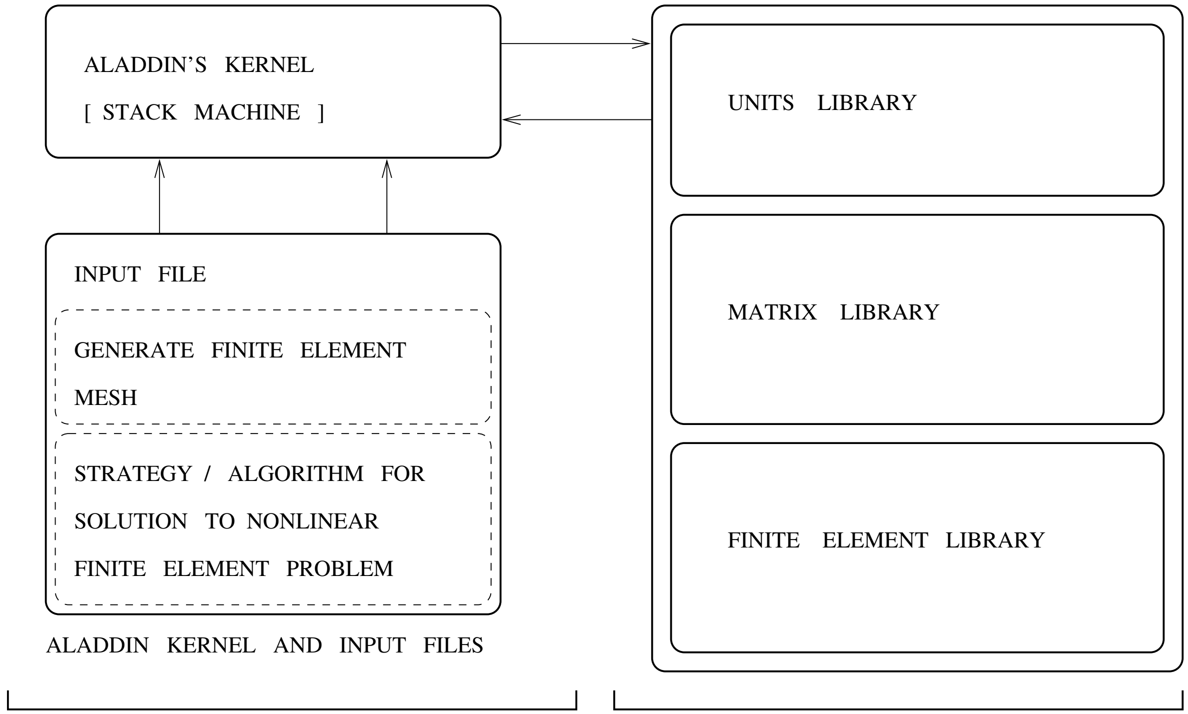
Aladdin is composed of four parts:

- **Model.** A **model** for the storage of physical quantities, matrices, and finite element information, and a stack machine for the execution of the matrix and finite element algorithms;
- **Language.** A **language** for describing physical quantities, matrices, finite element meshes, and problem solving procedures (it acts as the user interface to the underlying model); and
- **Well defined steps.** A family of **well defined steps** for transformations that can be carried out on system components (e.g., nearly all engineering processes require iteration and branching).
- **Guidelines for use.** Online documentation at:

`http://www.isr.umd.edu/~austin/aladdin.html`

.

Part 2 : High-Level Aladdin Architecture



SYSTEMS INTEGRATION

- Design of Language Syntax and Symantics for integration of system components.
- Hundreds / Thousands of low-level instructions per Aladdin statement.

SYSTEM LIBRARIES

- Design of data structures and algorithms for module libraries.
- Implemented in C. 5-10 low-level instructions per statement of C source code.

INSTITUTE FOR SYSTEMS RESEARCH

Part 2 : Systems Design

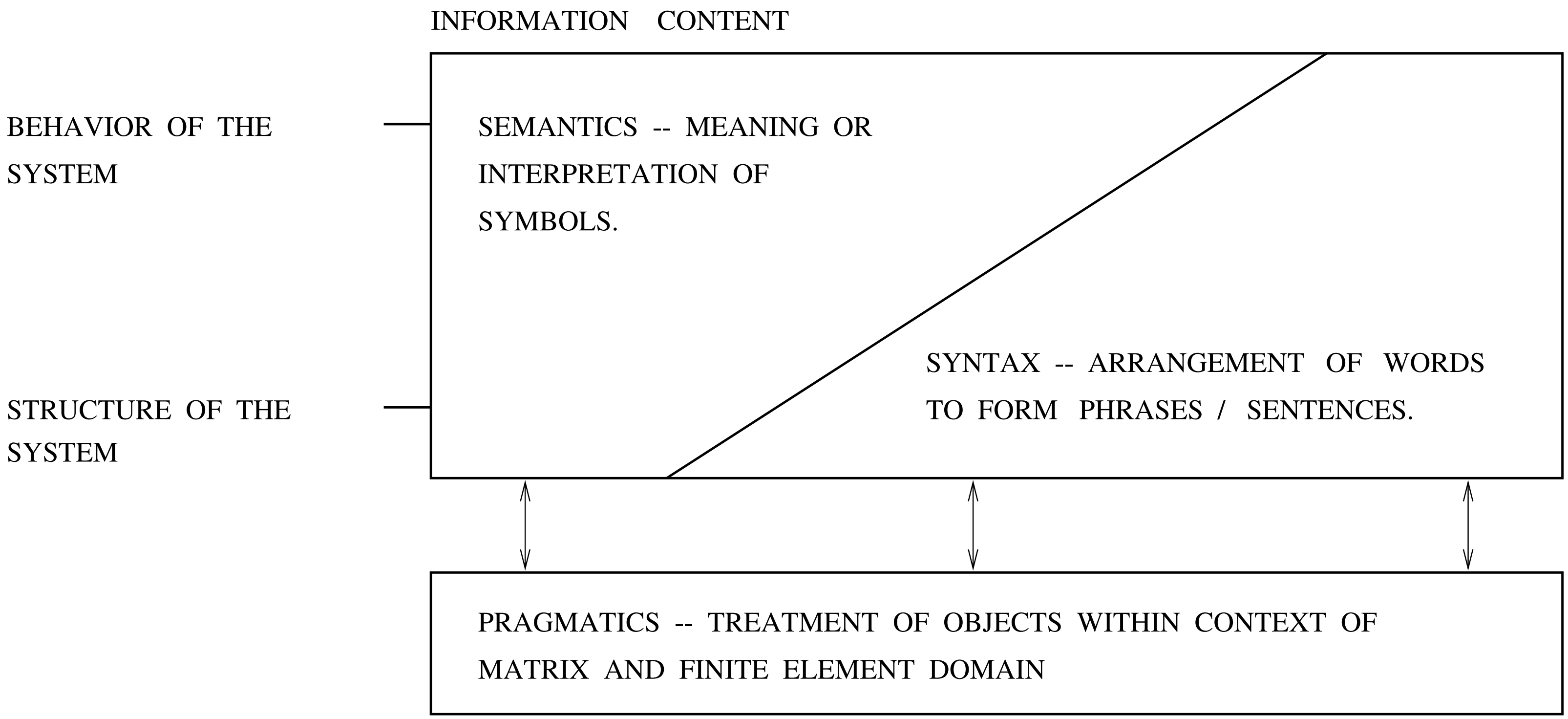
Matrix library

* Dynamic allocation of matrices,
* Printing of Matrices,
* Matrix Arithmetic
* Solution of Linear Equations
* Eigenvalues/Eigenvectors.

Finite element library

* Mesh Generation
* External Loads
* Boundary Conditions
* Section/Material Properties
* Linking of FE mesh d.o.f.
* 2 node frame elements
* 4 node plate element.
* 4 node flat shell element.
* Nonlinear Fiber beam-column elements.

Part 3 : Design and Implementation of Aladdin Language

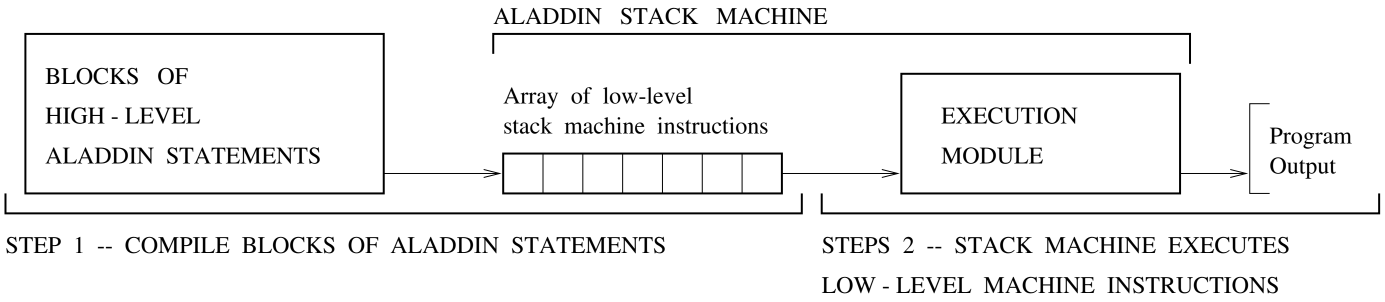


Challenge!

How to design easy-to-use language syntax and semantics so that pragmatics reflect an engineers understanding of the problem domain?

Part 3 : Processing Blocks of Aladdin Statements

Processing pathway for blocks of Aladdin statements

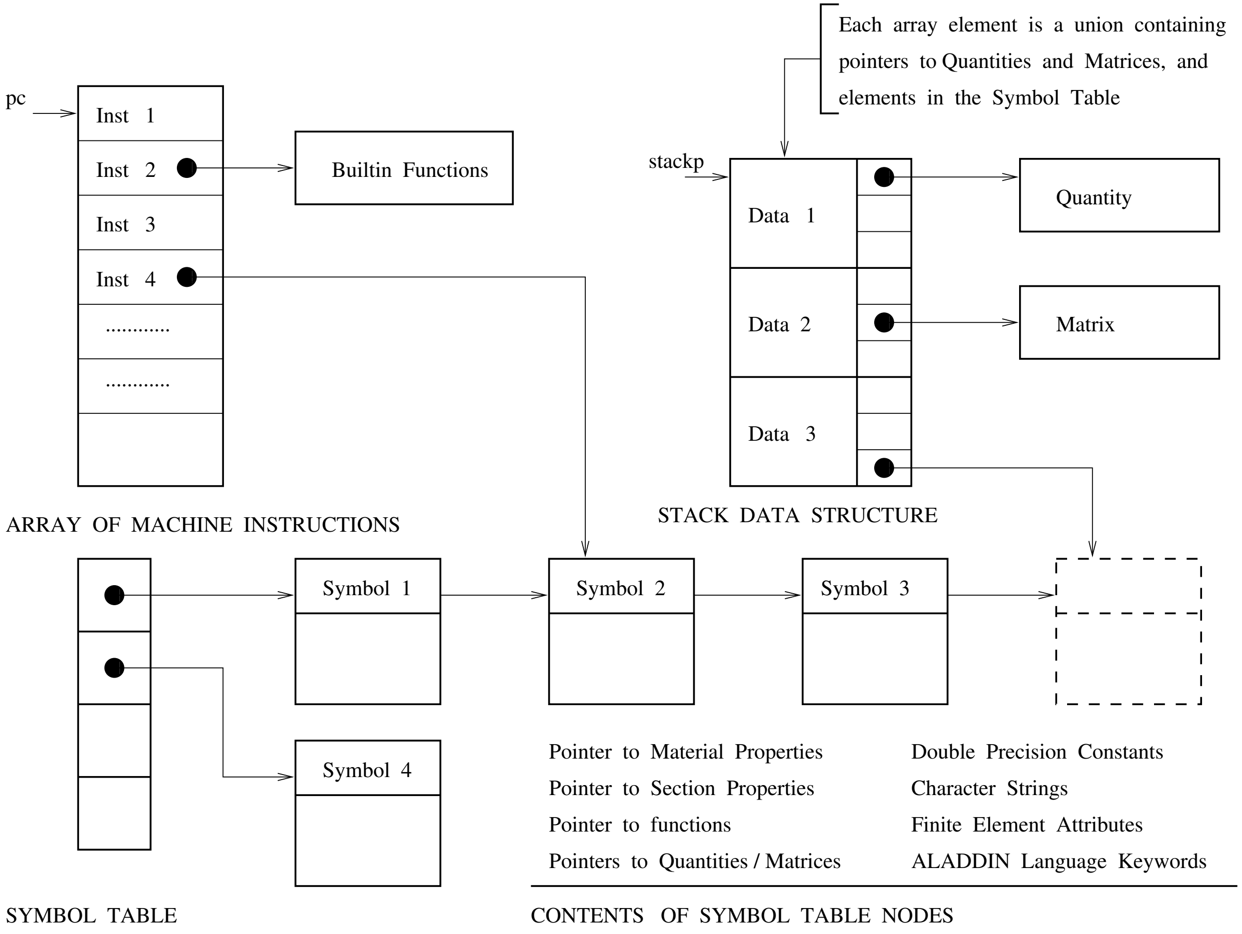


- **Step 1:** The parser reads blocks of language statements from the keyboard/ input file and constructs an array of low-level stack machine instructions.

  Machine instructions include calls to functions, and push/pop data operands to/from the program stack etc ...

- **Step 2:** The stack machine walks along the array of machine instructions and executes the functions pointed to by the machine instructions.

Part 3 : Aladdin's Stack Machine Architecture



INSTITUTE FOR SYSTEMS RESEARCH

Part 3 : Key features of the Aladdin Language

Aladdin problem descriptions/solution procedures are a composition of three elements;

- **data** – physical quantities and matrices of physical quantites;
- **control** – branching and looping constructs.
- **functions** – provide a hierarchy to the problem solving process.

Aladdin views:

- Finite element computations as a specialised form of matrix computations.
- Matrices as rectangular arrays of physical quantities,
- Numbers as dimensionless physical quantities.

INSTITUTE FOR SYSTEMS RESEARCH

Part 3 : Features of the Aladdin Language

Physical quantities

A quantifiable aspect of the modeled world, defined by a numerical value plus physical units.

$$unit = k\mathcal{L}^{\alpha}\mathcal{M}^{\beta}t^{\gamma}\mathcal{T}^{\delta}$$

Definition of physical quantities

```
v = 3 cm/sec;           /*  <--- A velocity              */

E = 200000 MPa;         /*  <--- Modulus of Elasticity */
I = 15.5e+06 mm^4;      /*  <--- Moment of Inertia     */
L = 4.50 m;             /*  <--- A length              */
```

Part 3 : Features of the Aladdin Language

Physical quantity : arithmetic operations

```
v = 3 cm/sec;           /*  <--- A velocity                 */
t = 1 sec;              /*  <--- time                       */

x = v*t                 /*  <--- Distance = velocity * time */
```

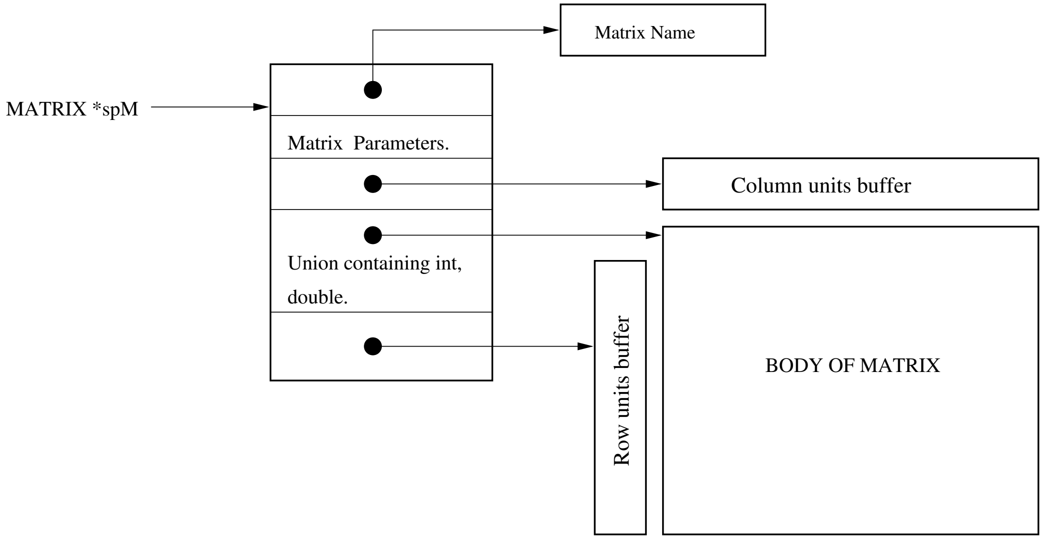
Physical quantity : relational operations

```
v <= 10 cm/sec          /*  <--- Evaluates to "true"      */
v  >  5 cm/sec          /*  <--- Evaluates to "false"     */
v >=  5 cm/sec^2        /*  <--- Units are inconsistent!  */
```

INSTITUTE FOR SYSTEMS RESEARCH

Part 3 : Features of the Aladdin Language

Layout of memory in matrix data structure



Examples

```
Force = [ 1.5 N, 1.5 N, 20.0 N*m ];

Elcentro = ColumnUnits( [ 14.56;    13.77;     6.13;  ..........
                         -12.35;   -15.72;     0.00  ], [cm/sec/sec] );
```

Part 3 : Features of the Aladdin Language : Looping Constructs

Example : The "for" and "while" looping constructs

```
Aladdin source code                                Aladdin output
===================                                ==============


for ( x = 1 cm; x <= 3 cm; x = x + 1 cm) {         x = 1 cm;
      print "x = ", x, "\n";                       x = 2 cm;
}                                                  x = 3 cm;

y = 1 cm;
while ( y <= 3 cm )                                y = 1 cm;
      print "y = ", y, "\n";                       y = 2 cm;
      y = y + 1 cm;                                y = 3 cm;
}
```

Part 3 : Features of the Aladdin Language : Branching Constructs

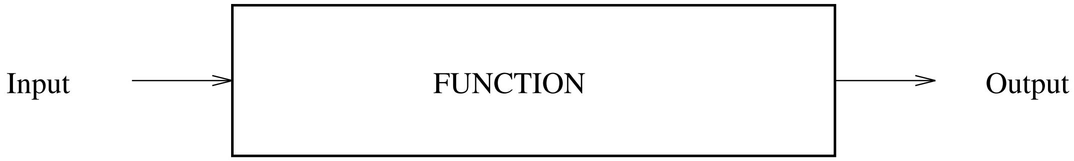
Example : The "if" and "if-else" branching constructs

```
Aladdin source code                                   Aladdin output
===================                                   ==============


x = 3 cm;                                             *** x^2 = 0.0009 cm^2
if ( x < 5 cm ) {
     print "*** x^2 = ", x^2, "\n";
}

y = 10 cm;                                            *** y^3 = 1000 cm^3
if ( y < 5 cm ) {
     print "*** y^2 = ", y^2 (cm^2), "\n";
} else {
     print "*** y^3 = ", y^3 (cm^3), "\n";
}
```

INSTITUTE FOR SYSTEMS RESEARCH

Part 3 : Features of the Aladdin Language : Functions

Function interface

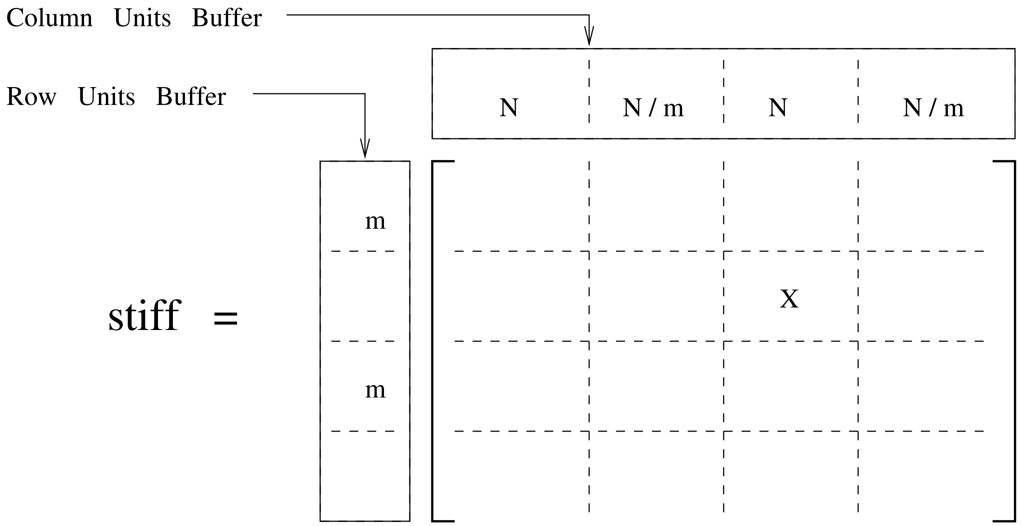


$$\underbrace{\text{RETURN TYPE}}_{\text{Output}} = \text{FUNCTION NAME } (\underbrace{\text{Arg 1 , Arg 2 , ... , Arg N}}_{\text{Input}});$$

Aladdin source code

```
returntype1 = Function1();                                /* <= area 1 */
returntype2 = Function2();                                /* <= area 1 */

returntype3 = Function3( returntype1, returntype2 );  /* <= area 2 */
```

Matrix data structure



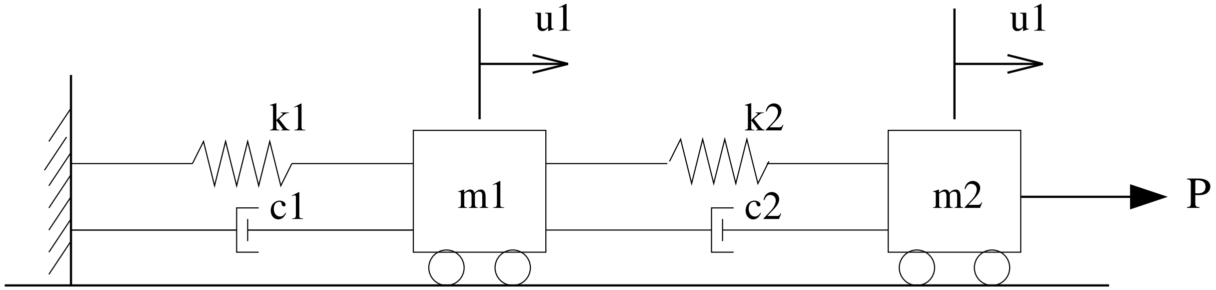
Aladdin source code

```
stiff = Matrix([4,4]);
stiff = ColumnUnits ( stiff, [N, N/m, N, N/m] );
stiff =    RowUnits ( stiff, [m], [1] );
stiff =    RowUnits ( stiff, [m], [3] );
```

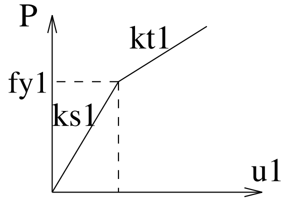
INSTITUTE FOR SYSTEMS RESEARCH

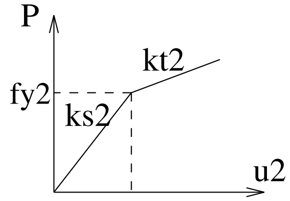
Example : Response of Nonlinear Mass-Spring System

Mass-spring system



ks1 = 2.0 N/cm	ks2 = 1.5 N/cm
kt1 = 0.8 N/cm	kt2 = 0.5 N/cm
fy1 = 18 N	fy2 = 15 N
m1 = 1.0 kg	m2 = 0.5 kg
c1 = 1.5 N·sec/m	c2 = 1.0 N·sec/m





Example : Response of Nonlinear Mass-Spring System

Definition of bilinear spring behavior

```
ks1 = 2.0 N/cm;   kt1 = 0.8 N/cm;    fy1 =  18 N;
ks2 = 1.5 N/cm;   kt2 = 0.5 N/cm;    fy2 =  15 N;
```

Initial structure tangent stiffness matrix

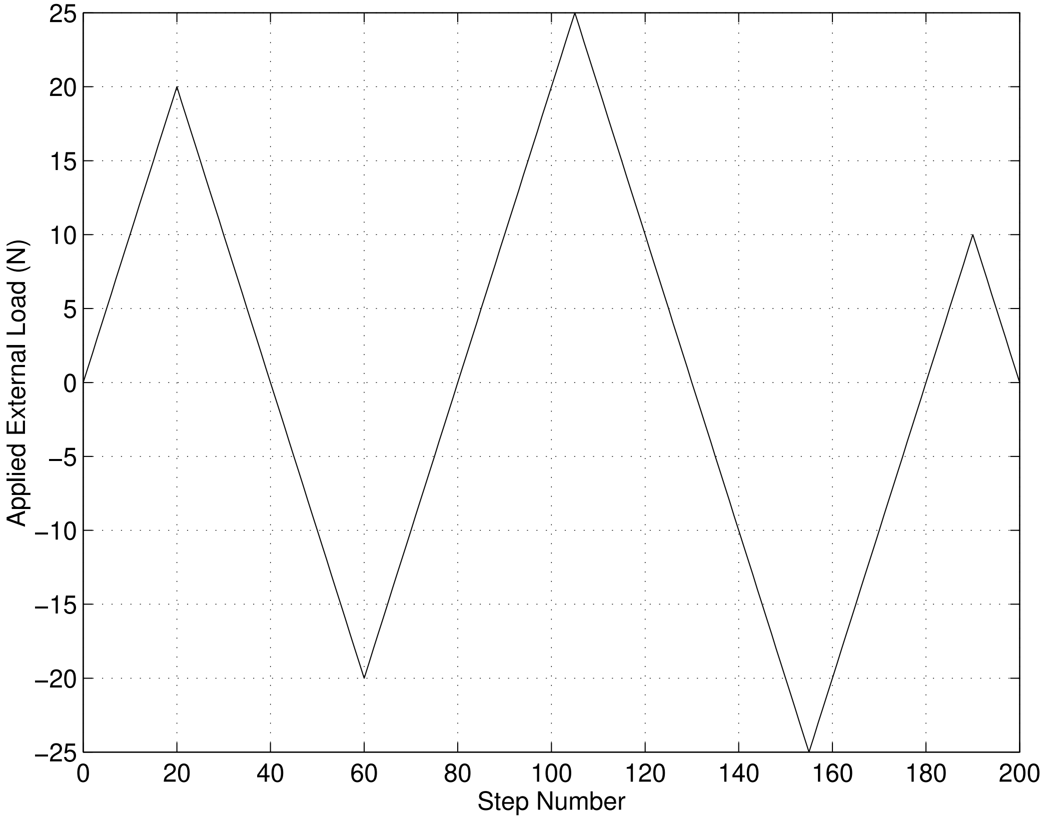
```
BigK = [ ks1+ks2, -ks2;
             -ks2,  ks2 ];
```

Mass matrix

```
BigM = [ 1.0 kg, 0.0 kg;
         0.0 kg, 0.5 kg ];
```

Example : Response of Nonlinear Mass-Spring System

Applied load versus step number



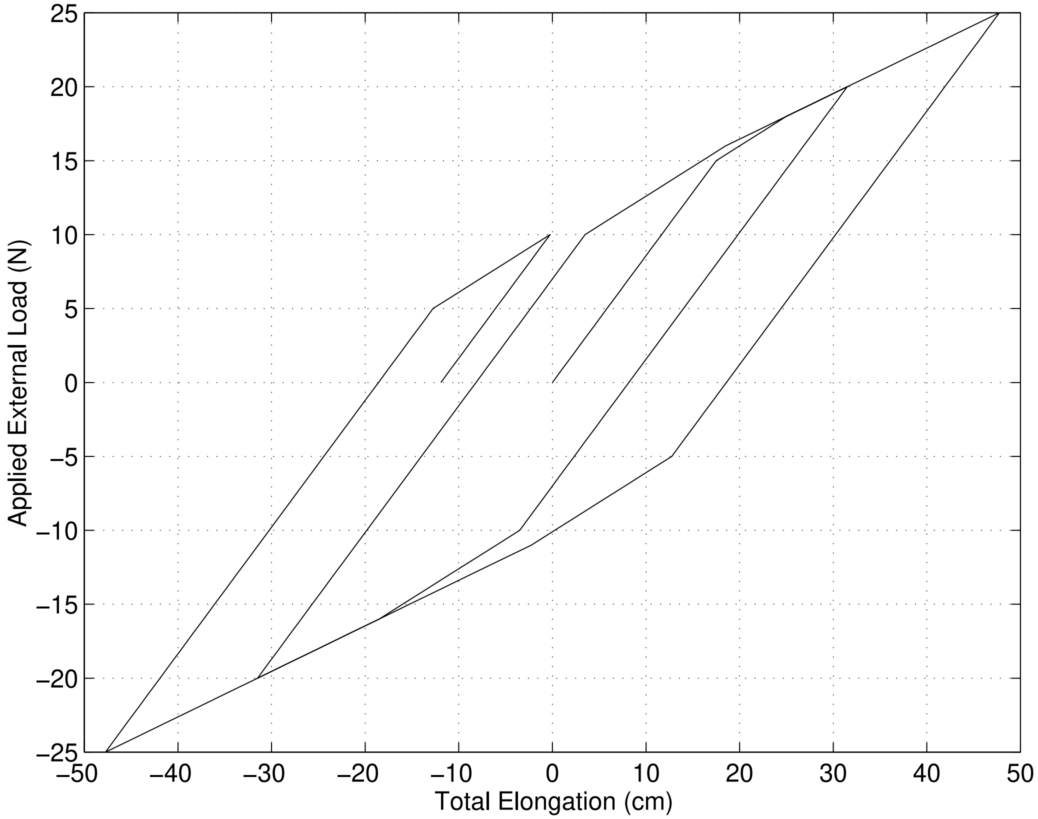
Example : Response of Nonlinear Mass-Spring System

Force increment at each step

```
for ( k=1; k<=total_step ; k=k+1 ) {

    /* define force increment for each step */

    if( k<=20 )           { d_P =  [0 N; 1 N]; }
    if( k>20  && k<=60  ) { d_P = -[0 N; 1 N]; }
    if( k>60  && k<=105 ) { d_P =  [0 N; 1 N]; }
    if( k>105 && k<=155 ) { d_P = -[0 N; 1 N]; }
    if( k>155 && k<=190 ) { d_P =  [0 N; 1 N]; }
    if( k>190 && k<=200 ) { d_P = -[0 N; 1 N]; }
    if( k>200 )           { d_P =  [0 N; 0 N]; }

    P = P + d_P;
    .........
```

Example : Response of Nonlinear Mass-Spring System

Load-deflection relationship



Example : Response of Nonlinear Mass-Spring System

Heart of incremental displacement solution procedure

```
for( step=1 ; step <= total_step ; step=step+1 ) {
     displ = displ + dp;
     ElmtStateDet(dp);         /* State determination for all elements */

     PR = InternalLoad(displ); /* Get structure resistant force        */
     UpdateResponse();         /* Update element information           */

     /* Calculate new delta displacement after yielding */

     /* Adjust delta displacement at yielding step */
}
```

Example : Response of Nonlinear Mass-Spring System

System eigenvalues

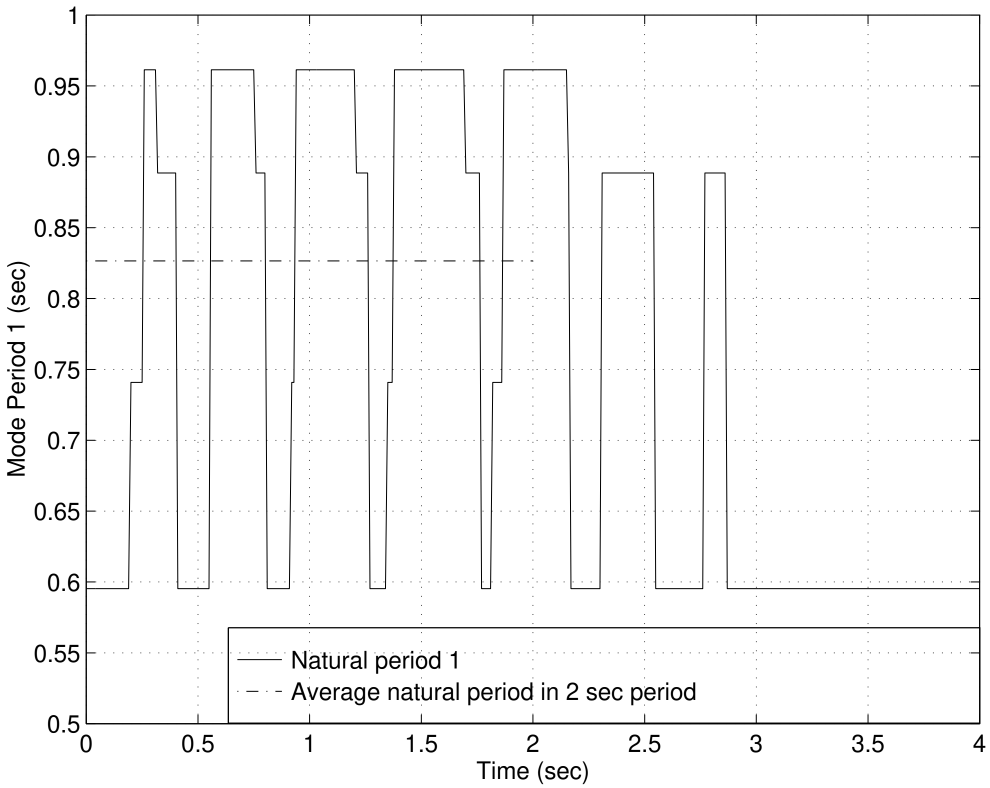
The time-variation in the first and second modes of vibration is computed by simply inserting:

```
eigen = Eigen(BigK, BigM, [2]);
eigenvalue  = Eigenvalue(eigen);

result[1][7] = 2*PI/sqrt( eigenvalue[1][1] );
result[1][8] = 2*PI/sqrt( eigenvalue[2][1] );
```

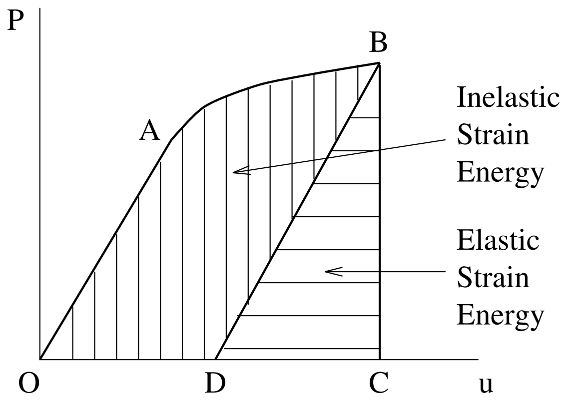
into the main loop of the time-history calculation.

Example : Response of Nonlinear Mass-Spring System

Mode 1 natural-period (sec) versus time (sec)



Example : System Energy Evaluation



Strain energy at load step n

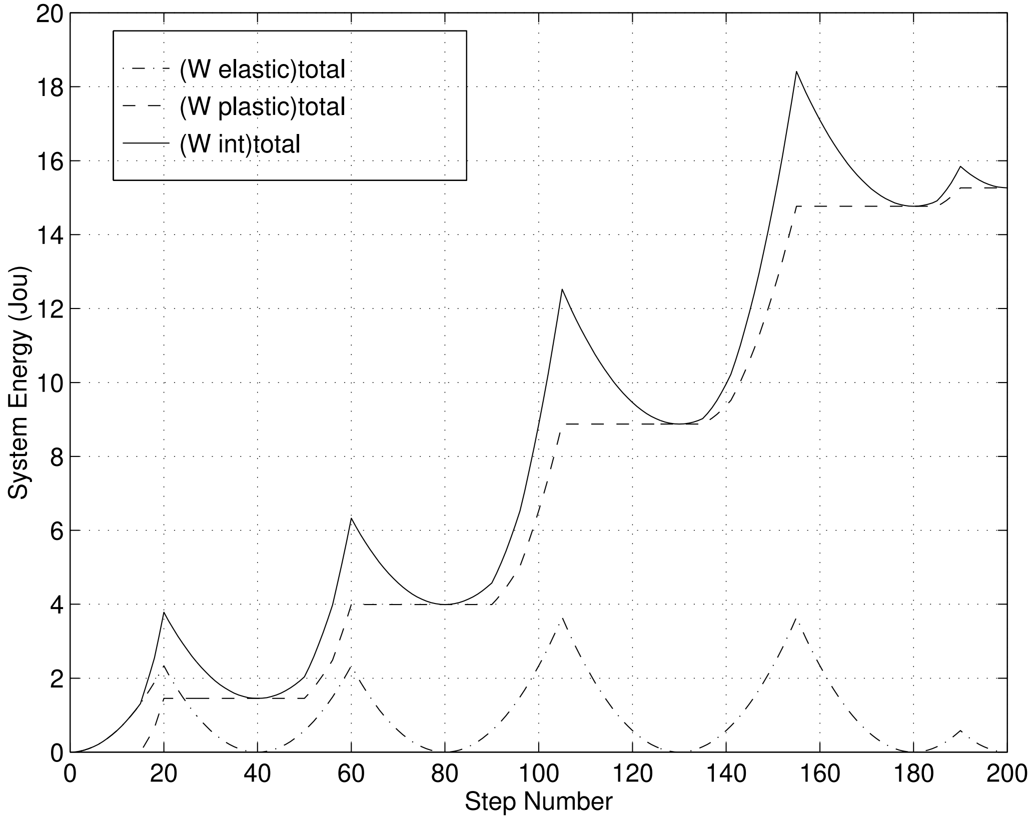
$$U_n = \int_0^{u_n} P(u) du \tag{1}$$

Discrete approximation of strain energy at load step n

$$U_n = Area_{OABCDO} = U_{n-1} + \frac{1}{2}(P_{n-1} + P_n) du_n \tag{2}$$

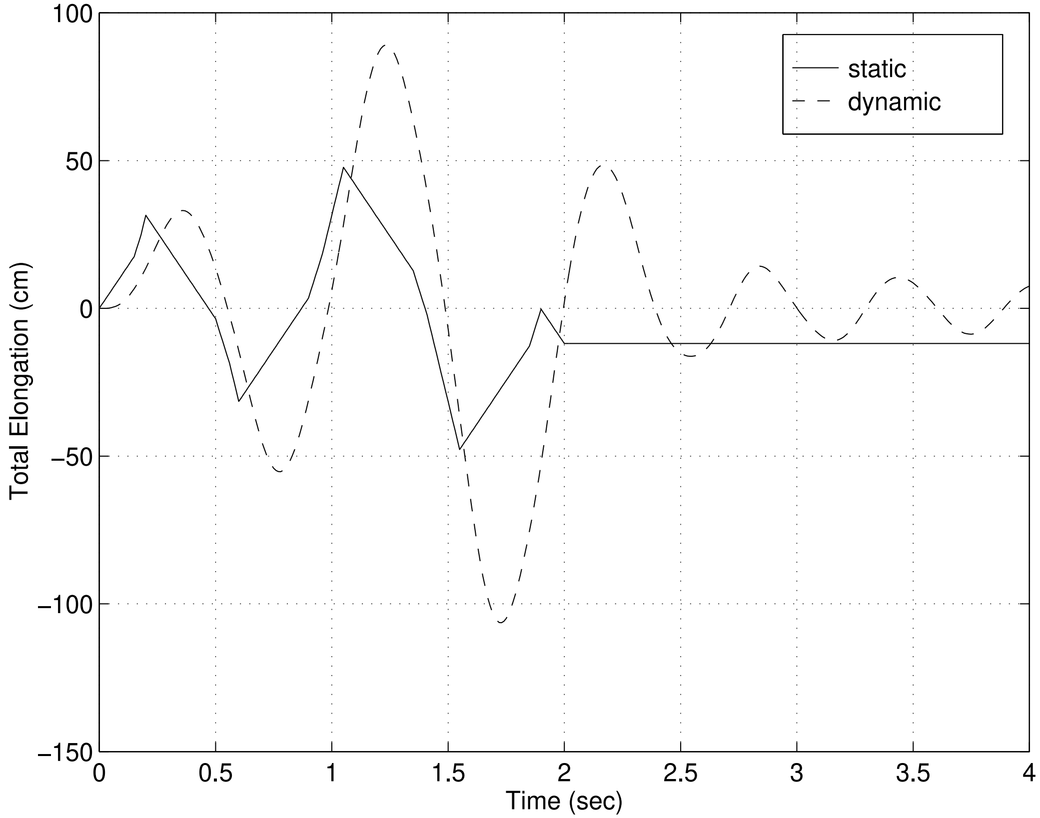
Example : Response of Nonlinear Mass-Spring System

System strain energy versus step no.



Example : Response of Nonlinear Mass-Spring System

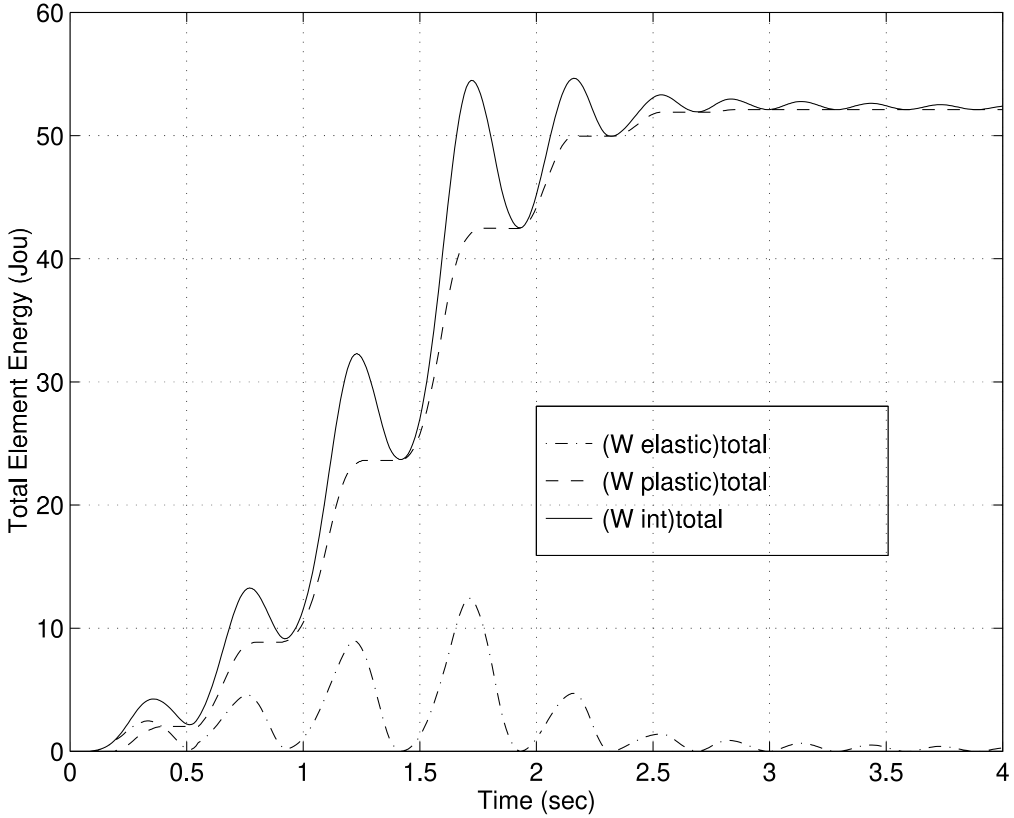
Time-history of total system displacement



![ISR Institute for Systems Research]

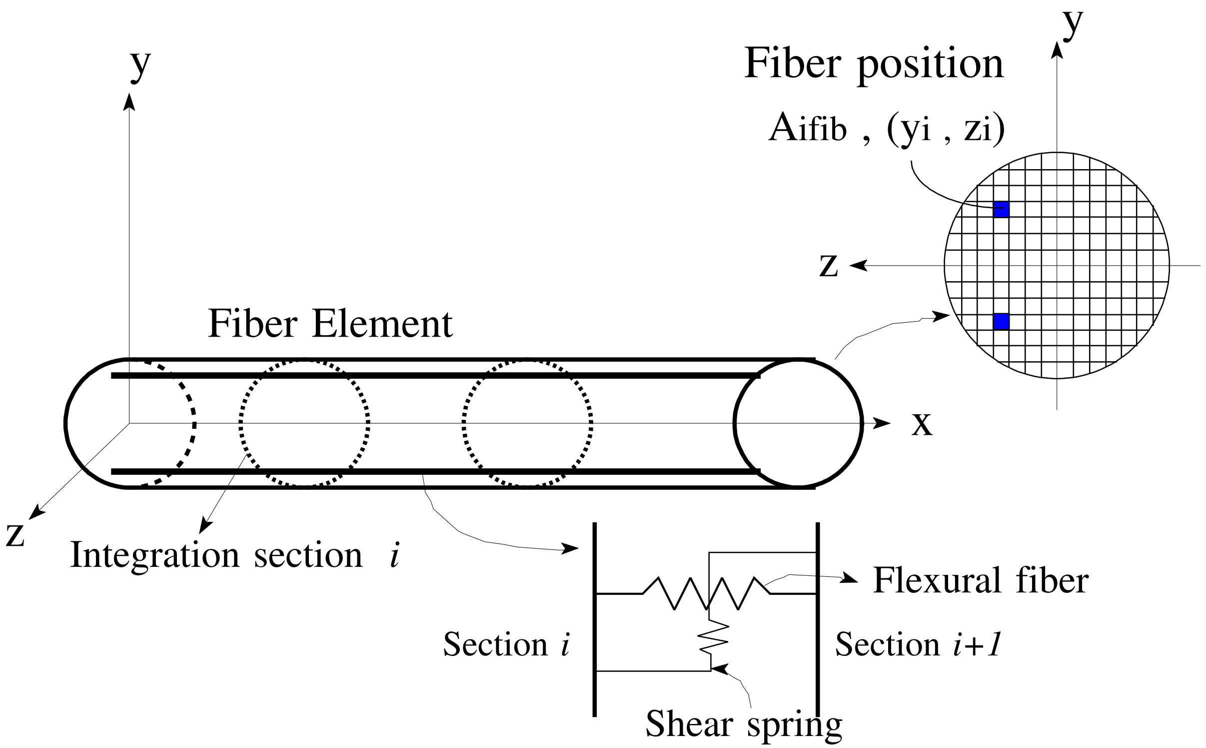
Example : Response of Nonlinear Mass-Spring System

Strain energy versus time (sec) for dynamic response



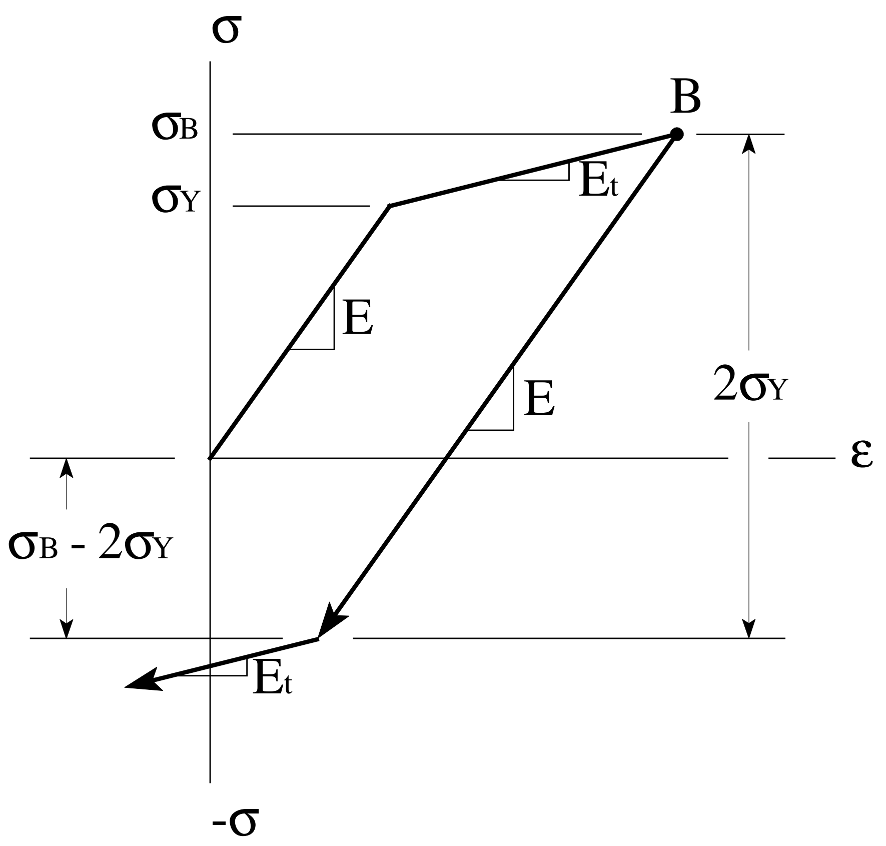
Recent Work : Fiber Finite Elements

Schematic for fiber finite elements



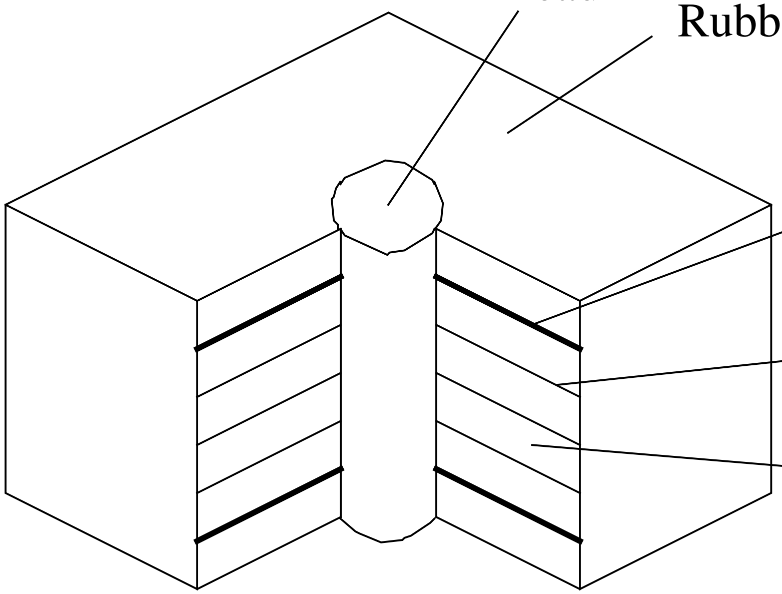
Recent Work : Fiber Finite Elements

Stress-strain relationship for fiber finite element

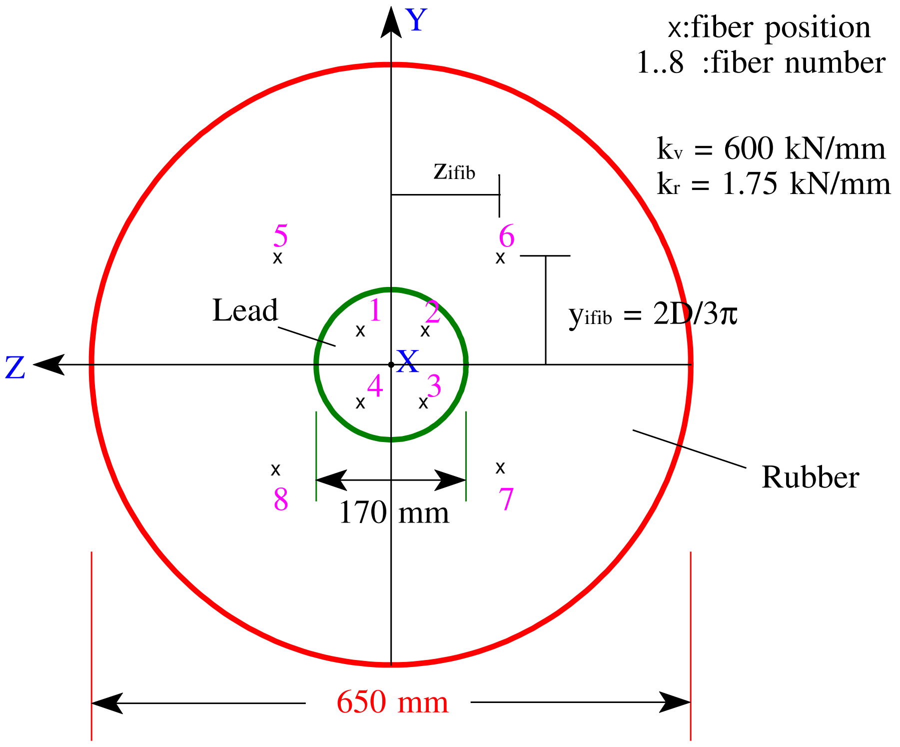


INSTITUTE FOR SYSTEMS RESEARCH

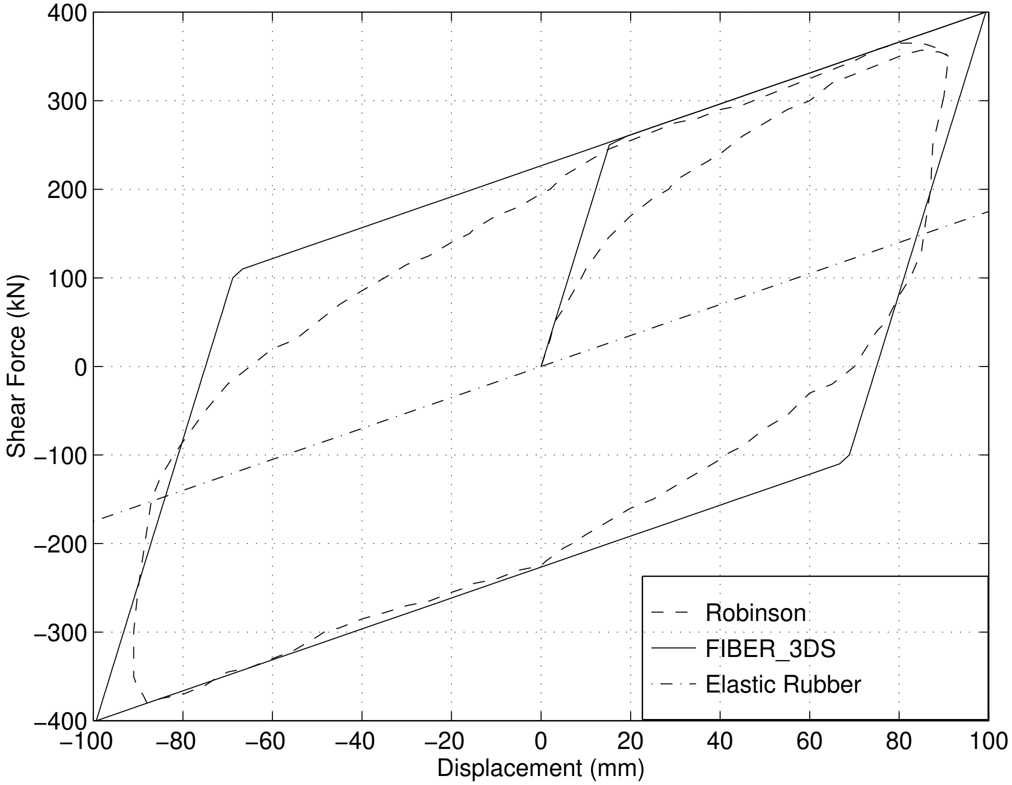
Example : Hysteretic Behavior of Lead-Plug Laminated Bearing



Example : Modeling of Lead-Plug Laminated Bearings

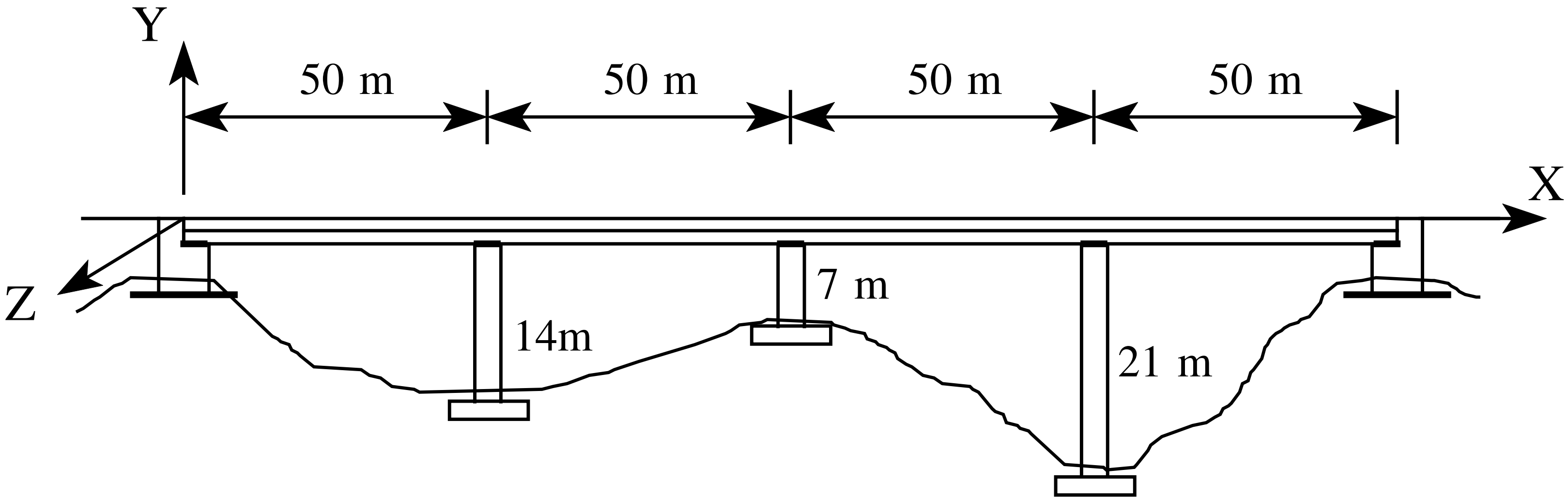


Example : Force-Displacement Loops for Lead-Plug Laminated Bearings



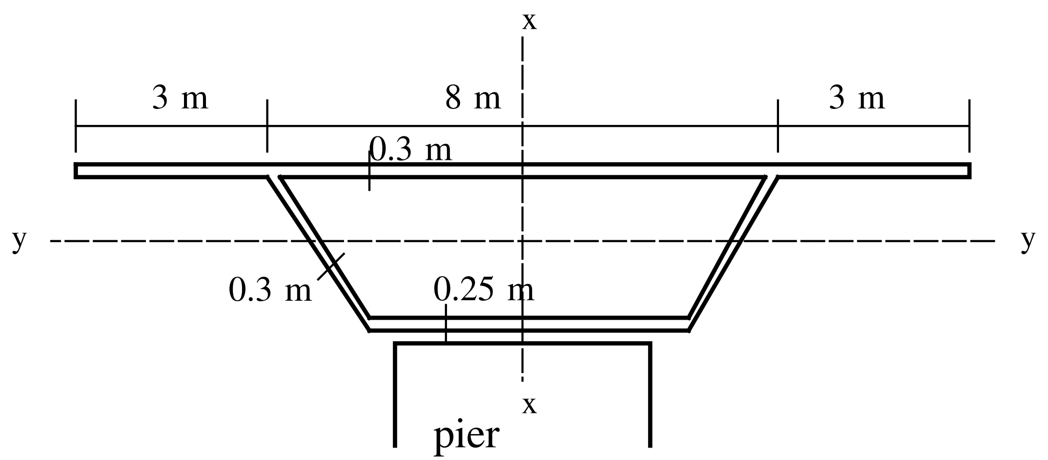
Example : Base-isolated Highway Bridge

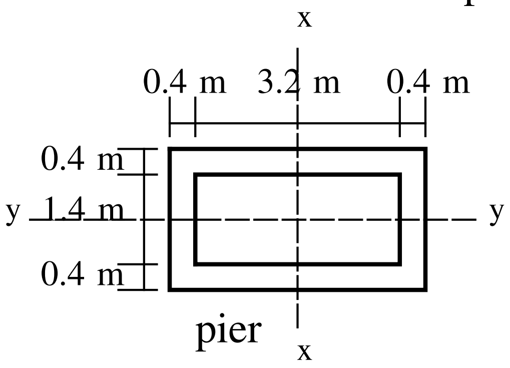
Elevation view of 4-span highway bridge



Example : Base-isolated Highway Bridge

Pier and deck section properties

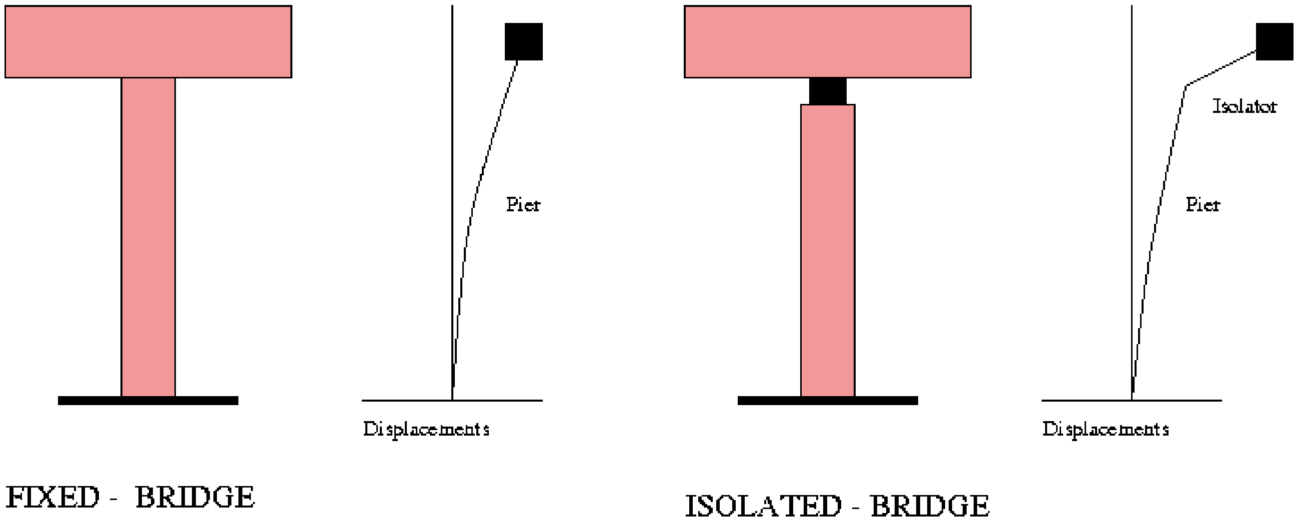




	deck	pier
area(m^2)	6.88	4.16
Ix(m^2)	87.24	7.39
Iy(m^2)	5.26	0.67
w($^{kN}/_{m}$)	200	104

Example : Base-isolated Highway Bridge

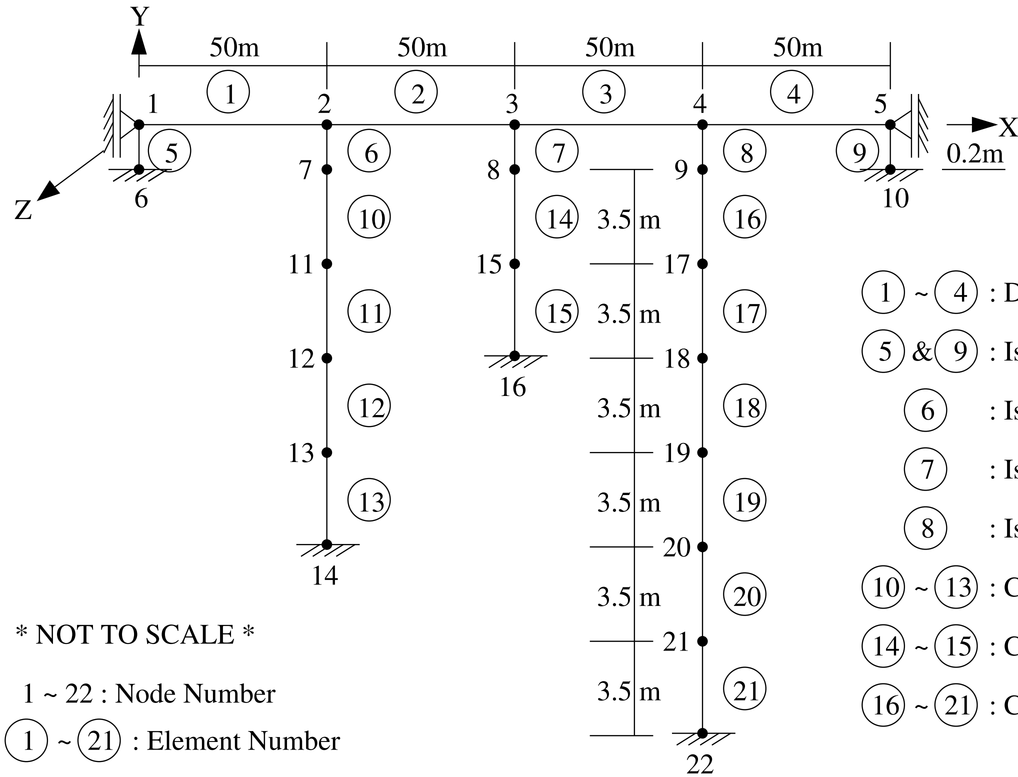
Principle of base isolation



INSTITUTE FOR SYSTEMS RESEARCH

Example : Base-isolated Highway Bridge

Finite element model



1 ~ 4 : Deck Attribute; `FRAME_3D`

5 & 9 : Isolator Attribute 1; `FIBER_3D`

6 : Isolaotr Attribute 2; `FIBER_3D`

7 : Isolator Attribute 3; `FIBER_3D`

8 : Isolator Attribute 4; `FIBER_3D`

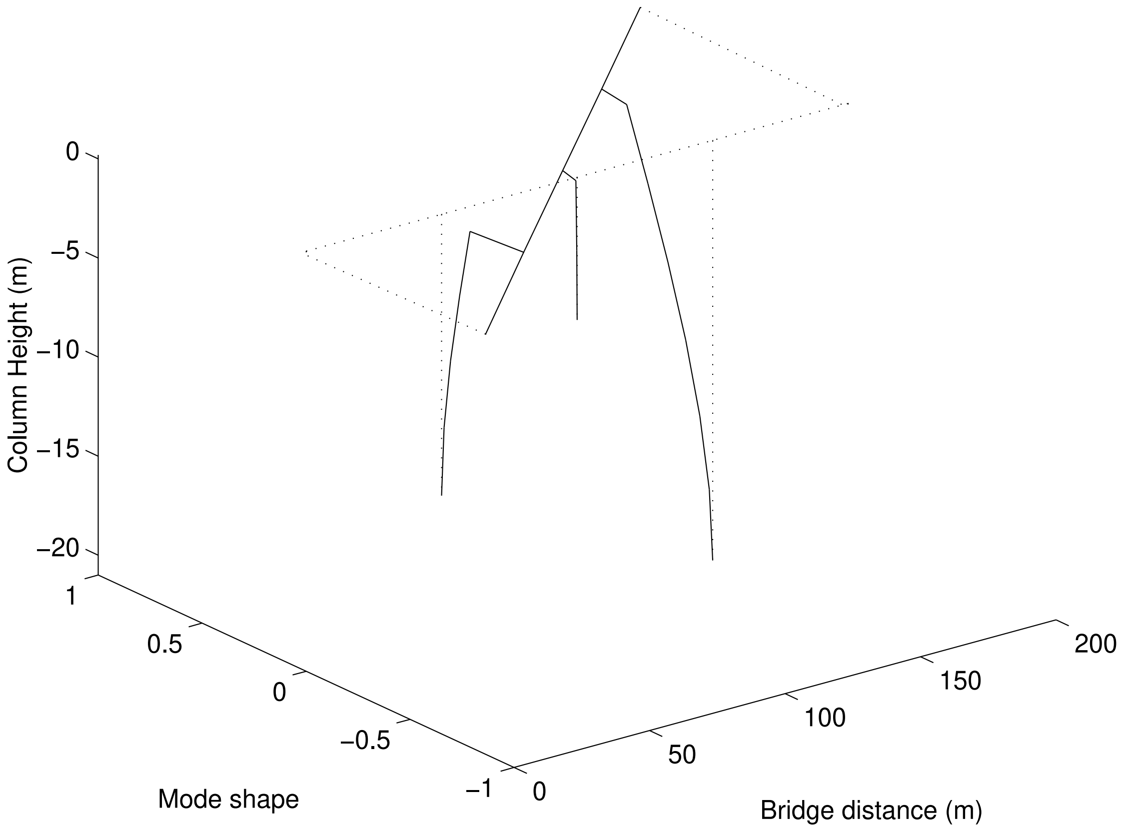
10 ~ 13 : Column Attribute 2; `FRAME_3D`

14 ~ 15 : Column Attribute 3; `FRAME_3D`

16 ~ 21 : Column Attribute 4; `FRAME_3D`

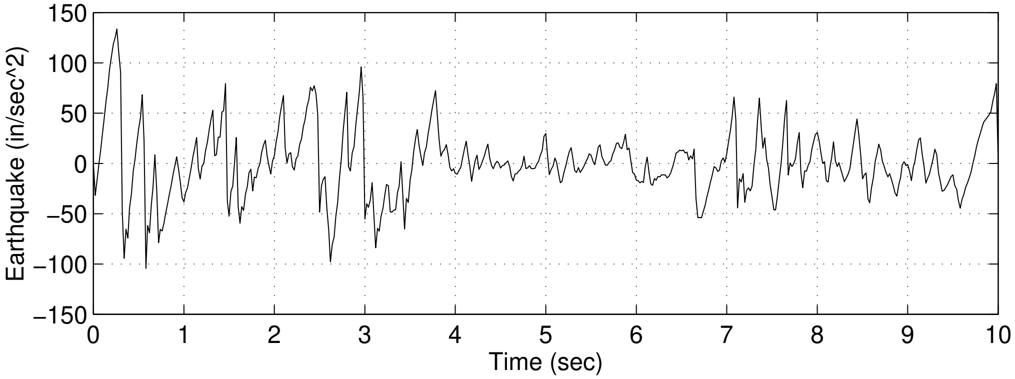
Example : Base-isolated Highway Bridge

Eigenvalue analysis : first mode response



Example : Base-isolated Highway Bridge

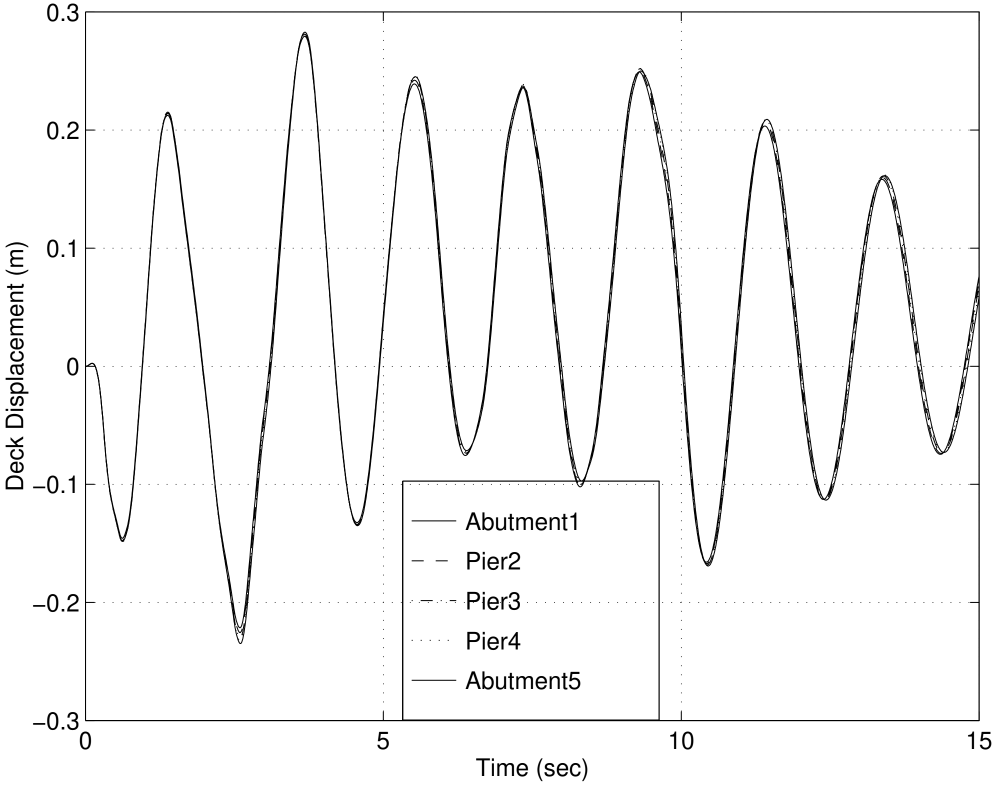
Earthquake ground acceleration record



1940 El Centro S00E record as the input earthquake with the peak ground acceleration scaled to 0.5g.

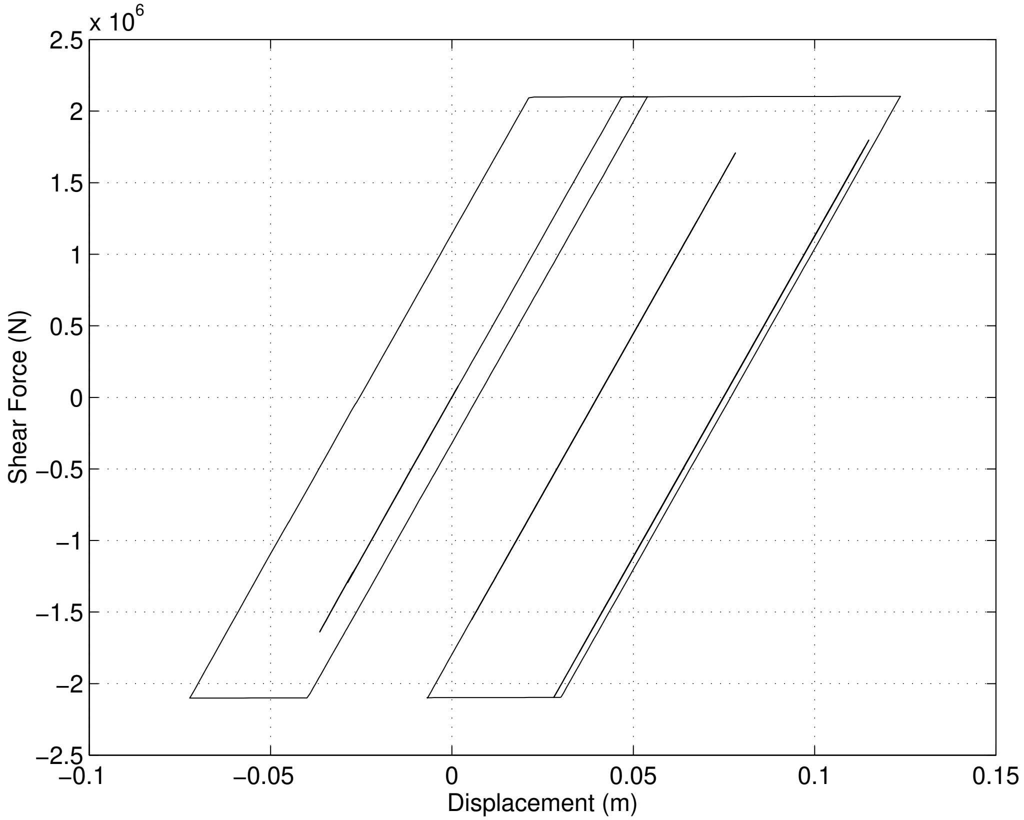
Example : Base-isolated Highway Bridge

Deck displacement of isolated bridge



Example : Base-isolated Highway Bridge

Force-displacement response of isolator at pier 4



Example : System Energy Evaluation

Energy balance calculations

Use energy balance calculations to check computations are stable and accurate. At the highest level of abstraction we can say that the system energy at time $(n+1)\Delta t$ should satisfy the equation:

$$W_{n+1}^{int} + T_{n+1} = W_{n+1}^{ext} \tag{3}$$

where W (without subscripts and superscripts) represents work done and T represents kinetic energy.

Physical interpretation

Equation (3) states that the work done by external loads is converted to kinetic energy, and to energy stored either elastically or dissipated by plastic deformations.

Example : System Energy Evaluation

Internal work : W_{n+1}^{int}

Work done by loads that are developed from straining of materials

$$W_{n+1}^{int} = W_n^{int} + \int_{n\Delta t}^{(n+1)\Delta t} \dot{W}^{int} dt. \tag{4}$$

Discrete approximation

The rate of internal work is given by

$$\dot{W}_n^{int} = \{\dot{\mathbf{D}}\}_n^T \{\mathbf{R}^{int}\}_n. \tag{5}$$

where $\{\dot{\mathbf{D}}\}_n$ represents the velocity vector at time step n, and $\{\mathbf{R}^{int}\}_n$ represents the resisting nodal loads at time step n. Hence,

$$W_{n+1}^{int} = W_n^{int} + \frac{\Delta t}{2}(\{\dot{\mathbf{D}}\}_n^T \{\mathbf{R}^{int}\}_n + \{\dot{\mathbf{D}}\}_{n+1}^T \{\mathbf{R}^{int}\}_{n+1}). \tag{6}$$

Example : System Energy Evaluation

External work : W_{n+1}^{ext}

The external work represents the work done by changes in externally applied loads moving through displacement of the system degrees of freedom. It is given by

$$W_{n+1}^{ext} = W_{n}^{ext} + \int_{n\Delta t}^{(n+1)\Delta t} \{\dot{\mathbf{D}}\}^T \{\mathbf{R}^{ext}\} dt. \tag{7}$$

A discrete approximation is

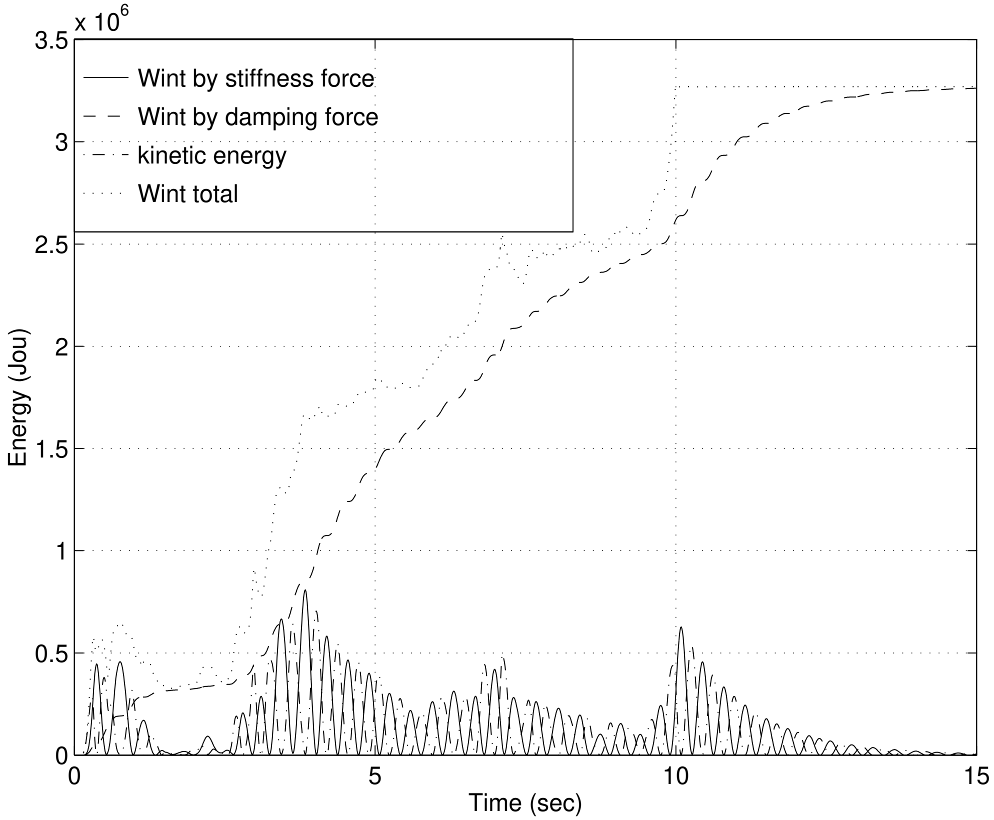
$$W_{n+1}^{ext} = W_{n}^{ext} + \frac{\Delta t}{2}(\{\dot{\mathbf{D}}\}_n^T \{\mathbf{R}^{ext}\}_n + \{\dot{\mathbf{D}}\}_{n+1}^T \{\mathbf{R}^{ext}\}_{n+1}). \tag{8}$$

Kinetic energy : T_{n+1}

$$T_{n+1} = \frac{1}{2}\{\dot{\mathbf{D}}\}_{n+1}^T [\mathbf{M}] \{\dot{\mathbf{D}}\}_{n+1}. \tag{9}$$

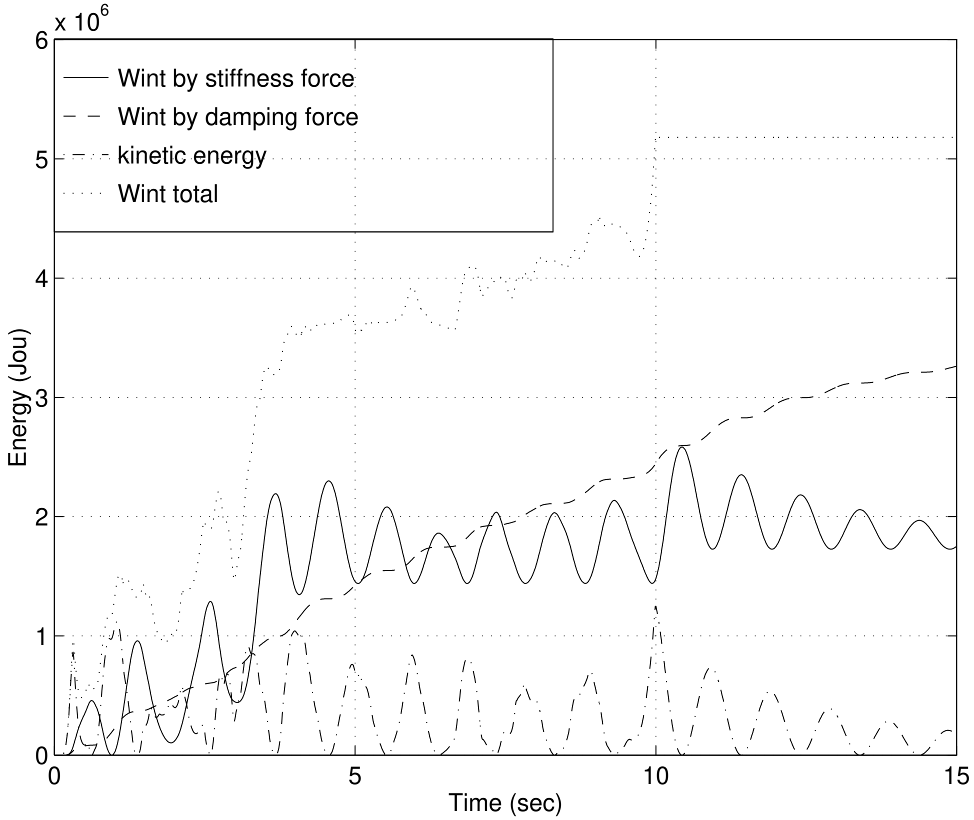
Example : Base-isolated Highway Bridge

Energy balance components in nonisolated bridge



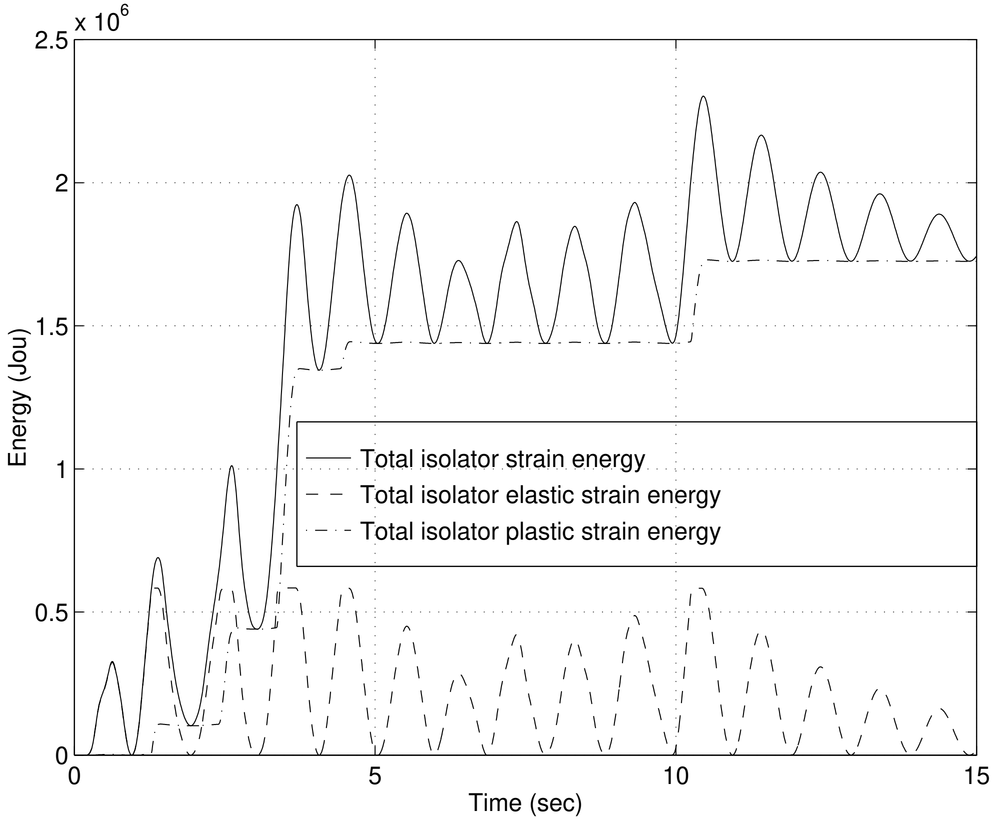
Example : Base-isolated Highway Bridge

Energy balance components in isolated bridge



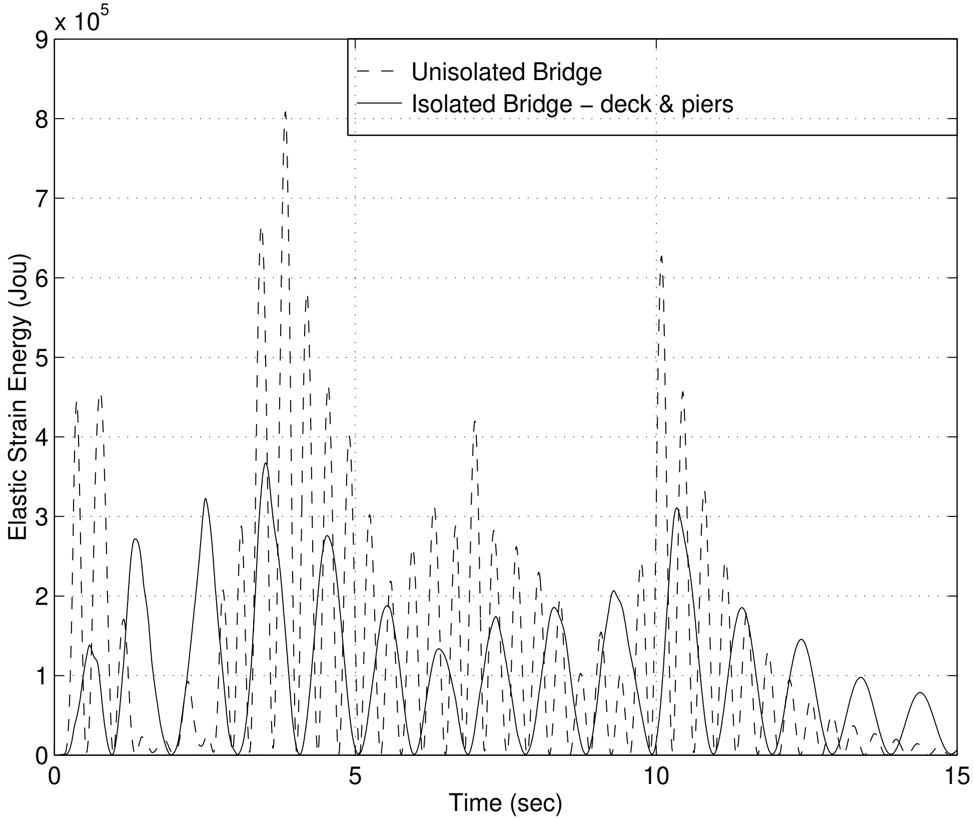
Example : Base-isolated Highway Bridge

Elastic and plastic strain energy of isolators



Example : Base-isolated Highway Bridge

Comparison of deck and pier strain energies



INSTITUTE FOR SYSTEMS RESEARCH

Part 5 : Conclusions and Future Work

Conclusions

- Aladdin (Version 1) is complete. Linear static and dynamic analysis of building and highway bridge structures. Examples can be found at:

  `http://www.isr.umd.edu/~austin/aladdin.html.`

- Aladdin (Version 2) will be released Fall semester 1998. Key application area is nonlinear time-history analysis of base-isolated highway bridge systems.

Future Work

- Suite of numerical algorithms for studies involving the hybrid control of large structural systems.
- **WebAladdin.** Aladdin operates on Web with server-side simulations and client-side graphical input/output in Netscape browser.