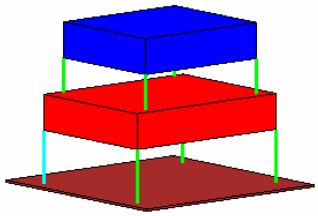
Application Demonstration

Maplesoft
www.maplesoft.com

3-D Oscillator with Two Masses, Coupled by Elastic Springs

www.gerb.com

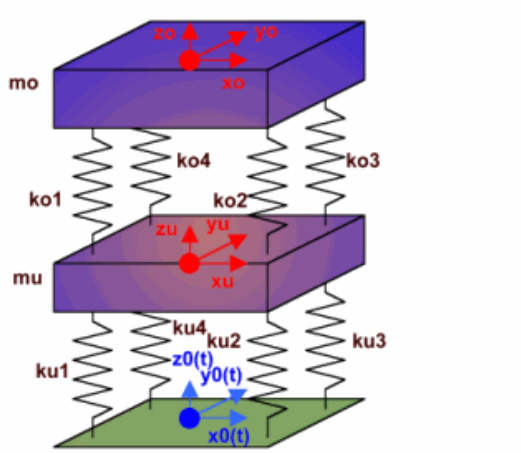


Introduction

This worksheet shows the calculation of the motion of a system with 12 degrees of freedom excited by ground motion.
The following Maple techniques are highlighted:

- Developing a general model
- Reading in specific data from a file
- Calculation of system matrices
- Creating a 3D animation of the system

Problem Description



Consider a system of two masses. Both masses are described by rectangular prisms. The support for the two masses are elastic. The ground motion is known and is stored in the file "motion.dat".
Note: The file "motion.dat" includes four columns. First the discrete time steps, second the values of the ground acceleration in x-direction, third in y-direction and fourth in z-direction.

Contents

1) In the first section of the worksheet the mechanical system is described. All data points are in m, kg, N and sec.

2) In the second section, the system matrices are calculated. The stiffness matrix is calculated according [1].

3) In section three, the eigenvalue problem is solved and the modal matrix is calculated.

4) In section four, the time history of the system movement is calculated numerically. The NEWMARK-algorithm is used to solve the incremental equations of motion [2].

5) In the last section, the solutions are displayed. First, the eigenvalues and eigenmodes are listed. Then the time history of the motion in every direction relative to the ground are plotted. Finally, the system motion is illustrated in the form of a small animation (which is duplicated at the top of this worksheet).

The lower mass and the lower spring in general are described by the index "u", the upper ones by the index "o".
The coordinates 1, 2 and 3 are the translation of the lower mass in x-, y- and z-direction, 4, 5 and 6 the corresponding rotations. The coordinates 7, 8 and 9 are the translations of the upper mass, 10, 11 and 12 the rotations.

Description of the Mechanical Model

Dimensions of the Bodies

x-direction: lu; lo [m]
y-direction: bu; bo [m]
z-direction: hu; ho [m]
mass: mu,mo [kg]

lower mass

```
> lu:=10.0:bu:=8.0:hu:=2.0:
> mu:=1000000:
```

End points of the diagonal

```
> ENDu1:=[-lu/2,-bu/2,-hu/2]:
> ENDu2:=[lu/2,bu/2,hu/2]:
```

upper mass

```
> lo:=8.0:bo:=7.0:ho:=2.0:
> mo:=100000:
```

End points of the diagonal

```
> ENDo1:=[-lo/2,-bo/2,-ho/2]:
> ENDo2:=[lo/2,bo/2,ho/2]:
```

distance between the centers of gravity

this is necessary to know for the animation later

```
> DSuo:=[0,0,ho+hu]:
```

General Spring Characteristics

The general characteristics in the horizontal and vertical direction for the lower and the upper springs are given.
This general data can be modified in the following subsection for each individual spring.

coh/cuh: stiffness of the upper/lower spring in horizontal direction [N/m]
cov/cuv: stiffness of the upper/lower spring in vertical direction [N/m]

lower springs

```
> cuh:=5.0*10**6: cuv:=5.0*10**6:
```

upper springs

```
> coh:=1.0*10**6: cov:=1.0*10**6:
```

Position of the Bodies Supporting Springs and their Individual Characteristics

all positions in metres (m), the stiffness of the individual springs can be manipulated in the following (see e.g. 1. spring of lower mass)

lower mass

a[i, j]: position of the lower spring i in direction j
cu[i, j]: stiffness of the lower spring i in direction j

Spring 1
Note: The stiffness are varied from the general stiffness defined previously.

```
> a[1,x] := -lu/2: a[1,y] := -bu/2: a[1,z]:=-hu/2:
  cu[1,x] := cuh*0.9: cu[1,y] := cuh*0.85: cu[1,z] :=
   cuv*0.8:
```

Spring 2
Notice that we assume the stiffness (in the z direction) of this spring is significantly different from that of the others. This could be caused by damage to the spring.

```
> a[2,x] := lu/2: a[2,y] := -bu/2: a[2,z]:=-hu/2:
  cu[2,x] := cuh: cu[2,y] := cuh: cu[2,z] := cuv/10:
```

Spring 3

```
> a[3,x] := lu/2: a[3,y] := bu/2: a[3,z]:=-hu/2:
  cu[3,x] := cuh: cu[3,y] := cuh: cu[3,z] := cuv:
```

Spring 4

```
> a[4,x] := -lu/2: a[4,y] := bu/2: a[4,z]:=-hu/2:
  cu[4,x] := cuh: cu[4,y] := cuh: cu[4,z] := cuv:
```

Overall Parameters
Here, we will display the two sets of parameters in matrix form

```
> a = Matrix( [seq( [seq(a[j,i],i in [x,y,z])], j=1..4 )]
  );
```

$$a = \begin{bmatrix} -5.000000000 & -4.000000000 & -1.000000000 \\ 5.000000000 & -4.000000000 & -1.000000000 \\ 5.000000000 & 4.000000000 & -1.000000000 \\ -5.000000000 & 4.000000000 & -1.000000000 \end{bmatrix}$$

```
> cu = Matrix( [seq( [seq(cu[j,i],i in [x,y,z])], j=1..4 )]
  );
```

$$cu = \begin{bmatrix} 0.450000000\,10^7 & 0.4250000000\,10^7 & 0.400000000\,10^7 \\ 0.50000000\,10^7 & 0.50000000\,10^7 & 500000.0000 \\ 0.50000000\,10^7 & 0.50000000\,10^7 & 0.50000000\,10^7 \\ 0.50000000\,10^7 & 0.50000000\,10^7 & 0.50000000\,10^7 \end{bmatrix}$$

upper mass

b[i, j]: position of the upper spring i in direction j
co[i, j]: position of the upper spring i in direction j
bzqlli: z position of the lower mass support

Spring 1

Notice that the upper springs here need additional information on the position of the lower mass support.

```
> b[1,x] := -lo/2: b[1,y] := -bo/2: b[1,z] := -ho/2:
  bzq1:=hu/2:
  co[1,x] := coh: co[1,y] := coh: co[1,z] := cov:
```

Spring 2

```
> b[2,x] := lo/2: b[2,y] := -bo/2: b[2,z] := -ho/2:
  bzq2:=hu/2:
  co[2,x] := coh: co[2,y] := coh: co[2,z] := cov:
```

Spring 3

```
> b[3,x] := lo/2: b[3,y] := bo/2: b[3,z] := -ho/2:
  bzq3:=hu/2:
  co[3,x] := coh: co[3,y] := coh: co[3,z] := cov:
```

Spring 4

```
> b[4,x] := -lo/2: b[4,y] := bo/2: b[4,z] := -ho/2:
  bzq4:=hu/2:
  co[4,x] := coh: co[4,y] := coh: co[4,z] := cov:
```

Overall Parameters

Here, we will display the two sets of parameters in matrix form

```
> b = Matrix( [seq( [seq(b[j,i],i in [x,y,z])], j=1..4 )]
  );
```

$$b = \begin{bmatrix} -4.000000000 & -3.500000000 & -1.000000000 \\ 4.000000000 & -3.500000000 & -1.000000000 \\ 4.000000000 & 3.500000000 & -1.000000000 \\ -4.000000000 & 3.500000000 & -1.000000000 \end{bmatrix}$$

```
> co = Matrix( [seq( [seq(co[j,i],i in [x,y,z])], j=1..4 )]
  );
```

$$co = \begin{bmatrix} 0.10000000\,10^{7} & 0.10000000\,10^{7} & 0.10000000\,10^{7} \\ 0.10000000\,10^{7} & 0.10000000\,10^{7} & 0.10000000\,10^{7} \\ 0.10000000\,10^{7} & 0.10000000\,10^{7} & 0.10000000\,10^{7} \\ 0.10000000\,10^{7} & 0.10000000\,10^{7} & 0.10000000\,10^{7} \end{bmatrix}$$

Calculate the System Matrices

System Stiffness

deformation of the lower springs

Here, we describe the deformation of the lower springs in terms of the position and orientation of the lower body.

q1, q2,...,q6 are the degrees of freedom (q1, q2, q3 being the linear motion in x, y, z direction respectively and q4, q5 and q6 being the rotation about the x, y, z axis) of the lower body.

u1, u2 and u3 are the spring displacement in the x, y and z direction respectively.

```
> u1:=q1 +akz*q5 -aky*q6:
  u2:=q2 -akz*q4 +akx*q6:
  u3:=q3 +aky*q4 -akx*q5:
> uq1:=expand(simplify(u1^2));
  uq2:=expand(simplify(u2^2));
  uq3:=expand(simplify(u3^2));
```

$$uq1 := q1^2 + 2\,q1\,akz\,q5 - 2\,q1\,aky\,q6 + akz^2\,q5^2 - 2\,akz\,q5\,aky\,q6 + aky^2\,q6^2$$

$$uq2 := q2^2 - 2\,q2\,akz\,q4 + 2\,q2\,akx\,q6 + akz^2\,q4^2 - 2\,akz\,q4\,akx\,q6 + akx^2\,q6^2$$

$$uq3 := q3^2 + 2\,q3\,aky\,q4 - 2\,q3\,akx\,q5 + aky^2\,q4^2 - 2\,aky\,q4\,akx\,q5 + akx^2\,q5^2$$

deformation of the upper springs

For the upper springs, the deformation is described in terms of the position and orientation of the upper and lower body.

p1, p2,...,p6 are the degrees of freedom of the upper body (similar to the q defined for the lower body).

o1, o2 and o3 are the spring displacement in the x, y, and z direction respectively.

```
> o1:=-q1 -bqkz*q5 +bky*q6 +p1 +bkz*p5 -bky*p6:
  o2:=-q2 +bqkz*q4 -bkx*q6 +p2 -bkz*p4 +bkx*p6:
  o3:=-q3 -bky*q4 +bkx*q5 +p3 +bky*p4 -bkx*p5:
> oq1:=expand(simplify(o1^2));
  oq2:=expand(simplify(o2^2));
  oq3:=expand(simplify(o3^2));
```

$$\begin{aligned} oq1 := {} & 2\,bky\,q6\,p1 + 2\,q1\,bqkz\,q5 - 2\,q1\,bkz\,p5 - 2\,p1\,bky\,p6 - 2\,bky^2\,q6\,p6 - 2\,q1\,bky\,q6 \\ & + 2\,q1\,bky\,p6 - 2\,bqkz\,q5\,p1 + 2\,p1\,bkz\,p5 - 2\,bqkz\,q5\,bky\,q6 + bky^2\,p6^2 + bkz^2\,p5^2 \\ & + p1^2 + bky^2\,q6^2 + bqkz^2\,q5^2 - 2\,q1\,p1 - 2\,bqkz\,q5\,bkz\,p5 + q1^2 - 2\,bkz\,p5\,bky\,p6 \\ & + 2\,bqkz\,q5\,bky\,p6 + 2\,bky\,q6\,bkz\,p5 \end{aligned}$$

$$\begin{aligned} oq2 := {} & 2\,q2\,bkz\,p4 - 2\,q2\,bkx\,p6 - 2\,bqkz\,q4\,bkx\,q6 + 2\,bqkz\,q4\,p2 - 2\,bkx\,q6\,p2 \\ & + 2\,p2\,bkx\,p6 + 2\,q2\,bkx\,q6 - 2\,bkx^2\,q6\,p6 - 2\,q2\,bqkz\,q4 - 2\,p2\,bkz\,p4 + bkx^2\,p6^2 \\ & + bkz^2\,p4^2 + p2^2 + bkx^2\,q6^2 + q2^2 + bqkz^2\,q4^2 - 2\,q2\,p2 - 2\,bkz\,p4\,bkx\,p6 \\ & + 2\,bkx\,q6\,bkz\,p4 - 2\,bqkz\,q4\,bkz\,p4 + 2\,bqkz\,q4\,bkx\,p6 \end{aligned}$$

$$\begin{aligned} oq3 := {} & 2\,q3\,bkx\,p5 - 2\,p3\,bkx\,p5 - 2\,q3\,bky\,p4 + 2\,bkx\,q5\,p3 - 2\,q3\,bkx\,q5 - 2\,bkx^2\,q5\,p5 \\ & + 2\,q3\,bky\,q4 + 2\,p3\,bky\,p4 - 2\,bky\,q4\,p3 - 2\,bky^2\,q4\,p4 + bkx^2\,p5^2 + bky^2\,p4^2 + p3^2 \\ & + bkx^2\,q5^2 + bky^2\,q4^2 + q3^2 - 2\,q3\,p3 - 2\,bky\,q4\,bkx\,q5 + 2\,bky\,q4\,bkx\,p5 \\ & + 2\,bkx\,q5\,bky\,p4 - 2\,bky\,p4\,bkx\,p5 \end{aligned}$$

general form of stiffness matrix for individual spring directions

Using the above deformation equations, we can extract the general form of the spring stiffness components for each direction from the potential energy according to [1]

cu1, cu2, cu3: 12 x 12 stiffness matrices for the lower springs in x, y, and z direction respectively.
co1, co2, co3: 12 x 12 stiffness matrices for the upper springs in x, y, and z direction respectively.

```
> xlist:=[seq(q||i,i=1..6),seq(p||i,i=1..6)]:
  x2list:= [seq(xlist[i]^2,i=1..nops(xlist))]:
> for k from 1 to 3 do
     cu||k := evalm( DiagonalMatrix( [ seq( coeff( uq||k,
          x2list[i] ), i=1..12 ) ] ) +
          (Matrix( [ seq( [seq( coeff( coeff( uq||k,
          xlist[i] ), xlist[j] ), j=1..12 )],
          i=1..12 )] ) )/2 ):
     co||k := evalm( DiagonalMatrix( [ seq( coeff( oq||k,
          x2list[i] ), i=1..12 ) ] ) +
          (Matrix( [ seq( [seq( coeff( coeff( oq||k,
          xlist[i] ), xlist[j] ), j=1..12 )],
          i=1..12 ) ] ) )/2 ):
  end do:
```

stiffness matrix for every spring in every direction

With the above general form of the stiffness matrices in the x, y, and z direction, we can generate the stiffness matrix associated with each spring by substituting the parameters corresponding to each of the springs.

The follow index notion are used for the resulting table of stiffness matrices:

first index: 1, 2, 3 and 4for lower springs; 5, 6, 7 and 8 for upper springs,
second index: for direction x, y and z

```
> for s_index from 1 to 4 do
     n := 1;
     for d_index in [x,y,z] do
          k[s_index,d_index] := cu[s_index,d_index].
               eval( cu||n, {
                    seq( ak||i=a[s_index,i], i in {x,y,z}
                    ),
                    seq( bk||i=b[s_index,i], i in {x,y,z}),
                    bqkz=bzq||s_index } ):
          k[s_index+4,d_index] := co[s_index,d_index].
               eval( co||n,
                    {seq(ak||i=a[s_index,i], i in {x,y,z}),
                     seq(bk||i=b[s_index,i], i in {x,y,z}),
               bqkz=bzq||s_index } ):
          n := n+1;
     end do:
end do:
```

overall stiffness matrix

Finally, to get the overall system stiffness matrix, we add up all of the individual stiffness matrices.

```
> K := Matrix( 12, 12, 0 ):
  for i from 1 to 8 do
     K := evalm(K + evalm(k[i,x]+k[i,y]+k[i,z]));
  end do:
  K := convert( K, Matrix );
```

$$K := \begin{bmatrix} 12 \text{ x } 12 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

System Mass

matrix of one body

To obtain the system mass matrix, we first define the mass matrix for one body.

```
> Mr := DiagonalMatrix( [m, m, m, m/12*(b^2+h^2),
  m/12*(l^2+h^2), m/12*(l^2+b^2)] );
```

$$Mr := \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{m\,(b^2+h^2)}{12} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{m\,(l^2+h^2)}{12} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{m\,(l^2+b^2)}{12} \end{bmatrix}$$

general mass matrix

Then we combine them to get the overall system mass matrix.

```
> M1:=evalm(subs({m=mu,l=lu,b=bu,h=hu},evalm(Mr))):
> M2:=evalm(subs({m=mo,l=lo,b=bo,h=ho},evalm(Mr))):
> M:=DiagonalMatrix( [M1, M2] );
```

$$M := \begin{bmatrix} 12 \text{ x } 12 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

System Matrix

Finally, the system matrix is the product of the inverse of the mass matrix and the system stiffness matrix.

```
> SM:=convert(evalm(MatrixInverse(M).K), Matrix);
```

$$SM := \begin{bmatrix} 12 \text{ x } 12 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

Eigenvalue Problem Solution

With the system matrix, we can now look at the eigenvalue problem.

Eigenvalues and Eigenmodes

First, we compute the eigenvalues and eigenvectors of the system matrix using the Eigenvectors procedure from the LinearAlgebra package.

```
> lambda, vecs := Eigenvectors(SM):
```

Here, we split the eigenvector matrix into individual vectors so that we can sort them based on the associated eigenvalues.

```
> for i from 1 by 1 to 12 do
     v||i := SubMatrix(vecs, [1..12], [i]);
  end do:
```

Sorting with respect to eigenvalues:

```
> lstsrt := sort( [ seq( lambda[i], i=1..12 ) ] );
> for i from 1 by 1 to 12 do
     for j from 1 by 1 to 12 do
          if (lstsrt[i]=lambda[j]) then
               num[i]:=j:
               if (i>1 and num[i]<>num[i-1]) then j:=12 fi:
          fi:
     end do:
  end do:
> for i from 1 by 1 to 12 do
     ev||i:=(v||(num[i]));
  end do:
> ew:=Vector([ seq( lstsrt[i],i=1..12) ]);
```

$$lstsrt := [132.144825771004605 + 0.\,I,\ 134.815111916210640 + 0.\,I,\ 134.527231153041584 + 0.\,I,\ 7.45598549215869788 + 0.\,I,\ 14.6423967805512892 + 0.\,I,\ 15.1581801399133287 + 0.\,I,\ 34.3724577704070456 + 0.\,I,\ 39.3352390695982735 + 0.\,I,\ 58.1571810529707989 + 0.\,I,\ 44.7916127478653650 + 0.\,I,\ 49.9526067329783246 + 0.\,I,\ 51.8224003533006866 + 0.\,I]$$

$$ew := \begin{bmatrix} 12 \text{ Element Column Vector} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

Eigenfrequencies

We can also extract the eigenfrequencies from the eigenvalues.

```
> omega := [ seq( sqrt( ew[i] ), i=1..12 ) ]:
> f := [ seq( evalf(omega[i]/(2*Pi)), i=1..12 )]:
```

Modal Matrix

We obtain the model matrix by normalizing the eigenvectors.

```
> for i from 1 to 12 do
     phi||i := Normalize( convert(ev||i[1..12, 1],
          Vector) );
  end do:
  phi := Matrix( [ seq( phi||i, i=1..12 ) ] );
> phit:=Transpose(phi);
```

$$\phi := \left[\begin{array}{l} 12 \text{ x } 12 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{array}\right]$$

$$phit := \left[\begin{array}{l} 12 \text{ x } 12 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{array}\right]$$

Modal Mass and Modal Stiffness

```
> Ms:=phit.M.phi:
  Ms := DiagonalMatrix( [ seq( Ms[i,i], i=1..12 ) ] );
> Ks:=phit.K.phi:
  Ks := DiagonalMatrix( [ seq( Ks[i,i], i=1..12 ) ] );
```

$$Ms := \left[\begin{array}{l} 12 \text{ x } 12 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{array}\right]$$

$$Ks := \left[\begin{array}{l} 12 \text{ x } 12 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{array}\right]$$

Numeric Solution of the Equation of Motion

Here, we will obtain the numeric solution to the equation of motion using the NEWMARK algorithm [2]

Time-History of the Ground Motion

Reading in data from file

First, we will read in the ground motion from the data file motion.dat.

Recall that the data is listed in 4 columns:
1. time, 2. x-acceleration, 3. y-acceleration, 4. z-acceleration

```
> loaddat:=readdata("motion.dat",4):
```

Extract the time step from the data

```
> dt:=loaddat[4][1]-loaddat[3][1];
```

$$dt := 0.01$$

Getting the number of samples

```
> ndt:=nops(loaddat);
```

$$ndt := 1500$$

Setting the total time

```
> maxT:=(ndt-1)*dt;
```

$$maxT := 14.99$$

Setting up the discrete time variables for use later.

```
> for i from 0 by 1 to ndt do t||i:=i*dt end do:
```

Assigning the direction data into separate variables.

```
> a0x := [seq( loaddat[i][2], i=1..ndt)]:
  a0y := [seq( loaddat[i][3], i=1..ndt)]:
  a0z := [seq( loaddat[i][4], i=1..ndt)]:
```

For the animation later, the ground acceleration is numerically integrated twice to get the ground displacement. For the numeric integration, it is assumed that the acceleration between two discrete time steps is linear. Additionally, we assume that the velocities and the displacement at the beginning of the time period is zero.

```
> v0x := Vector[column]( ndt, 0 ):
  v0y := Vector[column]( ndt, 0 ):
  v0z := Vector[column]( ndt, 0 ):
> x0x := Vector[column]( ndt, 0 ):
  x0y := Vector[column]( ndt, 0 ):
  x0z := Vector[column]( ndt, 0 ):
> for j from 2 by 1 to ndt do
     for i in [x,y,z] do
          v0||i[j]:=v0||i[j-1]+(a0||i[j]+a0||i[j-1])*dt/2;
          x0||i[j]:=x0||i[j-1]+(v0||i[j]+v0||i[j-1])*dt/2;
     end do:
  end do:
```

Plotting the input data

Setting up the line colour.

```
> clax:=red:clay:=blue:claz:=green:
```

Ground Acceleration

x - direction

```
> SSax:=[seq([t||j,a0x[j]],j=1..ndt)]:
  Pax:=curve(SSax,color=clax):
```

y - direction

```
> SSay:=[seq([t||j,a0y[j]],j=1..ndt)]:
  Pay:=curve(SSay,color=clay):
```

z-direction

```
> SSaz:=[seq([t||j,a0z[j]],j=1..ndt)]:
  Paz:=curve(SSaz,color=claz):
```

Ground Displacement

x - direction

```
> SSdx:=[seq([t||j,x0x[j]],j=1..ndt)]:
  Pdx:=curve(SSdx,color=clax):
```

y - direction

```
> SSdy:=[seq([t||j,x0y[j]],j=1..ndt)]:
  Pdy:=curve(SSdy,color=clay):
```

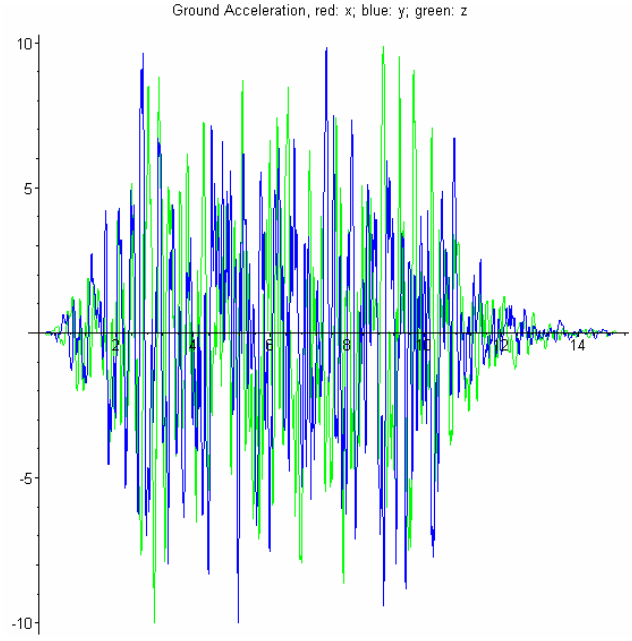
z - direction

```
> SSdz:=[seq([t||j,x0z[j]],j=1..ndt)]:
  Pdz:=curve(SSdz,color=claz):
```

Plots

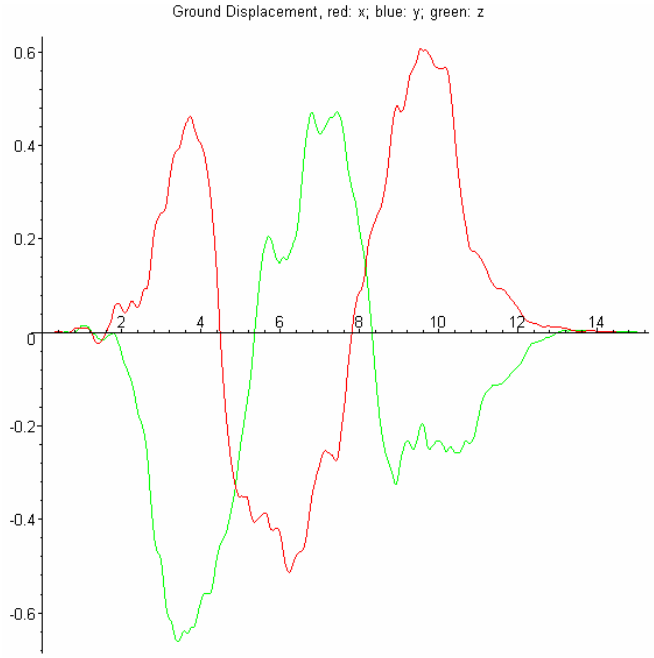
Accelerations

```
> display({Pax,Pay,Paz},title="Ground Acceleration, red: x;
  blue: y; green: z");
```



Displacement

```
> display({Pdx,Pdy,Pdz},title="Ground Displacement, red: x;
  blue: y; green: z");
```



Computing the ground forces from the ground motion

We will transform the ground motion into the substitute force, assuming no ground rotations, using $F = m\,a$.

```
> for i from 1 by 1 to ndt do
     a0vec||i := Vector(12, 0);
     a0vec||i[1]:=a0x[i];a0vec||i[7]:=a0x[i];
     a0vec||i[2]:=a0y[i];a0vec||i[8]:=a0y[i];
     a0vec||i[3]:=a0z[i];a0vec||i[9]:=a0z[i];
  end do:
> for i from 1 by 1 to ndt do
     ||i:=-(M.a0vec||i);
  end do:
```

Transform the Substitute Force into Modal Coordinates

```
> for i from 1 by 1 to ndt do
     Fs||i:=phit.F||i;
  end do:
```

NEWMARK-Algorithm

Now, apply the NEWMARK algorithm

First, setup the NEWMARK-constants

```
> alpha:=0.25:delta:=0.5:
```

Then, we compute the modified mass matrix

```
> Mn:=Ms+(Ks*alpha*dt^2);
```

$$Mn := \begin{bmatrix} 12 \text{ x } 12 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix} + 0.000025 \begin{bmatrix} 12 \text{ x } 12 \text{ Matrix} \\ \text{Data Type: anything} \\ \text{Storage: rectangular} \\ \text{Order: Fortran\_order} \end{bmatrix}$$

Next initialize the vectors.

```
> RS||1 := Vector[column]( 12, 0 ):
  Q||1 := Vector[column]( 12, 0 ):
  Qp||1 := Vector[column]( 12, 0 ):
  Qpp||1 := Vector[column]( 12, 0 ):
```

Then compute the output one step at a time.

```
> for i from 2 by 1 to ndt do
     RS||i := Vector[column]( [ seq( Fs||i[j] -
          Ks[ j, j ] * ( Q||(i-1)[j] +
          Qp||(i-1)[j]*dt +
          Qpp||(i-1)[j]*(1/2-alpha)*dt^2 ),
          j=1..12 ) ] ):
     Qpp||i := Vector[column]( [ seq( RS||i[j]/Mn[j,j],
          j=1..12 ) ] ):
     Qp||i := Vector[column]( [ seq( Qp||(i-1)[j] +
          dt*( Qpp||(i-1)[j]*(1-delta) + Qpp||i[j]*delta ),
          j=1..12 ) ] ):
     Q||i := Vector[column]( [ seq( Q||(i-1)[j] +
          dt*Qp||(i-1)[j] +
          dt^2*( Qpp||(i-1)[j]*(1/2-alpha) +
          Qpp||i[j]*alpha ),
          j=1..12 ) ] ):
  end do:
```

Finally, we can transform the solution into back real coordinates

```
> for i from 1 by 1 to ndt do
   X||i := map( Re, (phi.Q||i) );
end do:
```

Viewing the Solution

Graphical View of the Time History of the Motion

Translations

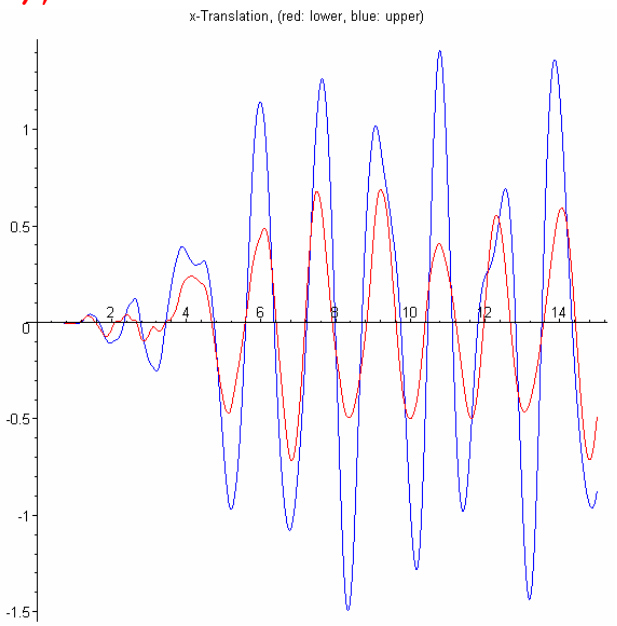
x - direction (relative to ground motion)

Setup the data points.

```
> SSXu:=[seq([t||j,(X||j[1])],j=1..ndt)]:
  SSXo:=[seq([t||j,(X||j[7])],j=1..ndt)]:
```

Setup the plot.

```
> PXu:=curve(SSXu,color=red):
  PXo:=curve(SSXo,color=blue):
> display({PXu,PXo},title="x-Translation, (red: lower,
  blue: upper)");
```



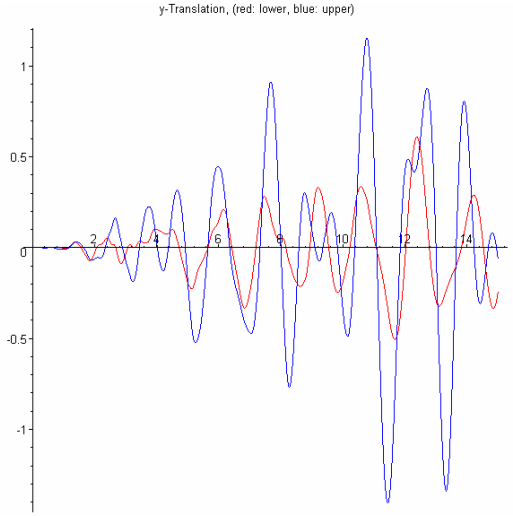
y - direction (relative to ground motion)

Setup the data points.

```
> SSYu:=[seq([t||j,(X||j[2])],j=1..ndt)]:
  SSYo:=[seq([t||j,(X||j[8])],j=1..ndt)]:
```

Setup the plot.

```
> PYu:=curve(SSYu,color=red):
 PYo:=curve(SSYo,color=blue):
> display({PYu,PYo},title="y-Translation, (red: lower,
  blue: upper)");
```



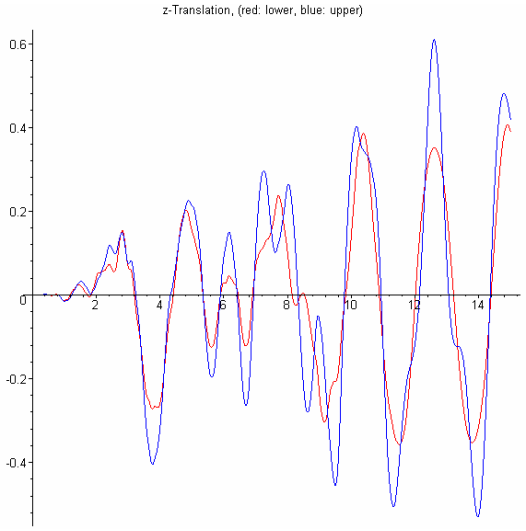
z -direction (relative to ground motion)

Setup the data points.

```
> SSZu:=[seq([t||j,(X||j[3])],j=1..ndt)]:
  SSZo:=[seq([t||j,(X||j[9])],j=1..ndt)]:
```

Setup the plot.

```
> PZu:=curve(SSZu,color=red):
  PZo:=curve(SSZo,color=blue):
> display({PZu,PZo},title="z-Translation, (red: lower,
  blue: upper)");
```



Rotations

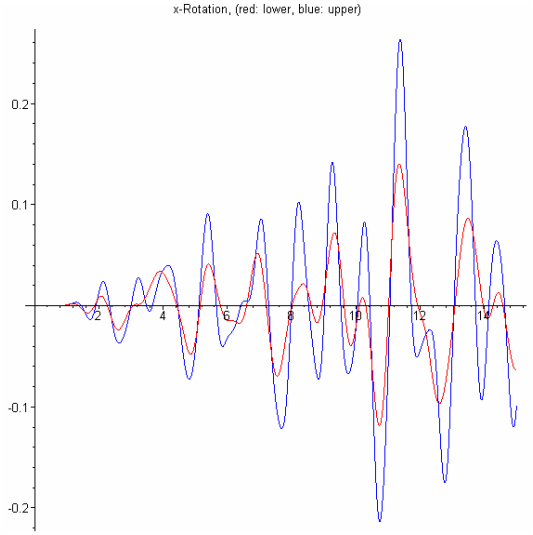
Rotation about x

Setup data points.

```
> SRXu:=[seq([t||j,(X||j[4])],j=1..ndt)]:
  SRXo:=[seq([t||j,(X||j[10])],j=1..ndt)]:
```

Setup the plot.

```
> RXu:=curve(SRXu,color=red):
  RXo:=curve(SRXo,color=blue):
> display({RXu,RXo},title="x-Rotation, (red: lower, blue:
  upper)");
```



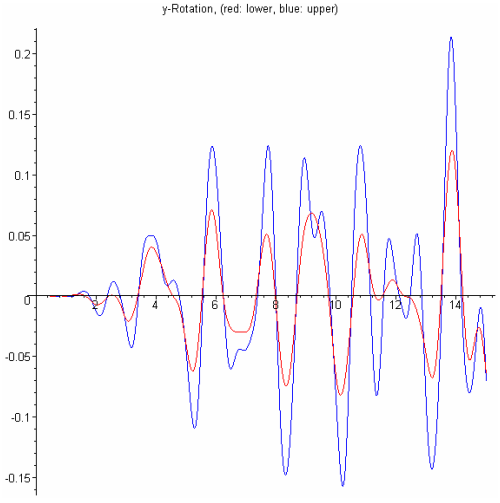
Rotation about y

Setup data points.

```
> SRYu:=[seq([t||j,(X||j[5])],j=1..ndt)]:
  SRYo:=[seq([t||j,(X||j[11])],j=1..ndt)]:
```

Setup the plot.

```
> RYu:=curve(SRYu,color=red):
  RYo:=curve(SRYo,color=blue):
> display({RYu,RYo},title="y-Rotation, (red: lower, blue:
  upper)");
```



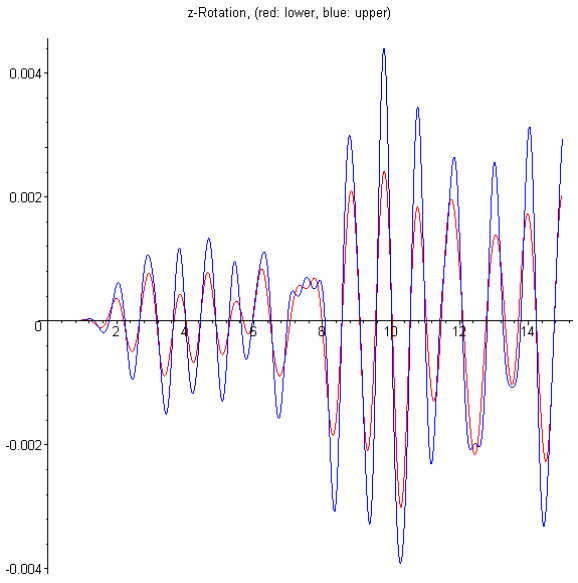
Rotation about z

Setup data points.

```
> SRZu:=[seq([t||j,(X||j[6])],j=1..ndt)]:
  SRZo:=[seq([t||j,(X||j[12])],j=1..ndt)]:
```

Setup the plot.

```
> RZu:=curve(SRZu,color=red):
  RZo:=curve(SRZo,color=blue):
> display({RZu,RZo},title="z-Rotation, (red: lower, blue:
  upper)");
```



Animation

To reduce execution time, only some time steps are used for the animation. If you have a fast machine, you can increase this number.

```
> nmx:=50:
```

Setting the number of plots used for the animation.

```
> if (ndt>nmx) then nplot:=nmx else nplot:=ndt fi:
```

Animation stepsize

```
> ds:=round(ndt/nplot);
```

$$ds := 30$$

Maximal motion

```
> maxm := max( seq( seq( (X||j[i]),i=1..12 ), j=1..ndt ) );
```

$$maxm := 1.40863654064256583$$

Calculate the normalization factor of the motion for the animation (using the Maximal motion magnitude).

```
> nor:=DSuo[3]/(maxm)/10:
```

Setting up the basic elements

Nominal ground plate location

```
> GROUND := translate( display( cuboid( [ (max(lu,lo)/2+1),
     -(max(lu,lo)/2+1), -(max(lu,lo)/2+1)/100 ],
     [ (max(lu,lo)/2+1), (max(lu,lo)/2+1),
     (max(lu,lo)/2+1)/100 ],
                                color=brown)),0,0,-DSuo[3]):
```

Nominal Diagonal Points for each prism at each animation time

```
> iplot:=0:
> for j from 1 by ds to ndt do
     # Animation index
     iplot:=iplot+1:
     i:=iplot:
     # Ground Motion
     xg[i]:=x0x[j]: yg[i]:=x0y[j]: zg[i]:=x0z[j]:
     # Cuboids rotation
     rotu||i := sqrt( (X||j[4])^2 + (X||j[5])^2 +
     (X||j[6])^2 ) * nor:
     roto||i := sqrt( (X||j[10])^2 + (X||j[11])^2 +
     (X||j[12])^2 ) * nor:
     diru||i := [ (X||j[4]), (X||j[5]), (X||j[6]) ];
     diro||i := [ (X||j[10]), (X||j[11]), (X||j[12]) ];
     # Diagonal end points position of the cuboids
     ENDu1t||i := [ ENDu1[1] + ( (X||j[1])+xg[i] )*nor,
                    ENDu1[2] + ( (X||j[2])+yg[i] )*nor,
                    ENDu1[3] + ( (X||j[3])+zg[i] )*nor ];
     ENDu2t||i := [ ENDu2[1] + ( (X||j[1])+xg[i] )*nor,
                    ENDu2[2] + ( (X||j[2])+yg[i] )*nor,
                    ENDu2[3] + ( (X||j[3])+zg[i] )*nor ];
     ENDo1t||i := [ ENDo1[1] + ( (X||j[7])+xg[i] )*nor,
                    ENDo1[2] + ( (X||j[8])+yg[i] )*nor,
                    ENDo1[3] + ( (X||j[9])+zg[i] )*nor ];
     ENDo2t||i := [ ENDo2[1] + ( (X||j[7])+xg[i] )*nor,
                    ENDo2[2] + ( (X||j[8])+yg[i] )*nor,
                    ENDo2[3] + ( (X||j[9])+zg[i] )*nor ];
end do:
```

Creating the plot objects.

```
> for j from 1 by 1 to nplot do
     CUBuc[j] := cuboid( ENDu1t||j, ENDu2t||j, color=red ):
     CUBoc[j] := cuboid( ENDo1t||j, ENDo2t||j, color=blue):
end do:
```

Translate and Rotation the basic elements

Ground and the Cuboids

Create the plots of the translated cuboids and the ground.

The plot of the upper cuboid must be translated by the constant distance between both bodies in the initial position, because all motions are calculated with respect of the bodies' center of gravity. Finally, the cuboids are rotated around the rotation vectors.

```
> G[1]:=GROUND:
> CUBu[1]:=CUBuc[1]:
  CUBo[1]:=translate( CUBoc[1], DSuo[1], DSuo[2], DSuo[3]):
> for j from 2 by 1 to nplot do
      G[j] := translate( GROUND, xg[j]*nor, yg[j]*nor,
      zg[j]*nor ):
      CUBu[j] := rotate( CUBuc[j], rotu||j, [ [0,0,0],
      diru||j ] ):
      CUBo[j] := translate( rotate( CUBoc[j], roto||j,
      [[0,0,0],diro||j] ),
      DSuo[1],DSuo[2],DSuo[3]):
end do:
```

Spring Connection Positions

Next, we calculate the spring connections between the cuboids. We use the spring deformation relationships derived above in the calculation.

```
> iplot:=0:
> for j from 1 by ds to ndt do
      iplot:=iplot+1:
      i:=iplot:
      # Ground and lower spring connections
      FSgx||i := [ seq( x0x[j]*nor + a[n,x], n=1..4 ) ]:
      FSgy||i := [ seq( x0y[j]*nor + a[n,y], n=1..4 ) ]:
      FSgz||i := [ seq( x0z[j]*nor - DSuo[3], n=1..4 ) ]:
      # Lower spring and lower cuboid connections
      FSux||i := [ seq( eval( Re((u1+x0x[j])*nor+a[p,x]), {
      seq( q||n=X||j[n], n=1..6 ),
      seq( ak||n=a[p,n], n in {x,y,z} ) } ), p=1..4 )]:
      FSuy||i := [ seq( eval( Re((u2+x0y[j])*nor+a[p,y]), {
      seq( q||n=X||j[n], n=1..6 ),

      seq( ak||n=a[p,n], n in {x,y,z} ) } ), p=1..4 ) ]:
      FSuz||i := [ seq( eval( Re((u3+x0z[j])*nor+a[p,z]), {
      seq( q||n=X||j[n], n=1..6 ),
      seq( ak||n=a[p,n], n in [x,y,z] ) } ), p=1..4 ) ]:
      # Lower cuboid and upper spring connections
      FTux||i := [ seq( eval( Re((u1+x0x[j])*nor+b[p,x]), {
      seq( q||n=X||j[n], n=1..6 ),
      seq( ak||n=a[p,n], n in [x,y] ), akz=bzq||p } ),
      p=1..4 ) ]:
      FTuy||i := [ seq( eval( Re((u2+x0y[j])*nor+b[p,y]), {
      seq( q||n=X||j[n], n=1..6 ),
```

```
     seq( ak||n=a[p,n], n in [x,y] ),
     akz=bzq||p } ), p=1..4 ) ]:
     FTuz||i := [ seq( eval( Re((u3+x0z[j])*nor+bzq||p), {
     seq( q||n=X||j[n], n=1..6 ),
     seq( ak||n=a[p,n], n in [x,y] ), akz=bzq||p } ),
                  p=1..4 ) ]:
     # Upper spring and upper cuboid connections
     FSox||i := [ seq( eval(
     Re((u1+x0x[j])*nor+b[p,x]+DSuo[1]), { seq(
     q||n=X||j[n+6], n=1..6 ),

     seq( ak||n=b[p,n], n in [x,y,z] ) } ),
                    p=1..4 ) ]:
     FSoy||i := [ seq( eval(
     Re((u2+x0y[j])*nor+b[p,y]+DSuo[2]), { seq(
     q||n=X||j[n+6], n=1..6 ),

     seq( ak||n=b[p,n], n in [x,y,z] )} ), p=1..4 ) ]:
     FSoz||i := [ seq( eval(
     Re((u3+x0z[j])*nor+b[p,z]+DSuo[3]), { seq(
     q||n=X||j[n+6], n=1..6 ),

     seq( ak||n=b[p,n], n in [x,y,z] )} ), p=1..4 ) ]:
end do:
```

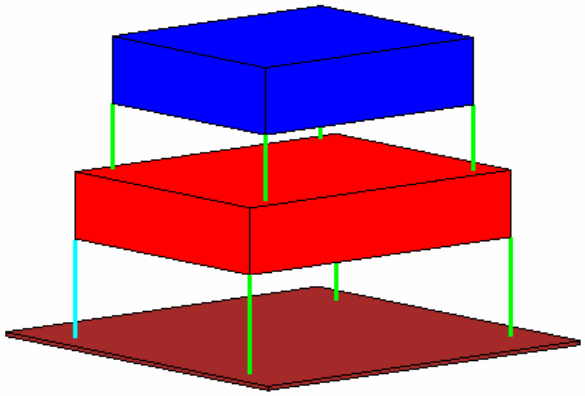
Connecting the Bodies

```
> for j from 1 by 1 to nplot do
     SU[j] :=
     line([FSgx||j[1],FSgy||j[1],FSgz||j[1]],[FSux||j[1],FS
     uy||j[1],FSuz||j[1]],color=green,thickness=3),

     line([FSgx||j[2],FSgy||j[2],FSgz||j[2]],[FSux||j[2],FS
     uy||j[2],FSuz||j[2]],color=cyan,thickness=3),

     line([FSgx||j[3],FSgy||j[3],FSgz||j[3]],[FSux||j[3],FS
     uy||j[3],FSuz||j[3]],color=green,thickness=3),

     line([FSgx||j[4],FSgy||j[4],FSgz||j[4]],[FSux||j[4],FS
     uy||j[4],FSuz||j[4]],color=green,thickness=3);

     SO[j] :=
     line([FSox||j[1],FSoy||j[1],FSoz||j[1]],[FTux||j[1],FT
     uy||j[1],FTuz||j[1]],color=green,thickness=3),

     line([FSox||j[2],FSoy||j[2],FSoz||j[2]],[FTux||j[2],FT
     uy||j[2],FTuz||j[2]],color=green,thickness=3),

     line([FSox||j[3],FSoy||j[3],FSoz||j[3]],[FTux||j[3],FT
```

```
     uy||j[3],FTuz||j[3]],color=green,thickness=3),

     line([FSox||j[4],FSoy||j[4],FSoz||j[4]],[FTux||j[4],FT
     uy||j[4],FTuz||j[4]],color=green,thickness=3);
end do:
```

Displaying the Animation

```
> ANIM:=seq(display({CUBu[j],CUBo[j],G[j],SU[j],SO[j]}),
     j=1..nplot):
> display(ANIM,insequence=true,style=patch,
     scaling=constrained,axes=none,orientation=[50,80]);
```



References

[1] E. Luz, "Schwingungsprobleme im Bauwesen", (Expert Verlag, 1992)
[2] K.-J. Bathe; E. L. Wilson, "Numerical Methods In Finite Elemente Analysis", (Prentice-Hall, 1976)