**Integrating across Friction Reversal and Collision**



t

h

h\*

a) monitor  that causes Coulomb friction and check for zero crossing

or

b) identify if collision occurs

interpolate to find h\* when anomaly occurs

**spring-mass-damper with viscous damping using ODE23**

 

mean time step = 17.05 msec

CPU time = 0.0466 sec

**spring-mass-damper with Coulomb friction using ODE23**



mean time step = 9.4787 msec

CPU time = 0.2339 sec

**drag-sled with Coulomb friction using ODE23**



**Integrating Differential Algebraic Equations (DAE)**

**Forward Dynamics**

know current state  integrate 

**DAE**

initial values must be kinematically consistent  and 

at each new time step 

 functions of 

 constant (typically)

 functions of 

 functions of 



solve for 

integrate 

**Kinematic consistency**

Do new  values satisfy  ?

Do new  values satisfy  ?

**Methods for DAEs**

1) Direct Integration

2) Coordinate Partitioning

3) Constraint Stabilization

**Direct Integration**

equations are numerically stiff

eigenvalues 

for M = mobility, typically M relatively slow modes for rigid body motion

(nq-M) very fast modes for algebraic equations

requires VERY small h

relatively slow execution speed

more time steps allow more numerical error to accumulate - solution error

does not guarantee  position error = 

does not guarantee  velocity error = 

**Coordinate Partitioning**



partition  rearrange 

apply Gaussian elimination with full pivoting to  to identify  and 

(an example is provided at the end of these notes)



 is not singular

 corresponding to  can be considered dependent

 corresponding to  can be considered independent

**Simple Partitioning**

compute all  using 

HOWEVER only independent generalized accelerations  are integrated across time step h

provides new values for  and 

 are used to determine  by enforcing  with Newton-Raphson at each time step



 are used to determine  by enforcing  with 

unfortunately coordinate partitioning does not always identify the same dependent and independent generalized coordinates at each time step which adds considerable bookkeeping

**Full Partitioning**

use partition 

rearrange 



the third equation can be rearranged symbolically 

and substituted into the first equation 

finally providing differential equations in only independent generalized coordinates

  are NOT unit matrix/vector





only independent generalized accelerations  are integrated across time step h

provides new values for  and 

 are used to determine  by enforcing  with Newton-Raphson at each time step



**Constraint Stabilization**

traditional acceleration constraint 

Baumgarte's modified acceleration constraint 







  is NOT unit vector



use direct integration

no general and uniformly valid method of selecting  and  have been found

can lead to divergence near singularities

**Gaussian Elimination with Full Pivoting**



first pivot position



find largest absolute value below and to the right of pivot



swap columns to get largest absolute value in pivot column (swap columns 1 and 3)

**and** swap corresponding rows in list of variables



swap rows to get largest absolute value in pivot row (swap rows 1 and 3)

**and** swap rows in right-hand side (RHS)



normalize pivot row including RHS



multiply pivot row times 4 and subtract from row 2



multiply pivot row times -2 and subtract from row 3



second pivot position



find largest absolute value below and to the right of pivot



swap columns to get largest absolute value in pivot column (swap columns 2 and 3)

**and** swap corresponding rows in list of variables



swap rows to get largest absolute value in pivot row (swap rows 1 and 3)

**and** swap rows in RHS



normalize pivot row including RHS



multiply pivot row times -0.6 and subtract from row 3



third pivot position



normalize pivot row including RHS



back-substitution



**Results for full Jacobian (kinematically driven) for web cutter at phi2 = 85.5 deg**

JAC =

 1.0000 0 1.9938 0 0 0 0 0 0

 0 1.0000 -0.1569 0 0 0 0 0 0

 -1.0000 0 1.9938 1.0000 0 8.6477 0 0 0

 0 -1.0000 -0.1569 0 1.0000 -6.4024 0 0 0

 0 0 0 -1.0000 0 4.3622 1.0000 0 -7.8080

 0 0 0 0 -1.0000 0.6374 0 1.0000 -7.0912

 0 0 0 0 0 0 1.0000 0 11.2196

 0 0 0 0 0 0 0 1.0000 0.0400

 0 0 1.0000 0 0 0 0 0 0

gamma\_transpose =

 -6.1949 -78.7134 -13.4946 -88.5732 -1.7142 -2.2860 0.0138 -3.8620 0

a =

 1.0000 0 0 0.0891 0 0 0 0 0

 0 1.0000 0.2306 0 -0.1156 0 0 0 0.1156

 0 0 1.0000 0 0.5016 0 0 0 0

 0 0 0 1.0000 0.5949 0 0 0 -0.8871

 0 0 0 0 1.0000 -0.7133 0 0.7133 -0.5281

 0 0 0 0 0 1.0000 -0.8598 -0.1402 -0.3149

 0 0 0 0 0 0 1.0000 -0.0013 -0.0016

 0 0 0 0 0 0 0 1.0000 0.0623

 0 0 0 0 0 0 0 0 1.0000

b\_transpose =

 0.0012 -1.5605 -3.1071 1.1661 67.3777 -11.1126 -3.9788 -87.9252 -147.8370

xlist\_transpose =

 9 6 3 7 1 5 8 2 4

qdd\_transpose =

 -6.1949 -78.7134 0 -147.8370 -72.4115 14.8187 -126.2931 -4.3123 11.2577

**Results for reduced Jacobian (dynamically driven) for web cutter at phi2 = 85.5 deg**

JAC8 =

 1.0000 0 1.9938 0 0 0 0 0 0

 0 1.0000 -0.1569 0 0 0 0 0 0

 -1.0000 0 1.9938 1.0000 0 8.6477 0 0 0

 0 -1.0000 -0.1569 0 1.0000 -6.4024 0 0 0

 0 0 0 -1.0000 0 4.3622 1.0000 0 -7.8080

 0 0 0 0 -1.0000 0.6374 0 1.0000 -7.0912

 0 0 0 0 0 0 1.0000 0 11.2196

 0 0 0 0 0 0 0 1.0000 0.0400

gamma8\_transpose =

 -6.1949 -78.7134 -13.4946 -88.5732 -1.7142 -2.2860 0.0138 -3.8620

a =

 1.0000 0 0 0.0891 0 0 0 0 0

 0 1.0000 0.2306 0 -0.1156 0 0 0.1156 0

 0 0 1.0000 0 0.5016 0 0 0 0

 0 0 0 1.0000 0.5949 0 0 -0.8871 0

 0 0 0 0 1.0000 -0.7133 0 -0.5281 0.7133

 0 0 0 0 0 1.0000 -0.8598 -0.3149 -0.1402

 0 0 0 0 0 0 1.0000 -0.0016 -0.0013

 0 0 0 0 0 0 0 1.0000 16.0487

b\_transpose =

 1.0e+003 \*

 0.0000 -0.0016 -0.0031 0.0012 0.0674 -0.0111 -0.0040 -1.4111

xlist\_transpose =

 9 6 3 7 1 5 8 4 2

% t\_gefp.m - test Gaussian elimination with full pivoting

% produces upper triangular form

% HJSIII, 12.03.12

% input

clear

% 3x3 square matrix in notes

a = [ 4 2 -2 ;

 1 3 4 ;

 2 1 5 ];

b = [ 6 ; -1 ; 2 ];

% Jacobian and gamma for web cutter at phi2 = 85.5 deg when y3P - y4Q = 0

JAC = [ ...

 1.0000 0 1.9938 0 0 0 0 0 0 ;

 0 1.0000 -0.1569 0 0 0 0 0 0 ;

 -1.0000 0 1.9938 1.0000 0 8.6477 0 0 0 ;

 0 -1.0000 -0.1569 0 1.0000 -6.4024 0 0 0 ;

 0 0 0 -1.0000 0 4.3622 1.0000 0 -7.8080 ;

 0 0 0 0 -1.0000 0.6374 0 1.0000 -7.0912 ;

 0 0 0 0 0 0 1.0000 0 11.2196 ;

 0 0 0 0 0 0 0 1.0000 0.0400 ;

 0 0 1.0000 0 0 0 0 0 0 ];

gamma = [ -6.1949 -78.7134 -13.4946 -88.5732 -1.7142 -2.2860 0.0138 -3.8620 0 ]';

% call routine for full Jacobian (inverse dynamics)

qdd = inv(JAC) \* gamma;

[ a, b, xlist ] = gefp( JAC, gamma );

JAC

gamma\_transpose = gamma'

a

b\_transpose = b'

xlist\_transpose = xlist'

qdd\_transpose = qdd'

% call routine for reduced Jacobian (forward dynamics)

JAC8 = JAC(1:8,:);

gamma8 = gamma(1:8);

[ a, b, xlist ] = gefp( JAC8, gamma8 );

JAC8

gamma8\_transpose = gamma8'

a

b\_transpose = b'

xlist\_transpose = xlist'

% bottom of t\_gefp

function [ a, b, xlist ] = gefp( amat, brhs )

% Gaussian elimination with full pivoting for linear model - amat \* xvec = brhs

% produces upper triangular form - a \* x = b

% HJSIII, 12.03.12

%

% USAGE

% [ a, b, xlist ] = gefp( amat, brhs )

%

% INPUT

% amat = nrow x ncol matrix of coefficients (may be non-square)

% brhs = nrow x 1 right-hand-side vector

%

% OUTPUT

% a = nrow x ncol upper triangular matrix

% b = nrow x 1 matching right-hand-side vector

% xlist = ncol x 1 vector of indices for shuffled entries after column pivoting

%

% nrow = ncol - linear solution but does not perform back-substitution

% nrow < ncol - triangulation of non-square matrix for DAE coordinate partitioning

% row/column order

a = amat;

b = brhs;

[ nrow, ncol ] = size( a );

% fill vector to sort variables

xlist = ( 1 : ncol )';

% loop through all rows - do not process last row

nrowm1 = nrow - 1;

for ipivot = 1 : nrowm1,

% find maximum absolute value below and right of pivot

% indices are relative to location within subset

 [ maxcol, indrow ] = max( abs( a( ipivot:nrow, ipivot:ncol ) ) );

 [ v, jcol ] = max( maxcol );

 irow = indrow( jcol );

% adjust indices to entire matrix

 irow = irow + ipivot - 1;

 jcol = jcol + ipivot - 1;

% swap columns - skip if already in pivot column

% must swap entire column

% also swap rows in variable list

 if jcol ~= ipivot,

 temp\_col = a(:,ipivot);

 a(:,ipivot) = a(:,jcol);

 a(:,jcol) = temp\_col;

 temp\_x = xlist(ipivot);

 xlist(ipivot) = xlist(jcol);

 xlist(jcol) = temp\_x;

 end

% swap rows - skip if already in pivot row

% right-hand-side handled by concatenation

 if irow ~= ipivot,

 temp\_row = a( ipivot, ipivot:ncol );

 a( ipivot, ipivot:ncol ) = a( irow, ipivot:ncol );

 a( irow, ipivot:ncol ) = temp\_row;

 temp\_b = b(ipivot);

 b(ipivot) = b(irow);

 b(irow) = temp\_b;

 end

% normalize pivot row

 d = a(ipivot,ipivot);

 a( ipivot, ipivot:ncol ) = a( ipivot, ipivot:ncol ) / d;

 b(ipivot) = b(ipivot) / d;

% factor rows below pivot

 ipp1 = ipivot + 1;

 for irow = ipp1 : nrow,

 d = a( irow, ipivot );

 a( irow, ipivot:ncol ) = a( irow, ipivot:ncol ) - d \* a( ipivot, ipivot:ncol );

 b(irow) = b(irow) - d \* b(ipivot);

 end

% bottom of loop for rows

end

% last row - only normalize

ipivot = nrow;

d = a(ipivot,ipivot);

a( ipivot, ipivot:ncol ) = a( ipivot, ipivot:ncol ) / d;

b(ipivot) = b(ipivot) / d;

return

% bottom of gefp