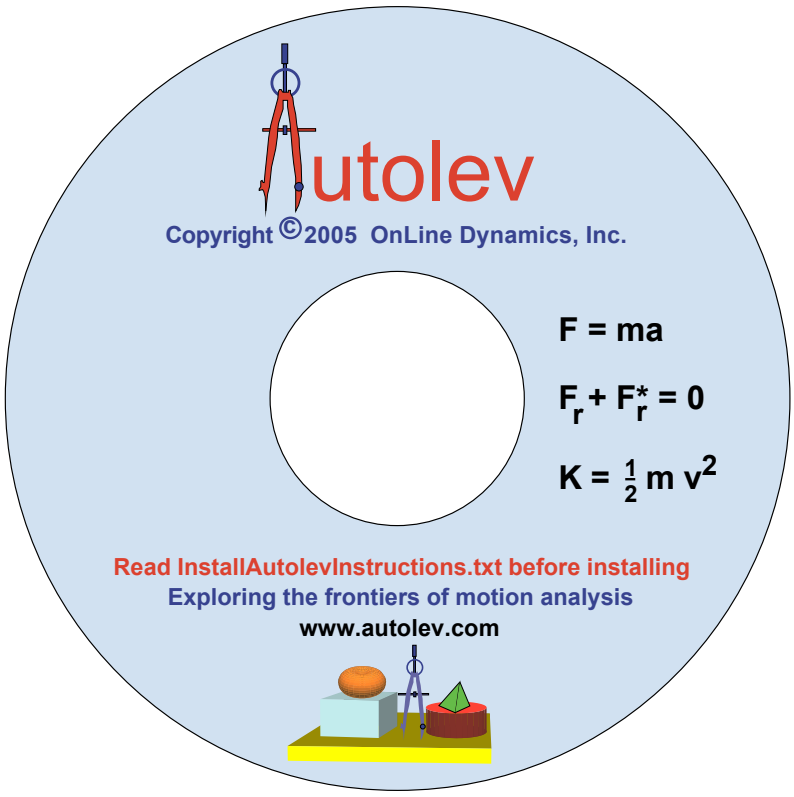
Autolev Tutorial



OnLine Dynamics, Inc.
1605 Honfleur Drive
Sunnyvale, CA 94087 USA
Phone: 408-736-9566
Web: http://www.autolev.com

Last updated March 12, 2005 for Autolev version 4

Contents

1 Getting Started

1.1 Running Autolev

- On a Windows computer, double click on the Autolev icon
- On a Windows computer, `cd` to the directory containing the file "al.exe" and type `al`
- On a Unix or Linux computer, `cd` to the directory containing the file "al" and type `./al`

When Autolev starts, the following prompt is displayed:

```
(1)
```

Autolev is case-insensitive. Thus "apple", "Apple", and "APPLE" are equivalent. Depending on the command entered, Autolev may or may not output a response. Autolev distinguishes between input and output lines by preceding output lines with an arrow (->). Try the following example.

```
   (1) Constants A, B
   (2) X = 4*(A+B) - 3*(a-b)
-> (3) X = A + 7*B
```

1.2 Online Help

Typing `WHAT` at an Autolev prompt causes an alphabetical list of commands to be displayed on the screen. To obtain help in connection with a particular command on the list, type `HELP commandname`. For example, type `HELP SAVE` for more information on `SAVE`.

1.3 The Save Command

Issuing a `SAVE` command causes a text file dealing with the current session to be created. Two of the six syntaxes of the `SAVE` command are

```
SAVE filename.al
SAVE filename.all
```

The first causes all input lines to be stored in `filename.al`, whereas the second causes both input and response lines to be stored in `filename.all`. The file `filename.al` may be subsequently edited with a text editor and re-run as described in the next section.

1.4 Autolev Input Files

Instead of entering Autolev commands interactively, one can use a text editor to create a text file, e.g., `michele.al`, and then execute the commands in `michele.al` by:

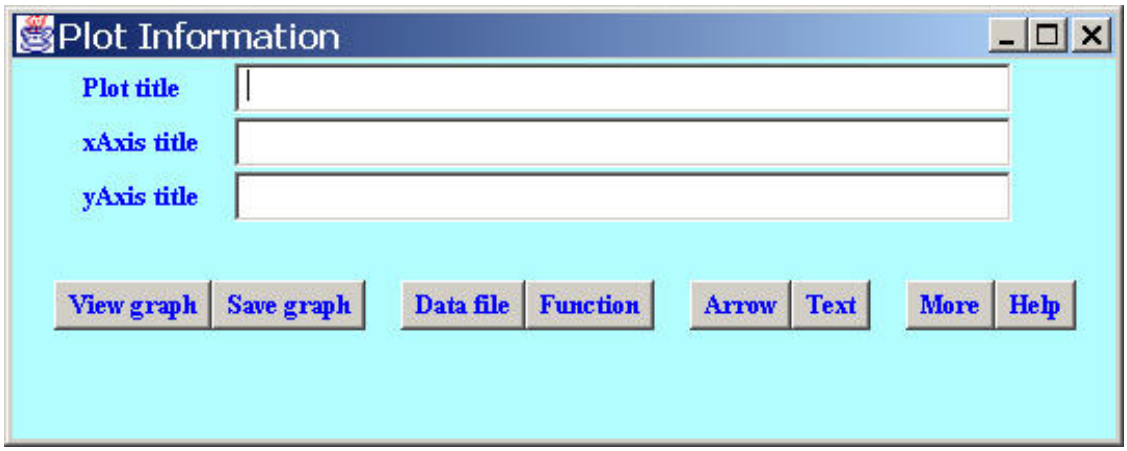
- invoking Autolev and typing `RUN michele.al` at an Autolev line prompt
- typing `al michele.al` at the operating system prompt (Windows, Unix, Linux)
- dragging and dropping the file `michele.al` on top of the Autolev icon (Windows)

One advantage of execution by reading from a text file is that commands may be broken into multiple lines. An Autolev input line cannot contain more than 256 characters, but lines of unlimited length may be entered, provided that an ampersand `(&)` appear as the last non-blank character of each but the last input line, informing Autolev that the command continues on the next line.

1.5 Running the Alplot plotting program

To invoke Alplot on a Windows machine,[1] double click on the Alplot icon and follow the on-screen instructions. The Windows version of Alplot supports drag and drop, i.e., drag a data file on top of the Alplot icon to start Alplot and load the data file. Alternately, the Windows version of Alplot can be invoked by typing one of the following syntaxes at the operating system prompt:

`alplot`	Invokes `alplot`
`alplot filename`	Invokes `alplot` and loads the data file `filename`
`alplot filenameA filenameB ...`	Invokes `alplot` and loads the data files `filenameA`, `filenameB`, ...
`alplot filenameA filenameA`	Invokes `alplot` and loads the data file `filenameA`, twice



[1] To run Alplot on a Linux/Unix machine, type `alplot` or `alplot filename` at the operating system prompt.

2 Mathematical Entities

2.1 Scalars

The names of scalar quantities must start with a letter. This letter may be followed by alphanumeric characters or underscores (_). For example, `XYZ`, `AB3`, and `ABC_3` are acceptable names. 1^{st}, 2^{nd}, 3^{rd}, ..., ordinary derivatives of a scalar variable with respect to `T` are denoted with primes ('). The prime appear at the end of the name, and each prime represents one differentiation, e.g., `X''` is the second ordinary derivative of `X` with respect to `T`. At most 64 characters, including primes, may be used to form the name of a scalar quantity. Before a scalar quantity may be used in an analysis, it must be either named in a `Constants`, `Specified`, `Variables`, `MotionVariables`, `MotionVariables'`, `Imaginary`, `Mass`, or `Inertia` declaration, or defined by appearing on the left-hand side of an equals sign in an assignment. For example,

```
Constants  A,B=3      % Declares the constants A and B and assigns the value 3 to B
Constants  C+         % Declares C to be a non-negative constant
Constants  D-         % Declares D to be a non-positive constant
Specified  Phi        % Declares Phi as a function of time, constants, and variables
Variables  q, s       % Declares the variables q and s
Variables  x''        % Declares the variables x, x', and x''
Variables  y{3}'      % Declares the variables y1, y2, y3, y1', y2', and y3'
MotionVariables  u{3} % Declares the generalized speeds u1, u2, and u3
MotionVariables' w'   % Declares the generalized speed w and its time derivative w'
Imaginary  j          % Declares j to be the imaginary number, i.e., sqrt(-1)
Tina = 2*pi           % Creates the quantity Tina and assigns the value 2*pi to Tina
```

2.2 Vectors, Dyadics, and Polyadics

To distinguish scalars, vectors, dyadics, and polyadics from each other, one "greater than" $>$ is appended to the end of a vector's name, two are appended to the end of a dyadic's name, and three to the end of the name of a triadic or higher-order polyadic. The names of vectors, dyadics, and other polyadics must start with a letter. This letter may be followed by alphanumeric characters or

underscores (_). At most 64 characters, including > symbols, may be used to form the name of a vector, dyadic, or higher order polyadic. Examples of names are

`ed>`, `ed1>`	vectors
`john>>`, `john_paul>>`	dyadics
`tom>>>`	triadic or higher order polyadic

2.3 Matrices

Matrices start with a left bracket (`[`) and end with a right bracket (`]`). Elements of a row are separated from each other by a comma, while rows are separated by semicolons. For example, `[1,2,3;4,5,6]` denotes the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$. The name of a matrix must start with a letter. This letter may be followed by alphanumeric characters or underscores (_). At most 64 characters may be used to form the name of a matrix.

The name of an element of a matrix consists of the name of the matrix, followed by a left bracket (`[`), an expression that must evaluate to a positive integer, a comma, another expression which must evaluate to a positive integer, and a right bracket (`]`). If a matrix is one-dimensional, then one may refer to its elements in a shorthand form, namely, the name of a matrix followed by a left bracket (`[`), an expression that evaluates to a positive integer, and a right bracket (`]`). For example, `X[7]` denotes the 7^{th} element of the matrix `X`, regardless of whether `X` is a row matrix or a column matrix.

2.4 Reserved Names

- `T` is a reserved variable, often used to denote time.
- `Pi` is a reserved constant with the value of 3.1415...
- `0>` is the zero vector.
- `1>>` is the unit dyadic.
- Unless otherwise declared, `imaginary =` $\sqrt{-1}$.
- `ZERO` is reserved for the matrix of expressions which one sets equal to zero to form dynamical equations of motion. Type `HELP Kane` for details.
- `DEPENDENT` and `AUXILIARY` are reserved for matrices of expressions which one sets equal to zero to form motion constraint equations. Type `HELP Constrain` for details.
- For backward compatibility, `Variables U{n}` or `Variables U{n}'` introduce motion variables.
- `ZEE_NOT` is reserved for a matrix of scalar quantities that are excluded from `Z1,Z2,...`.

3 Physical Entities

3.1 Bodies, Frames, Newtonian, Particles, Points

A Newtonian reference frame must be named in the `Newtonian` declaration and may consist of at most 11 characters. The names of bodies, frames, particles, and points must appear in `Bodies`, `Frames`, `Particles`, and `Points` declarations and may consist of at most 27 characters. These names must begin with a letter followed by alphanumeric characters (no underscores).

```
Bodies    A    % Declares the (massive) rigid body A.
               % Introduces orthonormal vectors A1>, A2>, A3> fixed in A.
               % Declares a point Ao that is coincident with A's mass center
Frames    B    % Declares the (massless) reference frame B.
               % Introduces orthonormal vectors B1>, B2>, B3> fixed in B.
               % Declares a point Bo that is fixed on B
Newtonian N    % Declares N as a Newtonian (inertial) reference frame.
               % Introduces orthonormal vectors N1>, N2>, N3> fixed in N.
               % Declares a point No that is fixed on N
Particles C,D  % Declares the (massive) particles C and D
Points    E,F  % Declares the (massless) points E and F
```

3.2 Syntactical Forms

The underscore (_) separates points and/or reference frames in the names of position vectors, velocities, accelerations, direction cosine matrices, angular velocities, angular accelerations, forces, torques, and inertia dyadics. For example:

```
P_O_Q>     % Position vector from point O to point Q
V_P_N>     % Velocity of point P in reference frame N
A_D_C>     % Acceleration of point D in reference frame C
W_B_F>     % Angular velocity of body B in reference frame F
ALF_E_A>   % Angular acceleration of body E in reference frame A
Force_P>   % Force acting on point P
```

```
Torque_B>  % Torque of a couple acting on body B
I_B_P>>    % Inertia dyadic of body B for point P
A_B        % Direction cosine matrix relating A1>,A2>,A3> to B1>,B2>,B3>
```

Note: Element `A_B[i,j]` of `A_B` is defined as `DOT(Ai>,Bj>) (i,j=1,2,3)`. As a consequence,

$$\texttt{A\_B} = \begin{bmatrix} A1> \cdot B1> & A1> \cdot B2> & A1> \cdot B3> \\ A2> \cdot B1> & A2> \cdot B2> & A2> \cdot B3> \\ A3> \cdot B1> & A3> \cdot B2> & A3> \cdot B3> \end{bmatrix}$$

3.3 Mass Declarations

The masses of bodies and particles are specified by the `Mass` declaration. For example:

```
   (1) Bodies     A,B
   (2) Particles  C
   (3) Mass       A=12.5, B=mB, C=mC=10
-> (4) mC = 10
   (5) Mass_A = Mass(A)          % Returns the mass of A
-> (6) Mass_A = 12.5
   (7) Mass_AB = Mass(A,B)       % Returns Mass(A) + Mass(B)
-> (8) Mass_AB = 12.5 + mB
   (9) TotalMass = Mass()        % Returns Mass(A) + Mass(B) + Mass(C)
-> (10) TotalMass = 22.5 + mB
```

After `A`, `B`, and `C` have been declared as bodies or particles, the `Mass` declaration specifies that `A` has a mass of `12.5`, `B` has a mass of `mB`, and `C` has a mass of `mC` where `mC=10`. Once the mass of a body or particle has been declared, its mass can be referenced by typing `Mass( NameOfBodyOrParticle)`.

3.4 Inertia Declarations

As previously mentioned, `I_B_P>>` denotes the inertia dyadic of body B for point P. Inertia dyadics may be created in the following three ways:

- By explicit assignment. To assign a dyadic to `I_B_P>>`, type, for example,

  ```
  I_B_P>> = 100*a1>*a1> + 200*a2>*a2> + 250*a3>*a3>
  ```

- With one of the variants of the `Inertia` declaration. For example, typing

  ```
  Inertia  A, I11,I22,I33,I12,I23,I31
  ```

  is equivalent to typing

  ```
  Constants  I11+, I22+, I33+, I12, I23, I31
  I_A_Ao>> = I11*A1>*A1> + I12*A1>*A2> + I31*A1>*A3>
  ```

```
        + I12*A2>*A1> + I22*A2>*A2> + I23*A2>*A3>
        + I31*A3>*A1> + I23*A3>*A2> + I33*A3>*A3>
```

Any inertia scalar that is not specified is set equal to zero by default.

- If the inertia dyadic of a body for a point has been entered, Autolev can find the inertia dyadic of the body for a second point if the mass of the body has been declared and appropriate position vectors have been entered. For example,

```
      (1) Points P
      (2) Bodies B
      (3) Mass B=MB
      (4) Constants I,J,L
      (5) P_Bo_P> = L*B1>
   -> (6) P_BO_P> = L*B1>
      (7) I_B_P>> = I*B1>*B1> + I*B2>*B2> + J*B3>*B3>
   -> (8) I_B_P>> = I*B1>*B1> + I*B2>*B2> + J*B3>*B3>
      (9) I_B_BO>> = Inertia(Bo,B)
   -> (10) I_B_BO>> = I*B1>*B1> + (I-MB*L^2)*B2>*B2> + (J-MB*L^2)*B3>*B3>
```

3.5 Forces and Torques

`Force_P>` is the force acting on a point or particle `P`, and `Torque_B>` is the torque of the couple acting on a frame or body B. There are four ways to assign values to a `Force` or `Torque` vector:

- `Force_Q> = 7*a1>  % Explicit assignment overwrites previous value for Force_Q>`
- `Force_P> += 5*a3>  % Adds 5*a3> to the value of Force_P>`
- `Force_P> -= 5*a3>  % Subtracts 5*a3> from the value of Force_P>`
- `Force(P/Q,VEC>)  % Equivalent to  Force_P> -= VEC>`
  `Force_Q> += VEC>`

- `Torque_B> = 7*B3>  % Explicit assignment overwrites previous value for Torque_B>`
- `Torque_A> += VEC>  % Adds VEC> to the value of Torque_A>`
- `Torque_A> -= VEC>  % Subtracts VEC> from the value of Torque_A>`
- `Torque(A/B,VEC>)  % Equivalent to  Torque_A> -= VEC>`
  `Torque_B> += VEC>`

4 Procedures for Solving Problems

4.1 Numerical Integration of Differential Equations

Autolev writes Matlab, C, or Fortran code for the numerical integration of sets of differential equations. The differential equations can be either dynamical equations of motion generated by Autolev or differential equations entered directly by the user. The procedure for generating Matlab, C, or Fortran codes and their input files is summarized below.

- Declare all variables and their derivatives.
- Enter the equations governing the highest derivative of each variable.
- Use `Input` statements to specify the values of constants, the initial values of variables, and values of integration parameters such as `tFinal`, `integStp`, `absErr`, and `relErr`.
- Use `Output` statements to specify the quantities to be output.
- Type `Code ODE() Filename.ext` (`ext` is `m`, `c`, `for`, or `f`).
  When `ext` is `c`, this creates ready-to-compile C code in `Filename.c` and an input file `Filename.in`.
  When `ext` is `f` or `for`, this creates Fortran code in `Filename.ext` and an input file `Filename.in`.
  When `ext` is `m`, this creates ready-to-run Matlab code in `Filename.m`

```
% Example - numerically integrate the following differential equations:
% dx/dt = -a*x - b*y
% dy/dt = -y - t*x^2
%         where a and b are constants and t is time
Constants a,b
Variables x', y'
x' = -a*x - b*y
y' = -y - t*x^2
Input a=1, b=2, x=2, y=3
Input tFinal=5, integStp=0.1
Output t,x,y,x',y'
Code ODE() odexy.c
```

This creates ready-to-compile C code in the file `odexy.c` and an input file `odexy.in` that is read by the executable program, e.g., `odexy.exe`. The executable program performs numerical integration of the differential equations for `x` and `y` from `t=0` to `t=5` with integration steps of 0.1.

4.2 Numerical Solution of Nonlinear Nondifferential Equations

The procedure for writing Matlab, C, or Fortran code for the numerical solution of sets of nonlinear algebraic equations is as follows:

- Declare all variables.
- Create a matrix whose elements represent the nonlinear equations.
- Use `Input` statements to specify guesses for the values of the variables, the values of constants, and values of convergence parameters such as `absErr` and `relErr`.
- Type `Code Nonlinear(matrixName,variables) Filename.ext` (`ext` is `m`, `c`, `for`, or `f`).

```
% Example - numerically solve the following equations for x and y:
%   x^2 + y^2 - r = 0
%   y - a*sin(x) = 0
Variables  x, y
Constants  a, r
eqn[1] = x^2 + y^2 - r              % Equation of a circle
eqn[2] = y - a*sin(x)               % Equation of a sinusoid
Input a=1.0, r=1.0
Input x=0.5, y=0.5
Code Nonlinear(eqn, x,y) katy.m
```

This creates the ready-to-run Matlab file `katy.m`. To solve the nonlinear equations, invoke Matlab and type `katy` at the Matlab prompt. To try a different initial guess for `x` or `y` or to use a different value for `a` or `b`, use a text editor to edit `katy.m`.

4.3 Numerical Solution of Linear Algebraic Equations

The procedure for writing Matlab, C, or Fortran code for the numerical solution of sets of linear equations is as follows:[2]

- Declare all variables.
- Create a matrix whose elements represent the linear equations.
- Use `Input` statements to specify the the values of constants.
- Type `Code Algebraic(matrixName,variables) Filename.ext` (`ext` is `m`, `c`, `for`, or `f`).

```
% Example - numerically solve the following equations for x and y:
%   a11*x + a12*y = b1
%   a21*x + a22*y = b2
Variables  x, y
Constants  a{1:2,1:2}, b{1:2}
eqn[1] = a11*x + a12*y - b1
eqn[2] = a21*x + a22*y - b2
Input a11=1, a12=2, a21=3, a22=4, b1=5, b2=6
Code Algebraic(eqn, x,y) becky.for
```

[2] Autolev's `Solve` command solves sets of linear equations in Autolev (without writing Matlab, C, or Fortran code).

This creates ready-to-compile Fortran code in the file `becky.for` and an input file `becky.in` that is read by the executable program, e.g., `becky.exe`. The executable program solves the linear equations for `x` and `y` for the given values of `a11`, `a12`, `a21`, `a22`. To solve for `x` and `y` with different values of `a11`, `a12`, `a21`, `a22`. use a text editor to edit `becky.in` and then re-run `becky.exe`.

4.4 Equations of Motion

Although Autolev can assist one in formulating equations of motion by any method, it is especially well suited for carrying out dynamic analyses with Kane's method (see Dynamics Online: Theory and Implementation with Autolev™ by T. R. Kane and D. A. Levinson). Examples are presented in Chapter 5 of this tutorial. The procedure for generating equations of motion via Kane's method is summarized below.

- Declare a `Newtonian` reference frame
- Declare `Bodies`, `Frames`, `Points`, and `Particles`
- Declare generalized speeds and their time-derivatives with a declaration of the form `MotionVariables' wx', wy', wz', x'', y'', z''`
- Declare generalized coordinates and their time-derivatives with a declaration of the form `Variables' qx', qy', qz'`
- If necessary, create kinematical differential equations relating time-derivatives of generalized coordinates to the motion variables, e.g., `qx' = wx*cos(qx) + wy*sin(qy)`
- Form position vectors and direction cosine matrices
- Form angular and linear velocities
- If necessary, impose motion constraints with the `Constrain` command
- Form angular and linear accelerations. Autolev can *automatically* form angular and linear accelerations by differentiation of corresponding angular and linear velocities. For large problems, or problems which require real-time solution, form accelerations with the `A2pts` and `A1pt` commands and turn `AutoZ ON`.
- Enter expressions for forces and torques
- Form equations of motion by entering `Zero = Fr() + FrStar()`
- Issue the `Kane()` command so the equations of motion are in a form suitable for motion simulation.

4.5 Motion Constraints

One function of the `Constrain` command is to solve constraint equations for dependent motion variables when a system is subject to motion constraints. Additionally, if the time-derivatives of motion variables are declared (i.e., the `MotionVariables'` declaration has been issued), the `Constrain` command produces expressions for the time-derivatives of the dependent motion variables. For more information about the `Constrain` command, type `HELP Constrain` at the Autolev line prompt, and/or see Dynamics Online: Theory and Implementation with Autolev™. Examples are presented in Chapter 5 of this tutorial. A summary of the procedure follows.

- Declare motion variables that characterize the motion of the unconstrained system.
- Construct m expressions such that setting the expressions equal to zero produces the motion constraint equations. Assign these expressions to the first m elements of the `Dependent` matrix. For example,

  ```
  Dependent[1] = U2 - U1
  Dependent[2] = U3 + U2
  ```

- Type `Constrain( Dependent[ Ui, Uj,...] )`
  where `Ui, Uj, ...` are the names of the m dependent motion variables.
  The `Constrain` command solves the m equations associated with the `Dependent` matrix for the m dependent motion variables, and if the time-derivatives of the dependent motion variables have been declared, the `Constrain` command solves for those also.

```
   (1) %  Problem:  Triple pendulum whose tip has a specified motion
   (2) Newtonian       N            % Newtonian reference frame
   (3) Bodies          A, B, C      % Links of pendulum
   (4) Points          P            % Distal end of C
   (5) MotionVariables u{3}         % Generalized speeds
   (6) Variables       qA, qB, qC   % Angles for A, B, C
   (7) Constants       LA, LB, LC   % Lengths of A, B, C
   (8) Specified       vx, vy       % Specified motion of P
   (9) Simprot(N, A, 3, qA)
-> (10) N_A = [COS(qA), -SIN(qA), 0; SIN(qA), COS(qA), 0; 0, 0, 1]
   (11) Simprot(N, B, 3, qB)
-> (12) N_B = [COS(qB), -SIN(qB), 0; SIN(qB), COS(qB), 0; 0, 0, 1]
   (13) Simprot(N, C, 3, qC)
-> (14) N_C = [COS(qC), -SIN(qC), 0; SIN(qC), COS(qC), 0; 0, 0, 1]
   (15) V_P_N> = LA*u1*A2> + LB*u2*B2> + LC*u3*C2>
-> (16) V_P_N> = LA*u1*A2> + LB*u2*B2> + LC*u3*C2>
   (17) Dependent[1] = Dot( V_P_N>, N1> ) - vx
-> (18) Dependent[1] = -vx - LA*SIN(qA)*u1 - LB*SIN(qB)*u2 - LC*SIN(qC)*u3
   (19) Dependent[2] = Dot( V_P_N>, N2> ) - vy
-> (20) Dependent[2] = LA*COS(qA)*u1 + LB*COS(qB)*u2 + LC*COS(qC)*u3 - vy
   (21) Constrain( Dependent[u2,u3] )
-> (22) u2 = -(vx*COS(qC)+vy*SIN(qC)+LA*SIN(qA-qC)*u1)/(LB*SIN(qB-qC))
-> (23) u3 = (vx*COS(qB)+vy*SIN(qB)+LA*SIN(qA-qB)*u1)/(LC*SIN(qB-qC))
```

5 Example Problems

5.1 Mathematical Capabilities Examples

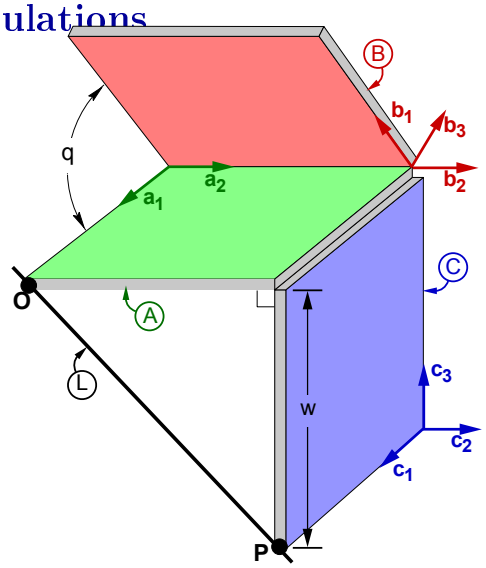
```
   (1) %     File:  tutor1.al    [Autolev: http://www.autolev.com]
   (2) %  Problem:  Examples of mathematical functions
   (3) %--------------------------------------------------------------------
   (4) %       Mathematical declarations: variables, constants
   (5) Variables  x', y'
   (6) Constants  a, b, c, d
   (7) Imaginary  i
   (8) %--------------------------------------------------------------------
   (9) %       Create an expression and then rearrange (simplify) it
   (10) E = (x+2*y)^2 + 3*(7+x)*(x+y)         % Create an expression
-> (11) E = (x+2*y)^2 + 3*(7+x)*(x+y)

   (12) Expand( E, 1:2 )                      % Clear parentheses
-> (13) E = 21*x + 21*y + 4*x^2 + 4*y^2 + 7*x*y

   (14) Factor( E, x )                        % Factor on x
-> (15) E = 21*y + 4*y^2 + 4*x*(5.25+x+1.75*y)

   (16) %--------------------------------------------------------------------
   (17) %       Mathematical commands
   (18) Dy = D( E, y )                        % Partial derivative wrt. y
-> (19) Dy = 21 + 7*x + 8*y

   (20) Dt = Dt( E )                          % Total derivative wrt. t
-> (21) Dt = 21*y' + 8*y*y' + 7*(3+y)*x' + 7*x*(y'+1.142857*x')

   (22) TY = Taylor( x*COS(y), 0:7, x=0,y=0)  % Multi-variable Taylor series
-> (23) TY = 0.001388889*x*(720+30*y^4-360*y^2-y^6)

   (24) F = Evaluate(TY, x=1, y=0.5)          % Symbolic/numerical evaluation
-> (25) F = 0.8775825

   (26) Poly = Polynomial( [a,b,c], x )       % Creates a*x^2 +b*x +c
-> (27) Poly = c + b*x + a*x^2
```

```
   (28) Root1 = Roots( [1; 2; 3; 4] )         % Roots of x^3+2*x^2+3*x+4 = 0
-> (29) Root1 = [-1.650629; -0.1746854 - 1.546869*i; -0.1746854 + 1.546869*i]

   (30) Root2 = Roots( Poly, x, 2 )           % Some symbolic roots
-> (31) Root2[1] = -0.5*(b-(b^2-4*a*c)^0.5)/a
-> (32) Root2[2] = -0.5*(b+(b^2-4*a*c)^0.5)/a

   (33) %---------------------------------------------------------------------
   (34) %       Creating row or column matrices
   (35) RowMatrix = [1, 2, 3, 4]     % Create a 1x4 matrix
-> (36) RowMatrix = [1, 2, 3, 4]

   (37) ColMatrix = [1; 2; 3; 4]     % Create a 4x1 matrix
-> (38) ColMatrix = [1; 2; 3; 4]

   (39) Zero[1] = a*x + b*y - 1      % Assign elements of column matrix
-> (40) Zero[1] = -1 + a*x + b*y

   (41) Zero[2] = c*x + d*y - Pi
-> (42) Zero[2] = -3.141593 + c*x + d*y

   (43) %---------------------------------------------------------------------
   (44) %       Solve a set of linear equations
   (45) Solve( Zero, x, y )
-> (46) x = -(3.141593*b-d)/(a*d-b*c)
-> (47) y = (3.141593*a-c)/(a*d-b*c)

   (48) %---------------------------------------------------------------------
   (49) %       Creating rectangular matrices
   (50) M0 = [a, b; c, 0]            % Create a 2x2 matrix
-> (51) M0 = [a, b; c, 0]

   (52) M0[2,2] := d                 % Assign element of rectangular matrix
-> (53) M0[2,2] = d

   (54) M1 = [M0, [1,2;3,4] ]        % Elements of matrices can be matrices
-> (55) M1 = [a, b, 1, 2; c, d, 3, 4]

   (56) %---------------------------------------------------------------------
   (57) %       Matrix commands
   (58) M2 = M0 + M0                 % Matrix addition
-> (59) M2 = [2*a, 2*b; 2*c, 2*d]

   (60) M3 = M0 * M0                 % Matrix multiplication
-> (61) M3 = [a^2 + b*c, b*(a+d); c*(a+d), b*c + d^2]

   (62) M4 = Transpose(M0)           % Matrix transposition
-> (63) M4 = [a, c; b, d]
```

```
   (64) M5 = Inv(M0)                % Matrix inversion
-> (65) M5[1,1] = d/(a*d-b*c)
-> (66) M5[1,2] = -b/(a*d-b*c)
-> (67) M5[2,1] = -c/(a*d-b*c)
-> (68) M5[2,2] = a/(a*d-b*c)

   (69) M6 = Diagmat(3,1)           % Makes 3x3 matrix with 1 along diagonal
-> (70) M6 = [1, 0, 0; 0, 1, 0; 0, 0, 1]

   (71) M7 = Cols( M0, 1 )          % Returns column 1 of M0
-> (72) M7 = [a; c]

   (73) M8 = Rows( M0, 2, 1:2)      % Returns rows 2 and rows 1 through 2 of M0
-> (74) M8 = [c, d; a, b; c, d]

   (75) N1 = Rows( M0 )             % Returns the number of rows in M0
-> (76) N1 = 2

   (77) N2 = Det( M0 )              % Determinant of a matrix
-> (78) N2 = a*d - b*c

   (79) M9 = Evaluate( M0, a=1, b=2, c=3, d=4 )
-> (80) M9 = [1, 2; 3, 4]

   (81) Lambda = Eig( M9 )          % Eigenvalues
-> (82) Lambda = [-0.3722813; 5.372281]

   (83) Eig( M9, EigValue, EigVec)  % Eigenvalues/eigenvectors
-> (84) EigValue = [-0.3722813; 5.372281]
-> (85) EigVec = [-0.8245648, 0.5657675; -0.4159736, -0.9093767]

   (86) %---------------------------------------------------------------------
   (87) %       Record Autolev responses
```

5.2 Mass, Mass Center, and Inertia Calculations

The figure to the right shows three identical uniform hinge-connected plates A, B, and C. Right-handed sets of mutually perpendicular unit vectors $\mathbf{a}_i$, $\mathbf{b}_i$, and $\mathbf{c}_i$ (i=1,2,3) are fixed in A, B, and C, respectively, with $\mathbf{a}_2 = \mathbf{b}_2 = \mathbf{c}_2$ parallel to the hinge connecting A and B. The plates are thin and square and have dimension w=1 meter and mass m=12 kg. Points O and P mark corners of A and C and the angle q characterizes the orientation of B in A.

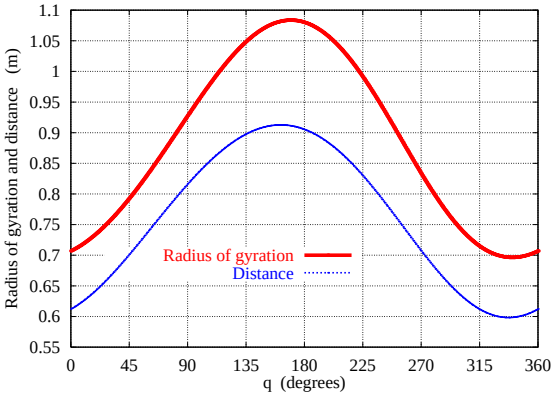


1. Find the distance between line $\overline{OP}$ and the mass center of the system formed by A, B, and C.
   Result:
   $$\text{Distance} = 0.1178511\sqrt{42 + \cos(q)^2 + 6\sin(q) - 16\cos(q)}$$

2. For q=90°, find λ_i (i=1,2,3), the system's *central* principal moments of inertia and find the angle between $\mathbf{a}_2$ and the principal axis associated with this system's *minimum* moment of inertia.
   Result:
   λ_1=5 λ_2=13 λ_3=14 Angle = 65.90516°

3. Radius of gyration is a measure of how far objects are from a line and is defined as $\sqrt{I/m}$ where I is the system's moment of inertia about that line and m is the mass of the system. Using physical intuition, estimate the values of q that produce the smallest and largest radius of gyration for this system about line $\overline{OP}$ and provides a reason for choosing these values.
   Result:
   $q_{small} \approx$ ° $q_{large} \approx$ ° $(0 \le q \le 360°)$
   Reason:

4. Plot the system's mass center distance from line $\overline{OP}$ and the system's radius of gyration about line $\overline{OP}$ for $0 \le q \le 360°$. Determine the minimum/maximum distance and radius of gyration and associated values of q.

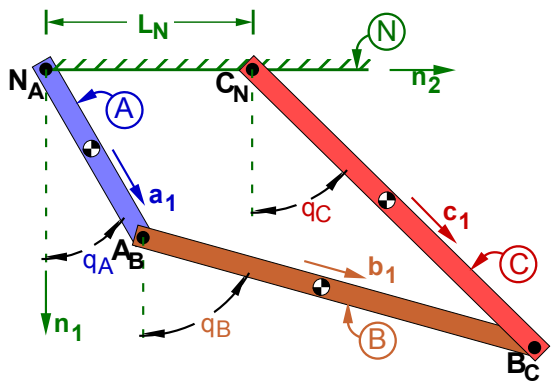


	Minimum		Maximum	
	Value (m)	q	Value (m)	q
Distance	0.598318	337°	0.912686	161°
Gyration	0.696493	340°	1.083707	169°

```
%     File:  tutor2.al    [Autolev: http://www.autolev.com]
%  Problem:  Mass and inertia calculations
%--------------------------------------------------------------------
AutoEpsilon 1.0E-14     % Round to integers
%--------------------------------------------------------------------
%       Physical declarations
Bodies     A, B, C      % Right, middle, top plate
Points     O, P         % Points on L
Points     CM           % Center of mass of system
%--------------------------------------------------------------------
%       Mathematical declarations
Variables  q            % Angle between plates A and B
Constants  w=1          % Width of plate
%--------------------------------------------------------------------
%       Mass and inertia
Mass       A=m=12, B=m, C=m
Inertia    A, m*w^2/12, m*w^2/12, m*w^2/6
Inertia    B, m*w^2/12, m*w^2/12, m*w^2/6
Inertia    C, m*w^2/12, m*w^2/6,  m*w^2/12
%--------------------------------------------------------------------
%       Geometry relating unit vectors
Simprot(A, B, -2, q)
A_C = Diagmat(3,1)      % 3x3 identity matrix
%--------------------------------------------------------------------
%       Position vectors     (Ao, Bo, Co are mass centers)
P_O_Ao> = -0.5*w*A1> + 0.5*w*A2>
P_O_Bo> = -w*A1> + 0.5*w*A2> + 0.5*w*B1>
P_O_Co> = w*A2> - 0.5*w*C1> - 0.5*w*C3>
P_O_P> =  w*A2> - w*C3>
%--------------------------------------------------------------------
%       Center of mass of system
P_O_CM> = CM(O)
uL> = Unitvec( P_O_P> )                % Unit vector parallel to line L
Distance = Mag( Cross(uL>, P_O_CM> ) ) % Distance from line L to CM
%--------------------------------------------------------------------
%       Inertia dyadic and matrix of system about its center of mass
I_System_CM>> = Inertia(CM, A, B, C)
I_System_CM = Matrix( A, I_System_CM>> )
Eig( Evaluate(I_System_CM, q=90*Units(deg,rad)), Lambda, EigenVecs )
EigenVec = Rows(EigenVecs, 1)             % Extract eigenvector
EigenVec> = Vector(A, EigenVec)           % Corresponding unit vector
AngleBetweenEigenVecAndA2 = acos( Dot(EigenVec>,A2>) ) * Units(rad,deg)
%--------------------------------------------------------------------
%       Moment of inertia of system about line joining O and P
I_System_O>> = Inertia(O)              % Inertia dyadic of system about O
IL = Dot(uL>, Dot(I_System_O>>, uL> )) % Moment of inertia about L
MTotal = Mass()                        % Mass of system
Gyration = sqrt( IL / MTotal )         % Radius of gyration about L
%--------------------------------------------------------------------
%       Output and units for use in program created by CODE command
UnitSystem  kg, meter, sec
Output q degrees, Distance meters, Gyration meters
%--------------------------------------------------------------------
%       C code generation for numerical solution
CODE Algebraic() [q deg=0,360,1] tutor2.c
%--------------------------------------------------------------------
%       Record Autolev responses
Save tutor2.all
```

5.3 Solving Nonlinear Equations

The figure to the right shows a planar four-bar linkage consisting of uniform rigid links A, B, and C and ground N. Link A is connected with revolute joints to N and B at points N_A and A_B, respectively. Link C is connected with revolute joints to N and B at points C_N and B_C, respectively.



Right-handed orthogonal unit vectors $\mathbf{a}_i$, $\mathbf{b}_i$, $\mathbf{c}_i$, and $\mathbf{n}_i$ (i=1,2,3) are fixed in A, B, C, and N, with $\mathbf{a}_1$ directed from N_A to A_B, $\mathbf{b}_1$ from A_B to B_C, $\mathbf{c}_1$ from C_N to B_C, $\mathbf{n}_1$ vertically downward, $\mathbf{n}_2$ from N_A to C_N, and $\mathbf{a}_3 = \mathbf{b}_3 = \mathbf{c}_3 = \mathbf{n}_3$ parallel to the axes of the revolute joints. Other symbols useful in the analysis are shown to the right.

Quantity	Identifier	Type
Distance from N_A to A_B	L_A	constant
Distance from A_B to B_C	L_B	constant
Distance from B_C to C_N	L_C	constant
Distance from C_N to N_A	L_N	constant
Angle between $\mathbf{n}_1$ and $\mathbf{a}_1$	q_A	variable
Angle between $\mathbf{n}_1$ and $\mathbf{b}_1$	q_B	variable
Angle between $\mathbf{n}_1$ and $\mathbf{c}_1$	q_C	variable

1. Create two configuration constraint equations f_i=0 (i=1,2) which relate q_A, q_B, and q_C.
   Result:

$$f_1 = L_A * \cos(q_A) + L_B * \cos(q_B) - L_C * \cos(q_C)$$
$$f_2 = L_A * \sin(q_A) + L_B * \sin(q_B) - L_C * \sin(q_C) - L_N$$

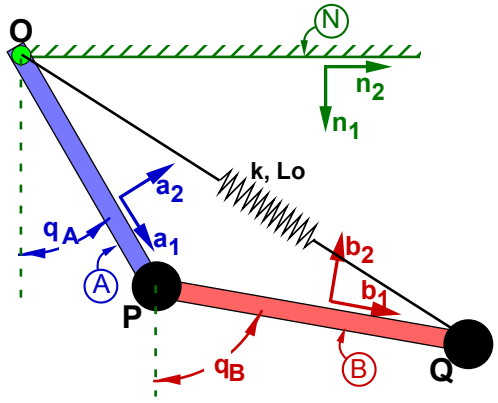
2. Determine values of q_B and q_C which satisfy the configuration constraint equations when L_A=1 m, L_B=2 m, L_C=2 m, L_N=1 m, and q_A=30° .
   Result:

$$q_B = 74.4775° \qquad q_C = 45.5225°$$

```
%     File:  tutor3.al    [Autolev: http://www.autolev.com]
%  Problem:  Solving a set of nonlinear equations (4-bar linkage)
%--------------------------------------------------------------------
%       Physical declarations: Frames and Points
Frames     A, B, C, N
Points     NA, AB, BC, CN
%--------------------------------------------------------------------
%       Mathematical declarations:  variables, constants
Variables  qA, qB, qC        % Configuration variables
Constants  LA, LB, LC, LN    % Lengths
%--------------------------------------------------------------------
%       Geometry relating unit vectors
Simprot(N, A, 3, qA)
Simprot(N, B, 3, qB)
Simprot(N, C, 3, qC)
%--------------------------------------------------------------------
%       Position vectors
P_NA_AB> = LA*A1>
P_AB_BC> = LB*B1>
P_CN_BC> = LC*C1>
P_NA_CN> = LN*N2>
%--------------------------------------------------------------------
%       Configuration constraints
Loop> = P_NA_AB> + P_AB_BC> + P_BC_CN> + P_CN_NA>
f1 = Dot( Loop>, N1> )
f2 = Dot( Loop>, N2> )
%--------------------------------------------------------------------
%       Input constants, variables, etc. for CODE
UnitSystem  kg, meter, sec
Input  LA=1 m,  LB=2 m,  LC=2 m,  LN=1 m
Input  qA=30 deg
Input  qB=60 deg, qC=20 deg            % Guess
Input  absErr=1.0E-07, relErr=1.0E-07  % Error tolerances
%--------------------------------------------------------------------
%       C/Fortran/Matlab code generation for numerical solution
CODE Nonlinear( [f1; f2], qB,qC )  tutor3.c
%--------------------------------------------------------------------
% Comment: Nonlinear equations may have more than one solution.
%          The solution depends on a guess.  If a guess does not
%          provide a satisfactory answer, try another guess.
%--------------------------------------------------------------------
%       Record Autolev responses
Save tutor3.all
```

5.4 Spring-Restrained Double Pendulum – Static Equilibrium

A spring-restrained double pendulum having a light linear spring and two light rigid rods A and B supports two particles P and Q in a Newtonian reference frame N. Rod A is connected with frictionless revolute joints to N and B at points O and P, respectively. Right-handed sets of mutually perpendicular unit vectors $\mathbf{n}_i$, $\mathbf{a}_i$, and $\mathbf{b}_i$ (i=1,2,3) are fixed in N, A, and B, with $\mathbf{n}_1$ vertically downward, $\mathbf{a}_1$ directed from O to P, $\mathbf{b}_1$ directed from P to Q, and $\mathbf{n}_3 = \mathbf{a}_3 = \mathbf{b}_3$ parallel to the axes of the revolute joints.



The distance from O to P is L_A and the distance from P to Q is L_B. The masses of P and Q are denoted m^P and m^Q. The spring's natural length is L_o and its spring constant is k. The angles q_A and q_B characterize the orientation of A and B in N and the motion variables are $\dot{q}_A$ and $\dot{q}_B$.

1. Form expressions for the system's generalized forces $F_{\dot{q}_A}$ and $F_{\dot{q}_B}$.

   Result:

$$
\begin{aligned}
L_{OQ} &= \sqrt{L_A^2 + L_B^2 + 2*L_A*L_B*\cos(q_A - q_B)} \\
\text{Stretch} &= L_{OQ} - L_o \\
F_{\dot{q}_A} &= L_A \left[k*L_B*\sin(q_A - q_B)*\frac{\text{Stretch}}{L_{OQ}} - g*(m^P + m^Q)*\sin(q_A) \right] \\
F_{\dot{q}_B} &= L_B \left[-k*L_A*\sin(q_A - q_B)*\frac{\text{Stretch}}{L_{OQ}} - g*m^Q*\sin(q_B) \right]
\end{aligned}
$$

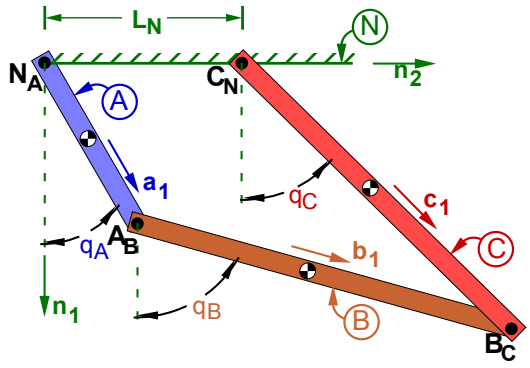
2. Optional**: Determine four solutions for q_A and q_B between -180° and 180° for this system to be in static equilibrium when $L_A = 1$ m, $L_B = 2$ m, $m^P = 10$ kg, $m^Q = 20$ kg, $L_o = 1$ m, $k = 200 \frac{\text{n}}{\text{m}}$, and $g = 9.81 \frac{\text{m}}{\text{sec}^2}$. Circle the ***stable*** solutions.

   Result:

Solution 1	$q_A = 0°$	$q_B = 0°$
Solution 2	$q_A = -50.1492°$	$q_B = 35.1548°$
Solution 3	$q_A = 50.1492°$	$q_B = -35.1548°$
Solution 4	$q_A = 180°$	$q_B = 180°$

```
%     File:  tutor4.al    [Autolev: http://www.autolev.com]
%  Problem:  Spring-restrained double pendulum -- Equilibrium configuration
%--------------------------------------------------------------------
%       Newtonian, bodies, frames, particles, points
Newtonian  N
Frames     A, B
Particles  P, Q
Points     O
%--------------------------------------------------------------------
%       Mathematical declarations: variables, constants, mass
MotionVariables  qA', qB'   % qA' and qB' are generalized speeds
Constants        LA, LB     % Lengths of A, B
Constants        k, Lo      % Spring constant, natural length of spring
Constants        g          % Local gravitational acceleration
%--------------------------------------------------------------------
%       Mass
Mass  P=mP, Q=mQ
%--------------------------------------------------------------------
%       Geometry relating unit vectors
Simprot( N, A, 3, qA )
Simprot( N, B, 3, qB )
%--------------------------------------------------------------------
%       Position vectors
P_O_P> = LA*A1>
P_P_Q> = LB*B1>
%--------------------------------------------------------------------
%       Angular velocities
w_A_N> = qA'*A3>
w_B_N> = qB'*B3>
%--------------------------------------------------------------------
%       Velocities
v_O_N> = 0>
v2pts( N, A, O, P )
v2pts( N, B, P, Q )
%--------------------------------------------------------------------
%       Forces
Gravity( g*N1> )
LOQ = Mag( P_O_Q> )               % Distance from O to Q
Stretch = LOQ - LO                % Stretch of spring
Uvec> = P_O_Q> / LOQ              % Unit vector from O to Q
Force(O/Q, -k*Stretch*Uvec> )     % Spring force
%--------------------------------------------------------------------
%       Generalized active forces
Zero = Fr()                       % Static equilibrium Fr() = 0
Zero := Factor( Zero, g )         % Simplify slightly
%--------------------------------------------------------------------
%       Units system for CODE input/output conversions
UnitSystem  kg,meter,sec
%--------------------------------------------------------------------
%       Input constants, variables, etc. for CODE
Input  LA=1 m, LB=2 m,  k=200 n/m, Lo=1 m
Input  g=9.81 m/sec^2,  mP=10 kg,  mQ=20 kg
Input  qA=-30 deg,  qB=30 deg          % Guess
Input  absErr=1.0E-07, relErr=1.0E-07  % Error tolerances
%--------------------------------------------------------------------
%       C code generation for numerical solution
Code Nonlinear(Zero,qA,qB)  tutor4.c
%--------------------------------------------------------------------
%       Record Autolev responses
Save tutor4.all
```

5.5 Four-Bar Linkage – Equilibrium Configuration



The figure to the right shows a planar four-bar linkage consisting of uniform rigid links A, B, and C and ground N. Link A is connected with revolute joints to N and B at points N_A and A_B, respectively. Link C is connected with revolute joints to N and B at points C_N and B_C, respectively.

Right-handed orthogonal unit vectors $\mathbf{a}_i$, $\mathbf{b}_i$, $\mathbf{c}_i$, and $\mathbf{n}_i$ (i=1,2,3) are fixed in A, B, C, and N, with $\mathbf{a}_1$ directed from N_A to A_B, $\mathbf{b}_1$ from A_B to B_C, $\mathbf{c}_1$ from C_N to B_C, $\mathbf{n}_1$ vertically downward, $\mathbf{n}_2$ from N_A to C_N, and $\mathbf{a}_3 = \mathbf{b}_3 = \mathbf{c}_3 = \mathbf{n}_3$ parallel to the axes of the revolute joints.

The **independent motion variable** is $\dot{q}_A$ and the **dependent motion variables** are $\dot{q}_B$ and $\dot{q}_C$. Other symbols useful in this analysis are shown to the right. For the questions that follow, use L_A=1 m, L_B=2 m, L_C=2 m, L_N=1 m.

Quantity	Identifier	Type
Distance from N_A to A_B	L_A	constant
Distance from A_B to B_C	L_B	constant
Distance from B_C to C_N	L_C	constant
Distance from C_N to N_A	L_N	constant
Mass of A	m^A	constant
Mass of B	m^B	constant
Mass of C	m^C	constant
Angle between $\mathbf{n}_1$ and $\mathbf{a}_1$	q_A	variable
Angle between $\mathbf{n}_1$ and $\mathbf{b}_1$	q_B	variable
Angle between $\mathbf{n}_1$ and $\mathbf{c}_1$	q_C	variable

1. Create two motion constraint equations which relate $\dot{q}_A$, $\dot{q}_B$, and $\dot{q}_C$.
   Result:
$$2*\sin(q_C)*\dot{q}_C - 2*\sin(q_B)*\dot{q}_B - \sin(q_A)*\dot{q}_A = 0$$
$$\cos(q_A)*\dot{q}_A + 2*\cos(q_B)*\dot{q}_B - 2*\cos(q_C)*\dot{q}_C = 0$$

2. Solve the motion constraint equations for $\dot{q}_B$ and $\dot{q}_C$ in terms of $\dot{q}_A$.
   Result:
$$\dot{q}_B = -0.5*\frac{\sin(q_A-q_C)}{\sin(q_B-q_C)}*\dot{q}_A \qquad \dot{q}_C = -0.5*\frac{\sin(q_A-q_B)}{\sin(q_B-q_C)}*\dot{q}_A$$

3. Form this system's generalized force $F_{\dot{q}_A}$ assuming frictionless revolute joints, a horizontal force $H*\mathbf{n}_2$ is applied to B_C, m^A=10 kg, m^B=20 kg, m^C=20 kg, and local gravity is 9.81 $\frac{\text{m}}{\text{sec}^2}$.[3]
   Result:
$$F_{\dot{q}_A} = \sin(q_A-q_B)\,\frac{196.2\,\sin(q_C) - H\,\cos(q_C)}{\sin(q_B-q_C)} - 147.15\,\sin(q_A)$$

4. Optional**: Determine values of q_A, q_B, q_C which satisfy the configuration constraints and the static equilibrium condition $F_{\dot{q}_A} = 0$ when H=200 N. Circle the ***stable*** solution.
   Result:

Solution 1	$q_A = 19.987°$	$q_B = 71.662°$	$q_C = 38.325°$
Solution 2	$q_A = 249.29°$	$q_B = 140.18°$	$q_C = 199.11°$

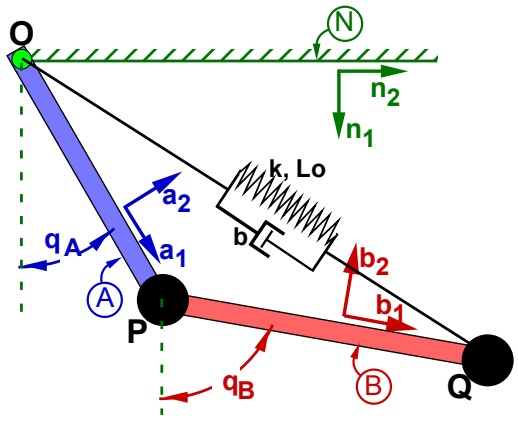
[3] The form of the expression for $F_{\dot{q}_A}$ depends on how it is calculated.

```
%     File:  tutor5.al
%  Problem:  Four-Bar Linkage -- Equilibrium Configuration
%----------------------------------------------------------------------
%       Newtonian, bodies, frames, particles, points
Newtonian  N
Bodies     A, B, C
Points     NA, AB, BC, CN
%----------------------------------------------------------------------
%       Variables, constants, mass, and specified
MotionVariables  qA', qB', qC'
Constants        H                        % Horizontal force on BC
Constants        LA=1, LB=2, LC=2, LN=1
Mass             A=mA=10, B=mB=20, C=mC=20
%----------------------------------------------------------------------
%       Geometry relating unit vectors
Simprot(N, A, 3, qA)
Simprot(N, B, 3, qB)
Simprot(N, C, 3, qC)
%----------------------------------------------------------------------
%       Position vectors
P_NA_AB> = LA*A1>
P_AB_BC> = LB*B1>
P_CN_BC> = LC*C1>
P_NA_CN> = LN*N2>
%----------------------------------------------------------------------
%       Angular velocities
W_A_N> = qA'*A3>
W_B_N> = qB'*B3>
W_C_N> = qC'*C3>
%----------------------------------------------------------------------
%       Velocities
v_AB_N> = Cross( W_A_N>, P_NA_AB> )
v_BC_N> = Cross( W_C_N>, P_CN_BC> )
%----------------------------------------------------------------------
%       Configuration Constraints
Loop> = P_NA_AB> + P_AB_BC> + P_BC_CN> + P_CN_NA>
ConfigurationConstraint[1] = Dot( Loop>, N1> )
ConfigurationConstraint[2] = Dot( Loop>, N2> )
%----------------------------------------------------------------------
%       Motion constraints
Dependent = Dt( ConfigurationConstraint )
Constrain( Dependent[qB',qC'] )
%----------------------------------------------------------------------
%       Forces - replaces gravity forces with equivalent set
g = 9.81
Force_AB> = 1/2*( Mass(A)*g*n1> + Mass(B)*g*n1> )
```

```
Force_BC> = 1/2*( Mass(B)*g*n1> + Mass(C)*g*n1> )
Force_BC> += H*N2>
%-------------------------------------------------------------------
%       Equations of motion
Zero = Fr()
Zero[2] = ConfigurationConstraint[1]
Zero[3] = ConfigurationConstraint[2]
%-------------------------------------------------------------------
%       Units system for CODE input/output conversions
UnitSystem  kg,meter,sec
%-------------------------------------------------------------------
%       Input constants, variables, etc. for CODE
Input  H=200 newton
Input  qA=30 deg, qB=60 deg, qC=30 deg   % Guess
Input  absErr=1.0E-07, relErr=1.0E-07    % Error tolerances
%-------------------------------------------------------------------
%       MATLAB code generation for numerical solution
CODE Nonlinear( Zero, qA,qB,qC )  tutor5.m
%-------------------------------------------------------------------
%       Record Autolev responses
Save tutor5.all
```

5.6 Spring-Damper Double Pendulum – Motion and Contact Forces

A spring-damper double pendulum having a light linear spring-damper and two light rigid rods A and B supports two particles P and Q in a Newtonian reference frame N. Rod A is connected with frictionless revolute joints to N and B at points O and P, respectively. Right-handed sets of mutually perpendicular unit vectors $\mathbf{n}_i$, $\mathbf{a}_i$, and $\mathbf{b}_i$ (i=1,2,3) are fixed in N, A, and B, with $\mathbf{n}_1$ vertically downward, $\mathbf{a}_1$ directed from O to P, $\mathbf{b}_1$ directed from P to Q, and $\mathbf{n}_3 = \mathbf{a}_3 = \mathbf{b}_3$ parallel to the axes of the revolute joints.



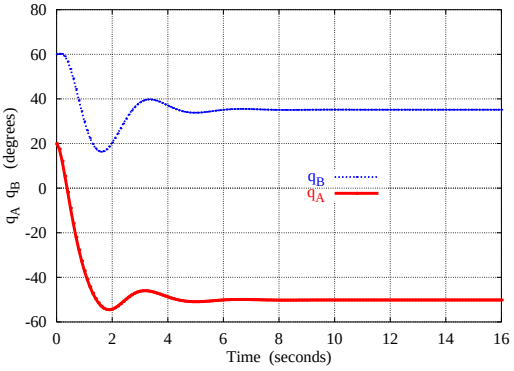
The distance from O to P is L_A and the distance from P to Q is L_B. The masses of P and Q are denoted m^P and m^Q. The spring's natural length is L_o, its spring constant is k, and the damping constant is b. The angles q_A and q_B characterize the orientation of A and B in N and the motion variables are $\dot{q}_A$ and $\dot{q}_B$. A damping torque of $-c*\dot{q}_A$ and $-c*\dot{q}_B$ act on A and B.

1. Find the equations which govern the motion of this system.

   Result:

$$
\begin{aligned}
L_{OQ} &= \sqrt{L_A^2 + L_B^2 + 2*L_A*L_B*\cos(q_A - q_B)} \\
\text{Stretch} &= L_{OQ} - L_o \\
\text{Stretch'} &= -L_A*L_B\sin(q_A - q_B)*(\dot{q}_A - \dot{q}_B)/L_{OQ} \\
0 &= L_A^2\,(m^P + m^Q)\,\ddot{q}_A + L_A\,L_B\,m^Q\cos(q_A - q_B)\,\ddot{q}_B + L_A\,L_B\,m^Q\sin(q_A - q_B)\,\dot{q}_B^2 \\
&\quad - L_A\,L_B\,(k\,\text{Stretch} + b\,\text{Stretch'})\sin(q_A - q_B)/L_{OQ} + c\,\dot{q}_A + g\,L_A\,(m^P + m^Q)\sin(q_A) \\
0 &= L_A\,L_B\,m^Q\cos(q_A - q_B)\,\ddot{q}_A + m^Q\,L_B^2\,\ddot{q}_B - L_A\,L_B\,m^Q\sin(q_A - q_B)\,\dot{q}_A^2 \\
&\quad + L_A\,L_B\,(k\,\text{Stretch} + b\,\text{Stretch'})\sin(q_A - q_B)/L_{OQ} + c\,\dot{q}_B + g\,L_B\,m^Q\sin(q_B)
\end{aligned}
$$

2. Plot q_A and q_B for $0 \le t \le 16$ sec when $L_A = 1$ m, $L_B = 2$ m, $m^P = 10$ kg, $m^Q = 20$ kg, $L_o = 1$ m, $k = 200\ \frac{\text{n}}{\text{m}}$, $b = 100\ \frac{\text{n*sec}}{\text{m}}$, $c = 100\ \frac{\text{n*m*sec}}{\text{rad}}$, $g = 9.81\ \frac{\text{m}}{\text{sec}^2}$, and the system is released from ***rest*** with $q_A = 20°$ and $q_B = 60°$. Verify that $q_A(t{=}16)$ and $q_B(t{=}16)$ are identical to one of the solutions in problem (5.4.5.4). Form a numerical integration checking function and verify that it remains constant during numerical integration.

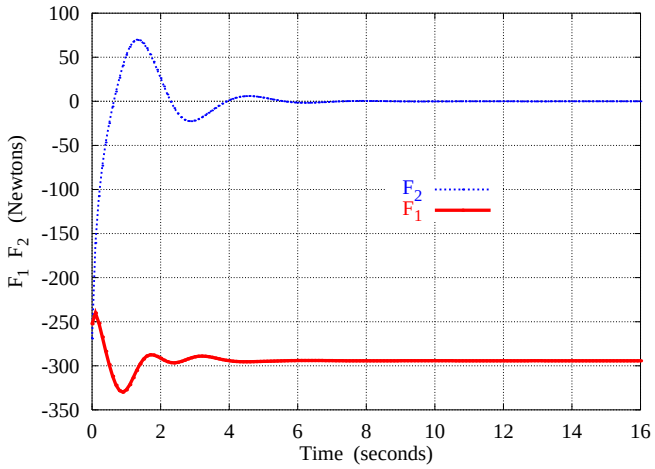


   Result:

$q_A(t{=}16) = -50.1492°$ $\qquad$ $q_B(t{=}16) = 35.1548°$

The set of contact forces exerted by N on A across the revolute joint at O is equivalent to a couple of torque $T_1 * \mathbf{n}_1 + T_2 * \mathbf{n}_2$ together with a force $F_1 * \mathbf{n}_1 + F_2 * \mathbf{n}_2 + F_3 * \mathbf{n}_3$ applied at O. Plot F_1 and F_2 for $0 \leq t \leq 16$ sec. Note: To find F_1 and F_2, modify the file `tutor6.al` as follows:

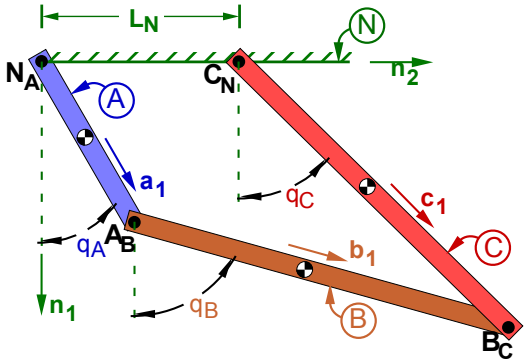
- Declare auxiliary generalized speeds with
  `MotionVariables' u3', u4'`
- Change the velocity of point O in N to
  `v_O_N> = u3*N1> + u4*N2>`
- Uncomment the lines:
  `Auxiliary[1] = DOT(V_O_N>, N1>)`
  `Auxiliary[2] = DOT(V_O_N>, N2>)`
  `Constrain( Auxiliary[u3,u4] )`
- Change the `Kane` command to
  `Kane( F1, F2 )`
- Uncomment the line
  `Output t sec, F1 newtons, F2 newtons`




```
%     File:  tutor6.al    [Autolev: http://www.autolev.com]
%  Problem:  Spring-damper double pendulum - Motion and contact forces
%----------------------------------------------------------------------
%       Newtonian, bodies, frames, particles, points
Newtonian  N
Frames     A, B
Particles  P, Q
Points     O
%----------------------------------------------------------------------
%       Variables, constants, and specified
%MotionVariables' u3', u4'  % Auxiliary generalized speeds
MotionVariables' qA'', qB'' % qA' and qB' are generalized speeds
Constants        LA, LB     % Lengths of A, B
Constants        k, Lo      % Spring constant, natural length of spring
Constants        g          % Local gravitational acceleration
Constants        b, c       % Damping constants
Variables        F1, F2     % Contact forces
Specified        Stretch'   % Elongation of spring joining O and Q
%----------------------------------------------------------------------
%       Mass
Mass  P=mP, Q=mQ
%----------------------------------------------------------------------
%       Geometry relating unit vectors
Simprot( N, A, 3, qA )
Simprot( N, B, 3, qB )
%----------------------------------------------------------------------
%       Position vectors
P_O_P> = LA*A1>
P_P_Q> = LB*B1>
%----------------------------------------------------------------------
%       Angular velocities
w_A_N> = qA'*A3>
w_B_N> = qB'*B3>
```

```
%-----------------------------------------------------------------------
%       Velocities
v_O_N> = 0>      % = u3*N1> + u4*N2>   % Brings F1 and F2 into evidence
v2pts( N, A, O, P )
v2pts( N, B, P, Q )
%-----------------------------------------------------------------------
%       Motion constraints
% Auxiliary[1] = Dot( v_O_N>, N1> )    % Dot( v_O_N>, N1> ) = 0
% Auxiliary[2] = Dot( v_O_N>, N2> )    % Dot( v_O_N>, N2> ) = 0
% Constrain( Auxiliary[u3,u4] )        % Solves for u3,u4
%-----------------------------------------------------------------------
%       Forces
Gravity( g*N1> )
LOQ = Mag( P_O_Q> )                        % Distance from O to Q
Stretch = LOQ - Lo                         % Stretch of spring
Uvec> = P_O_Q> / LOQ                       % Unit vector from O to Q
Stretch' = Dot( v_Q_N>, Uvec> )            % Time rate of change of stretch
Force(O/Q, (-k*Stretch-b*Stretch')*Uvec> ) % Spring/damper force
Force_O> += F1*N1> + F2*N2>                % Contact force on A from N at O
%-----------------------------------------------------------------------
%       Torques
Torque_A> += -c*qA'*A3>
Torque_B> += -c*qB'*B3>
%-----------------------------------------------------------------------
%       Equations of motion
Zero = Fr() + FrStar()
Zero := -Arrange( Zero, 1, g )    % Simplify slightly
Kane()
%Kane(F1,F2)                      % Solve for F1, F2
%-----------------------------------------------------------------------
%       Units system for CODE input/output conversions
UnitSystem  kg,meter,sec
%-----------------------------------------------------------------------
%       Integration parameters and values for constants and variables
Input  tFinal=16, integStp=0.1, absErr=1.0E-07, relErr=1.0E-07
Input  LA=1 m, LB=2 m,  k=200 n/m, Lo=1 m
Input  g=9.81 m/sec^2,  mP=10 kg,  mQ=20 kg
Input  b=100 n*sec/m, c=100 n*m*sec/rad
Input  qA=20 deg, qB=60 deg            % Initial values
Input  qA'=0 rad/sec, qB'=0  rad/sec   % Initial values
%-----------------------------------------------------------------------
%       Quantities to be output from CODE
EnergyCheck = NiCheck()                % Checking function
Output t sec, qA deg, qB deg, EnergyCheck joules
%Output t sec, F1 newtons, F2 newtons
%-----------------------------------------------------------------------
%       C code generation for numerical solution
CODE ODE(Zero,qA'',qB'') tutor6.c
%-----------------------------------------------------------------------
%       Record Autolev responses
Save tutor6.all
```

5.7 Four-Bar Linkage – Motion and Contact Forces



The figure to the right shows a planar four-bar linkage consisting of uniform rigid links A, B, and C and ground N. Link A is connected with revolute joints to N and B at points N_A and A_B, respectively. Link C is connected with revolute joints to N and B at points C_N and B_C, respectively.

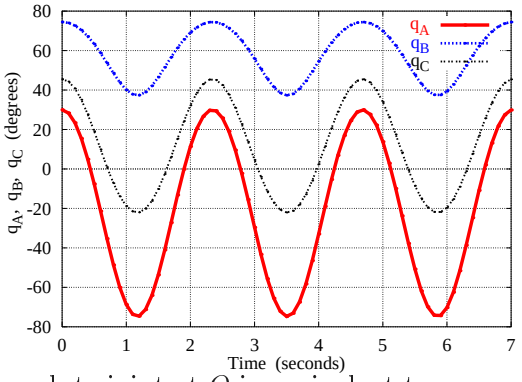
Right-handed orthogonal unit vectors $\mathbf{a}_i$, $\mathbf{b}_i$, $\mathbf{c}_i$, and $\mathbf{n}_i$ (i=1,2,3) are fixed in A, B, C, and N, with $\mathbf{a}_1$ directed from N_A to A_B, $\mathbf{b}_1$ from A_B to B_C, $\mathbf{c}_1$ from C_N to B_C, $\mathbf{n}_1$ vertically downward, $\mathbf{n}_2$ from N_A to C_N, and $\mathbf{a}_3 = \mathbf{b}_3 = \mathbf{c}_3 = \mathbf{n}_3$ parallel to the revolute joints' axes.

The independent motion variable is $\dot{q}_A$ and the dependent motion variables are $\dot{q}_B$ and $\dot{q}_C$. Other symbols useful in this analysis are shown to the right.

Using the initial values $q_A = 30°$ and $\dot{q}_A = 0$, plot the angles q_A, q_B, and q_C for $0 \le t \le 7$ sec.

Note: The values $q_B = 74.47751219°$ and $q_C = 45.52248781°$ satisfy the configuration constraint equations when $q_A = 30°$.

Quantity	Identifier	Value
Distance from N_A to A_B	L_A	1 m
Distance from A_B to B_C	L_B	2 m
Distance from B_C to C_N	L_C	2 m
Distance from C_N to N_A	L_N	1 m
Mass of A	m^A	10 kg
Mass of B	m^B	20 kg
Mass of C	m^C	20 kg
Angle between $\mathbf{n}_1$ and $\mathbf{a}_1$	q_A	variable
Angle between $\mathbf{n}_1$ and $\mathbf{b}_1$	q_B	variable
Angle between $\mathbf{n}_1$ and $\mathbf{c}_1$	q_C	variable



The set of contact forces exerted by N on A across the revolute joint at O is equivalent to a couple of torque $T_1^A \mathbf{n}_1 + T_2^A \mathbf{n}_2$ together with a force $F_1^A \mathbf{n}_1 + F_2^A \mathbf{n}_2 + F_3^A \mathbf{n}_3$ applied at O. Similarly, the set of contact forces exerted by N on C across the revolute joint at R is equivalent to a couple of torque $T_1^C \mathbf{n}_1 + T_2^C \mathbf{n}_2$ together with a force $F_1^C \mathbf{n}_1 + F_2^C \mathbf{n}_2 + F_3^C \mathbf{n}_3$ applied at R.

- Form a numerical integration checking function and verify that it remains constant during numerical integration. Verify that the configuration constraint equations, which also serve as numerical integration checking functions, are satisfied during numerical integration.

Result:
See the file `tutor7.2` for the time-histories of `Config[1]`, `Config[2]`, and `ECheck`.

- Plot the time histories of F_1^C and F_2^C for $0 \leq t \leq 10$ sec

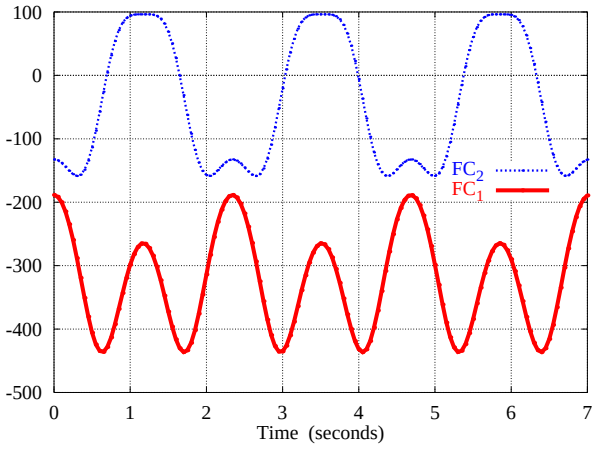
  Note: To find F_1^C and F_2^C, modify the file `tutor7.al` as follows:

Change these lines	To these lines
`Dependent[1] = Dot( V_R_N>, N1> )`	`Auxiliary[1] = Dot( V_R_N>, N1> )`
`Dependent[2] = Dot( V_R_N>, N2> )`	`Auxiliary[2] = Dot( V_R_N>, N2> )`
`Constrain( Dependent[qB',qC'] )`	`Constrain( Auxiliary[qB',qC'] )`
`Kane()`	`Kane( FC1, FC2 )`
Uncomment these lines	`Force_R> = FC1*N1> + FC2*N2>`
	`Output T, FC1, FC2`

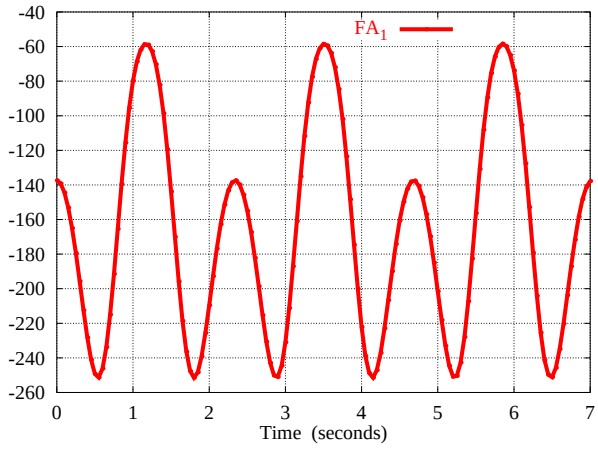
- Plot the time-history of F_1^A for $0 \leq t \leq 10$ sec.

  Note: To find F_1^A, modify the file `tutor7.al` once more.

Uncomment the declaration of the auxiliary speeds	`MotionVariables' vx'`
Change the velocity of point `O` in `N` to	`V_O_N> = vx*N1>`
Uncomment the line	`Auxiliary[3] = Dot( V_O_N>, N1> )`
Change the `Constrain` command to	`Constrain( Auxiliary[qB',qC',vx] )`
Change the `KANE` command to	`Kane( FC1, FC2, FA1 )`
Uncomment the line	`Output t, FA1`



Time-histories of F_1^C and F_2^C

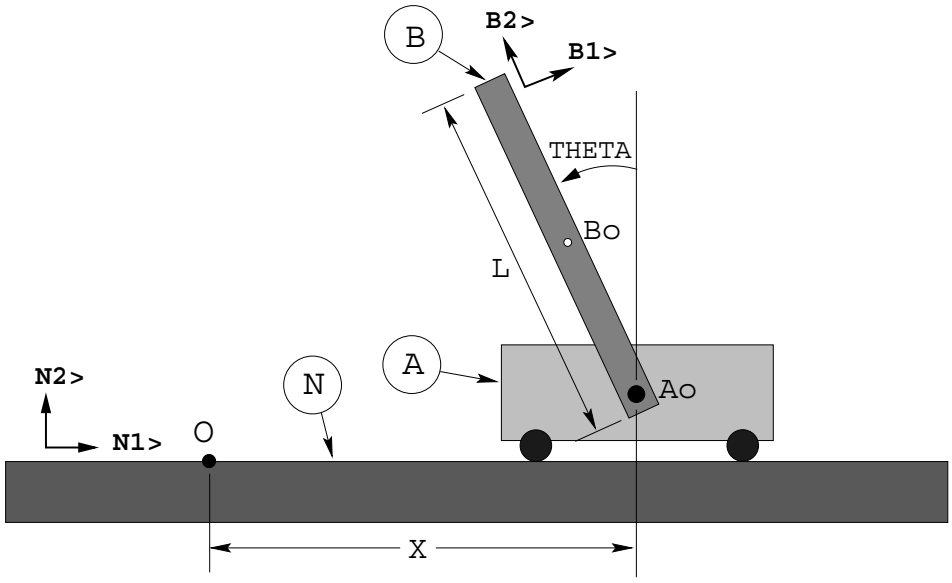


Time-history of F_1^A

```
%     File:  tutor7.al    [Autolev: http://www.autolev.com]
%  Problem:  Four-Bar Linkage -- Motion and Contact Forces
%----------------------------------------------------------------------
%       Default Settings
AutoZ        ON              % Program introduces Zees automatically
Digits       7               % Significant digits
%----------------------------------------------------------------------
%       Newtonian, bodies, frames, particles, points
Newtonian    N
Bodies       A, B, C
Points       NA, AB, BC, CN
%----------------------------------------------------------------------
%       Variables, constants, and specified
MotionVariables'  qA'', qB'', qC''     % qA', qB', qC' are generalized speeds
Variables         FA{2}, FC{2}         % Contact forces
Constants         g=9.81                 % Gravitational acceleration
Constants         LA=1, LB=2, LC=2, LN=1 % Lengths
ZEE_NOT = [FA1, FA2, FC1, FC2]
%----------------------------------------------------------------------
%       Mass and inertia
Mass         A=mA=10, B=mB=20, C=mC=20
Inertia      A, 0, mA*LA^2/12, mA*LA^2/12
Inertia      B, 0, mB*LB^2/12, mB*LB^2/12
Inertia      C, 0, mC*LC^2/12, mC*LC^2/12
%----------------------------------------------------------------------
%       Geometry relating unit vectors
Simprot(N, A, 3, qA)
Simprot(N, B, 3, qB)
Simprot(N, C, 3, qC)
%----------------------------------------------------------------------
%       Position vectors
P_NA_AB> = LA*A1>
P_AB_BC> = LB*B1>
P_CN_BC> = LC*C1>
P_NA_CN> = LN*N2>
P_NA_Ao> = 0.5*P_NA_AB>
P_AB_Bo> = 0.5*P_AB_BC>
P_CN_Co> = 0.5*P_CN_BC>
%----------------------------------------------------------------------
%       Angular velocities
W_A_N> = qA'*A3>
W_B_N> = qB'*B3>
W_C_N> = qC'*C3>
%----------------------------------------------------------------------
%       Velocities
% MotionVariables'  vx'
% V_NA_N> = vx*N1>
V_NA_N> = 0>
V2pts(N, A, NA, Ao)
V2pts(N, A, NA, AB)
V2pts(N, B, AB, Bo)
V2pts(N, B, AB, BC)
V2pts(N, C, BC, Co)
V2pts(N, C, BC, CN)
%----------------------------------------------------------------------
%       Motion constraints
Dependent[1] = Dot( V_CN_N>, N1> )      % Dot( V_CN_N>, N1> ) = 0
Dependent[2] = Dot( V_CN_N>, N2> )      % Dot( V_CN_N>, N2> ) = 0
Constrain( Dependent[qB',qC'] )        % Solve for qB', qC'
% Auxiliary[3] = Dot( V_NA_N>, N1> )    % Dot( V_NA_N>, N1> ) = 0
% Constrain( Auxiliary[qB',qC',vx] )   % Solve for qB', qC', vx
%----------------------------------------------------------------------
%       Angular accelerations
ALF_A_N> = Dt( W_A_N>, N )
```

```
ALF_B_N> = Dt( W_B_N>, N )
ALF_C_N> = Dt( W_C_N>, N )
%-------------------------------------------------------------------
%       Accelerations of particles and mass centers of bodies
A_NA_N> = 0>
A2pts(N, A, NA, Ao)
A2pts(N, A, NA, AB)
A2pts(N, B, AB, Bo)
A2pts(N, B, AB, BC)
A2pts(N, C, BC, Co)
%-------------------------------------------------------------------
%       Forces
Gravity( g*N1> )
Force_NA> = FA1*N1> + FA2*N2>
% Force_CN> = FC1*N1> + FC2*N2>
%-------------------------------------------------------------------
%       Equations of motion
Zero = Fr() + FrStar()
Kane( )
% Kane( FC1, FC2, FA1 )
%-------------------------------------------------------------------
%       Units system for CODE input/output conversions
UnitSystem  kg,meter,sec
%-------------------------------------------------------------------
%       Integration parameters and values for constants and variables
Input  tFinal=7, integStp=0.05, absErr=1.0E-07, relErr=1.0E-07
Input  qA=30 deg, qB=74.47751219 deg, qC=45.52248781 deg % Initial values
%-------------------------------------------------------------------
%       Quantities to be output from CODE
LOOP> = P_NA_AB> + P_AB_BC> + P_BC_CN> + P_CN_NA>
Config[1] = Dot( Loop>, N1> )          % Should always equal 0
Config[2] = Dot( Loop>, N2> )          % Should always equal 0
ECheck = NiCheck()                     % Checking function
Output t, qA deg, qB deg, qC deg
Output t, Config[1], Config[2], ECheck
% Output t, FC1 newtons, FC2 newtons
% Output t, FA1 newtons
%-------------------------------------------------------------------
%       Fortran code generation for numerical solution
CODE Dynamics() tutor7.for
%-------------------------------------------------------------------
%       Record Autolev responses
Save tutor7.all
```

5.8 Dynamics of a Cart Carrying an Inverted Pendulum



The sketch shows a thin uniform rod B attached to a cart A which slides on a smooth horizontal plane fixed in a Newtonian reference frame N. Right-handed sets of mutually perpendicular unit vectors Ni> and Bi> (i=1,2,3) are fixed in N and B, as shown, with N2> vertically upward, B2> parallel to the long axis of B, and N3> = B3> all parallel to the axis of the frictionless revolute joint which joins A and B at point Ao, the mass center of A. The length of B is L, and the masses of A and B are MA and MB, respectively. x is the distance from a point O fixed in N to Ao, and the angle theta characterizes the orientation of B in N, as shown. The generalized speeds are x' and theta'.

To control the cart, a force F*N1> is applied to Ao. A state-space controller is used so that F is specified in terms of constant feedback gains K_i (i=1,2,3,4) as F = k1*x + k2*theta + k3*U1 + k4*U2.

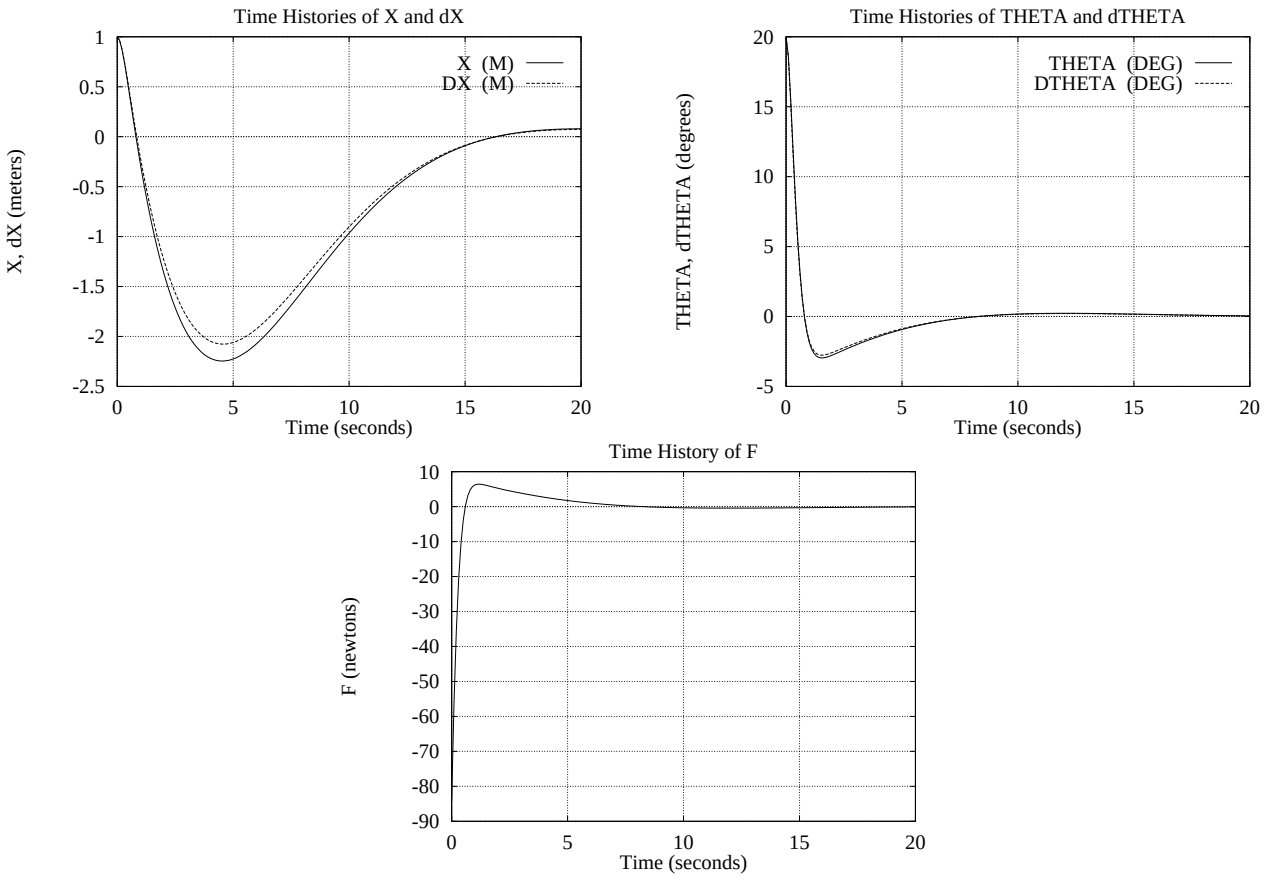
- Form equations of motion for the system.
  Result: The output from the KANE command is
  ```
  ZERO[1] = F + 0.5*L*MB*COS(theta)*U2' - 0.5*L*MB*SIN(theta)*U2^2 - (MA+MB)*U1'
  ZERO[2] = 0.5*G*L*MB*SIN(theta) + 0.5*L*MB*COS(theta)*U1' - 0.25*(4*IB+MB*L^2)*U2'
  ```
- For L=1, MA=10, MB=1, G=9.81, show that x=theta=U1=U2=U1'=U2'=0 is a solution of the equations of motion.
  Result: The output from CHECK = evaluate(ZERO, x=0, theta=0, U1=0, U2=0, U1'=0, U2'=0) is CHECK = [0; 0]
- After introducing the variables dx, dtheta, dU1, and dU2 as perturbations of x, theta, U1, and U2, respectively, linearize the kinematical and dynamical equations in the perturbations about the nominal solution x=theta=U1=U2=U1'=U2'=0. Put the linearized equations in the form x' = A*x where A is a 4x4 coefficient matrix and x is the 4x1 state matrix [dx; dtheta; dU1; dU2].
  Result:
  ```
  A = [       0,              0,         1,         0;
  ```

```
        0,                0,          0,          1;
0.098*k1,  0.72 + 0.098*k2, 0.098*k3,  0.098*k4;
 0.15*k1,  16.0 +  0.15*k2,  0.15*k3,   0.15*k4 ]
```

- Show that if `k1=k2=k3=k4=0`, then `x=theta=U1=U2=U1'=U2'=0` is an unstable solution.
  Result: The output `ROOTS1 = EIG( evaluate(A, k1=0, k2=0, k3=0, k4=0) )` is
  `ROOTS1 = [-4; 0; 0; 4]`
  Since one of the eigenvalues is positive, the solution is unstable.

- Show that if `k1=1, k2=-244, k3=5.4, k4=-59`, then `x=theta=U1=U2=U1'=U2'=0` is a stable solution.
  Result: The output from `ROOTS2 = EIG( evaluate(A, k1=1, k2=-244, k3=5.4, k4=-59) )` is
  `ROOTS2 = [-3.8 - 1.3*i; -3.8 + 1.3*i; -0.22 - 0.2*i; -0.22 + 0.2*i]`
  Since all of the eigenvalues have negative real parts, the solution is stable for "small" disturbances.

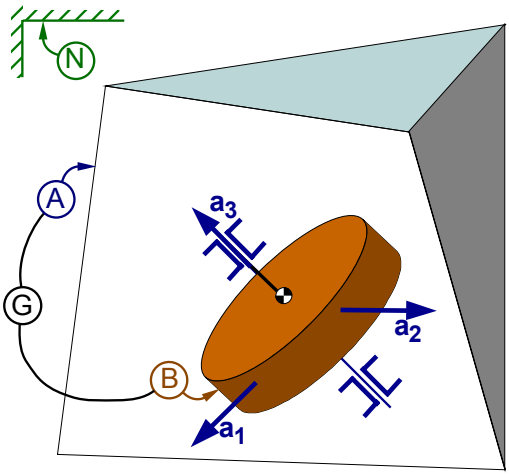
- Using the linearized and nonlinear equations of motion, simulate the motion of the system for `k1=1`, `k2=-244`, `k3=5.4`, `k4=-59`. Use the initial values `x=dx=1 m, theta=dtheta=20°, U1=dU1=0, U2=dU2=0,` and simulate from `T=0` to `T=20` sec. Plot `x,dx,theta,dtheta vs. time` and `F vs. time`.
  Result: See the plots.



```
%     File:  tutor8.al    [Autolev: http://www.autolev.com]
%  Problem:  Dynamics of a Cart Carrying an Inverted Pendulum
%--------------------------------------------------------------------
%       Default Settings
Digits       2               % Significant digits
%--------------------------------------------------------------------
%       Newtonian, bodies, frames, particles, points
Newtonian    N               % Newtonian reference frame
Bodies       A, B            % Cart, inverted pendulum
Points       O               % Point fixed in N
%--------------------------------------------------------------------
%       Variables, constants, and specified
MotionVariables' x'', theta'' % x' and theta' are motion variables
Constants        L            % Length of rod
Constants        g            % Gravitational acceleration
Constants        k{4}         % Control gains
Specified        F = k1*x + k2*theta + K3*x' + K4*theta'
%--------------------------------------------------------------------
%       Mass and inertia
Mass         A=MA, B=MB
Inertia      B, IB=MB*L^2/12, 0, IB
%--------------------------------------------------------------------
%       Geometry relating unit vectors
Simprot(N, B, 3, theta)
%--------------------------------------------------------------------
%       Position vectors
P_O_Ao> = x*N1>
P_Ao_Bo> = 0.5*L*B2>
%--------------------------------------------------------------------
%       Angular velocities
W_A_N> = 0>
W_B_N> = theta'*B3>
%--------------------------------------------------------------------
%       Velocities
V_Ao_N> = Dt( P_O_Ao>, N )
v2pts(N, B, Ao, Bo)
%--------------------------------------------------------------------
%       Forces
Gravity( -g*N2> )
Force_Ao> += F*N1>
%--------------------------------------------------------------------
%       Equations of motion
Zero = Fr() + FrStar()
Kane()
Zero := Evaluate( Zero, L=1, MA=10, MB=1, g=9.81 )
%********************************************************************
%       CONTROL SYSTEM / STABILITY ANALYSIS
%********************************************************************
%       Linearization: Perturbation variables
Variables  dx''            % Perturbations of x, x', x''
Variables  dtheta''        % Perturbations of theta, theta', theta''
Imaginary  i               % Imaginary number
%--------------------------------------------------------------------
%       Check whether x=x'=x''=0 and theta=theta'=theta''=0 is a solution
Check = Evaluate( Zero, x=0, x'=0, x''=0, theta=0, theta'=0, theta''=0 )
%--------------------------------------------------------------------
%       Linearize equations of motion about nominal solution
```

```
Perturb = Taylor( Zero, 1, x=0:dx, x'=0:dx', x''=0:dx'', &
                  theta=0:dtheta, theta'=0:dtheta', theta''=0:dtheta'' )
Solve( Perturb, dx'', dtheta'' )
%---------------------------------------------------------------------
%       Form matrix of peturbations and its time-derivative
Xm = [ dx ;  dtheta ;  dx' ;  dtheta' ]    % Matrix of perturbations
Xp = [ dx';  dtheta';  dx'';  dtheta'']    % Time derivative of Xm
%---------------------------------------------------------------------
%       Form matrix A such that Xm' = A * Xm
A = D( Xp, Transpose(Xm) )
%---------------------------------------------------------------------
%       Stability when F=0
Roots1 = Eig( evaluate(A, k1=0, k2=0, k3=0, k4=0) )
%---------------------------------------------------------------------
%       Stability when F = x - 244*theta + 5.4*x' - 59*theta'
Roots2 = Eig( evaluate(A, k1=1, k2=-244, k3=5.4, k4=-59) )
%---------------------------------------------------------------------
%       Units system for CODE input/output conversions
UnitSystem  kg,meter,sec
%---------------------------------------------------------------------
%       Integration parameters and values for constants and variables
Input  tFinal=20, integStp=0.1, absErr=1.0E-08, relErr=1.0E-08
Input  k1=1,  k2=-244,  k3=5.4,  k4=-59
Input   x=1 m,  theta=20 deg,  x'=0 m/sec,  theta'=0 rad/sec
Input  dx=1 m, dtheta=20 deg, dx'=0 m/sec, dtheta'=0 rad/sec
%---------------------------------------------------------------------
%       Quantities to be output from CODE
Output t sec, x meters,  dx meters,  theta deg,  dtheta deg, F newtons
%---------------------------------------------------------------------
%       C code generation for numerical solution
Digits  5                       % Significant digits in output files
CODE Dynamics() tutor8.c
%---------------------------------------------------------------------
%       Record Autolev responses
Save tutor8.all
```

5.9 Spin Stabilization of a Gyrostat



The sketch to the right a gyrostat `G` consisting of a carrier A and a thin uniform cylindrical rotor B moving in a Newtonian frame N. Dextral sets of mutually perpendicular unit vectors $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$ are fixed in A and are parallel to central principal axes of G. The rotor B has a central moment of inertia of J about its symmetric axes, which is parallel to $\mathbf{a}_3$. The central principal moments of inertia of G for $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$ are denoted $I1$, $I2$, $I3$, respectively. The generalized speeds ω_i (i=1,2,3) are the $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$ measure numbers of the angular velocity of A in N, and the constant Ω is the $\mathbf{a}_3$ measure number of the angular velocity of B in A.

- Form equations of motion which govern angular motions of the system.
  Result: The output from the `Kane` command is

```
Zero[1] = -w2*(J*Omega-(I2-I3)*w3) - I1*w1'
Zero[2] = w1*(J*Omega-(I1-I3)*w3) - I2*w2'
Zero[3] = (I1-I2)*w1*w2 - I3*w3'
```

- After introducing the constant `nw3`, show that `w1=w2=w1'=w2'=w3'=0, w3=nw3` is a solution of the equations of motion.
  Result: The output from `Check = Evaluate(Zero, w1=0, w1'=0, w2=0, w2'=0, w3=nw3, w3'=0)` is `Check = [0; 0; 0]`

- After introducing the variables `dUi` and `dUi'` as perturbations of `Ui` and `Ui'` (i=1,2,3), respectively, linearize the equations of motion in the perturbations about the nominal solution `w1=w2=w1'=w2'=w3'=0, w3=nw3`. Put the linear equations in the form `x' = A*x` where `A` is a 3x3 coefficient matrix and `x` is the 3x1 state matrix `[dw1; dw2; dw3]`.
  Result:

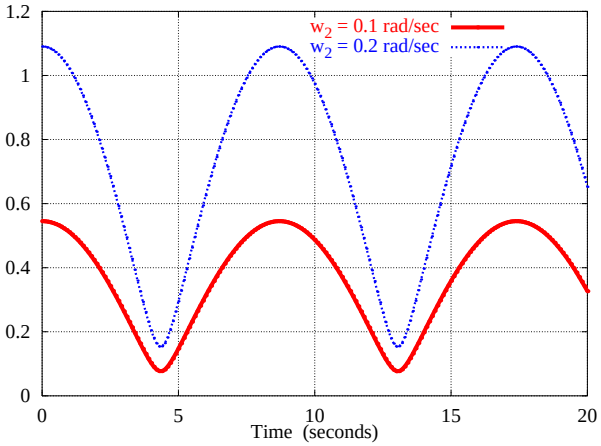
```
A = [                        0, -(J*Omega-(I2-I3)*Nw3)/I1,                0;
     (J*Omega-(I1-I3)*Nw3)/I2,                         0,                0;
                              0,                         0,                0 ]
```

- With `J=0.07634`, `I1=1.25`, `I2=4.25`, `I3=5`, `nw3=1`, determine values of `Omega` which result in `LAMBDA`, the eigenvalues of `A` to be positive.
  Result: The Autolev output response to the last line of the file `tutor9.al` is

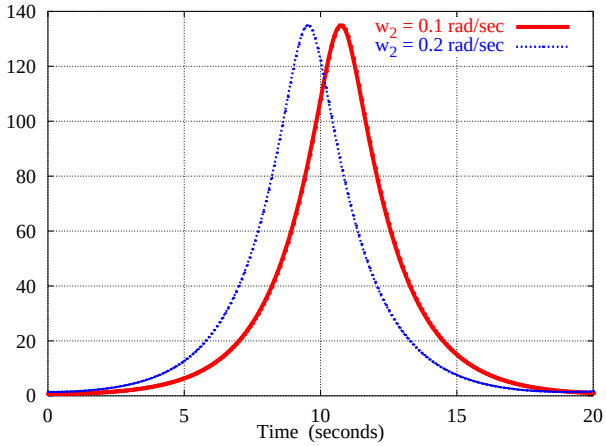
```
DET = 0.001096997*(9.824469+Omega)*(49.12235+Omega) + LAMBDA^2
```

  This shows that `LAMBDA`^{2} is positive when `-49.12 < Omega < -9.82` . When the real part of `LAMBDA` is positive, the system is said to be "unstable". When the real part of `LAMBDA` is 0, the system is said to be "neutrally stable". Hence, the gyro is neutrally stable for `Omega > -9.82` and for `Omega < -49.12`.

- Using the nonlinear equations of motion, run four numerical motion simulations for `0` $\leq$ `t` $\leq$ `20 sec`.
  In all four simulations, set `J=0.07634`, `I1=1.25`, `I2=4.25`, `I3=5`, `w1=0`, and `w3=1`.
  In the first, set `Omega=-7` and `w2=.02`. In the second, set `Omega=-7` and `w2=.01`.
  In the third, set `Omega=-20` and `w2=.02`. In the fourth, set `Omega=-20` and `w2=.01`.
  Plot the time-history of ϕ, the angle between $\mathbf{a}_3$ and the inertial angular momentum of G. For each simulation, check that H, the magnitude of the inertial angular momentum of the gyrostat, is time-invariant.
  Result: See the plots. By examining the simulation output file `tutor9.1`, it is clear that `H` is time-invariant.



Time-histories of ϕ when Ω= -7 $\frac{\text{rad}}{\text{sec}}$



Time-histories of ϕ when Ω= -20 $\frac{\text{rad}}{\text{sec}}$

```
%     File:  tutor9.al    [Autolev: http://www.autolev.com]
%  Problem:  Spin stability of gyrostat
%--------------------------------------------------------------------
%       Newtonian, bodies, frames, particles, points
Newtonian  N              % Newtonian reference frame
Bodies     A              % Carrier
Frames     B              % Rotor
%--------------------------------------------------------------------
%       Variables, constants, and specified
MotionVariables' w{3}'    % Generalized speeds; derivatives
Constants        Omega    % Angular speed of B in A
Constants        J        % Moment of inertia of B about spin axis
%--------------------------------------------------------------------
%       Mass and inertia
Mass       A=M            % Attribute mass of G to A
Inertia    A, I1, I2, I3  % Attribute inertia of G to A
%--------------------------------------------------------------------
%       Angular velocities
W_A_N> = Vector( A, w1, w2, w3 )
W_B_A> = Omega*A3>
%--------------------------------------------------------------------
%       Angular accelerations
ALF_A_N> = DT( w_A_N>, N )
ALF_B_A> = DT( w_B_A>, A )
%--------------------------------------------------------------------
%       Acceleration of mass center of G is 0>
A_Ao_N> = 0>
%--------------------------------------------------------------------
%       Equations of motion
Zero = Fr() + FrStar() + Gyrostat(FrStar,CYLINDER,A,B,J)
Kane()
%--------------------------------------------------------------------
%       Angular momentum of gyrostat
H> = Momentum(Angular, Ao) + Gyrostat(Angmom,CYLINDER,A,B,J)
H = Mag( H> )
%--------------------------------------------------------------------
%       Angle between angular momentum vector and A3>
Phi = acos( Dot( UnitVec(H>), A3>) )
%--------------------------------------------------------------------
%       Units system for CODE input/output conversions
UnitSystem  kg,meter,sec
%--------------------------------------------------------------------
%       Integration parameters and values for constants and variables
Input  tFinal=20, integStp=0.1, absErr=1.0E-07, relErr=1.0E-07
Input  J=0.07634, I1=1.25, I2=4.25, I3=5
Input  Omega=-20.0, w1=0, w2=0.02, w3=1
%--------------------------------------------------------------------
%       Quantities to be output from CODE
Output  t sec, Phi degs, H kg*m^2/sec
```

```
%-----------------------------------------------------------------------
%       C code generation for numerical solution
CODE Dynamics() tutor9.c
%***********************************************************************
%       STABILITY ANALYSIS
%***********************************************************************
%       Linearization: Perturbation vars + nominal solution parameters
Variables  dw{3}'       % Perturbations of w1, w2, w3
Constants  nw3          % Nominal solution for w3
%-----------------------------------------------------------------------
%       Check nominal solution
Check = Evaluate(Zero, w1=0, w1'=0, w2=0, w2'=0, w3=nw3, w3'=0)
%-----------------------------------------------------------------------
%       Linearize equations of motion about nominal solution
Perturb = Taylor(Zero, 1, w1=0:dw1, w1'=0:dw1', w2=0:dw2, w2'=0:dw2', &
                          w3=nw3:dw3, w3'=0:dw3')
Solve( Perturb,  dw{1:3}' )
%-----------------------------------------------------------------------
%       Form, x, x', and the A matrix in the equation x'=A*x
Xm = [ dw1,  dw2,  dw3 ]
Xp = [ dw1'; dw2'; dw3']
Am = D( Xp, Xm )
%-----------------------------------------------------------------------
%       To find eigenvalues of Am symbolically, find the roots of the
%       equation found by setting the determinant of Lambda*I-A = 0
Variables Lambda
Det = Det( Lambda*Diagmat(3,1) - Am )
%-----------------------------------------------------------------------
%       Inspection of Det shows that Lambda=0 is a root
Det /= Lambda
%-----------------------------------------------------------------------
%       With J=0.07634, I1=1.25, I2=4.25, I3=5, nw3=1,
%       determine the values of Omega which result in Lambda > 0
Det := Evaluate( Det, J=0.07634, I1=1.25, I2=4.25, I3=5, nw3=1 )
%-----------------------------------------------------------------------
%       Record Autolev responses
Save tutor9.all
```

6 Other Information

6.1 Functions and Commands

Autolev is equipped with more than 100 commands. An alphabetical list of commands appears on the screen when one types `WHAT` at any Autolev line prompt. A Quick-Reference and a Summary of Commands are included at the end of the tutorial. In addition, a detailed description of a command appears on the screen when one types `HELP commandname` at any Autolev line prompt.

Most commands may be nested. For example, the `DOT` command may appear as an argument of the `ACOS` command, e.g., `ANGLE = ACOS( DOT(A1>,B1>) )`.

6.2 Stand-Alone Commands

In many commands, e.g., `RENEE = COS(X)`, one uses an equals sign to make an assignment. In some commands, called *stand-alone commands*, one makes assignments without using equals signs. For example, the equations

```
3*X + 4*Y = 37
4*X - 2*Y = -2
```

can be solved by executing the following Autolev input file

```
   (1) Variables X, Y
   (2) Zero[1] = -37 + 3*X + 4*Y
-> (3) Zero[1] = -37 + 3*X + 4*Y
   (4) Zero[2] = 2 + 4*X - 2*Y
-> (5) Zero[2] = 2 + 4*X - 2*Y
   (6) SOLVE( Zero, X,Y )
-> (7) X = 3
-> (8) Y = 7
```

In line 6, the command `SOLVE(Zero,X,Y)` causes values to be assigned to `X` and `Y`, but the command does not involve the typing of any equals sign. Some other stand-alone commands are `V2pts`, `V1pt`, `A2pts`, and `A1pt`.

6.3 Dual Functions

The commands `ARRANGE`, `EXPAND`, `EXPLICIT`, `EXPRESS`, `FACTOR`, and `ZEE` are called *dual functions* because they can be used in two ways, namely, on the right-hand side of an equals sign, e.g.,

NEWY = EXPLICIT(Y), or as stand-alone commands. When a dual-function appears on an Autolev input line in the form

```
DualFunctionName( X, list_of_arguments )
```

where DualFunctionName is one of the dual functions, X is the *name* of an expression, not the expression itself, and list_of_arguments stands for arguments of the dual function, then Autolev interprets this line as

```
X := DualFunctionName( X, list_of_arguments )
```

Dual functions differ from other commands in that they do not change the functional character of an expression, but alter its appearance.

6.4 Creating Your Own Commands: .A and .R Files

To add to Autolev's more than 100 commands, you can create new commands by creating an ASCII file that resides in the Autolev toolbox directory, the current working directory, or a directory that is specified in the AUTOLEVPATH. For example, suppose you wish to create a command called SUM, which adds two expressions and returns their sum. The syntax of the SUM command could be SUM(x,y), where x and y are expressions. To create the command SUM, use a text editor to compose a file named sum.r with the following contents:

```
%SUM.R
%
%Function:  Returns the sum of two expressions.
%
%  Syntax:  SUM(x,y)
%
#1# + #2#
```

The filename has a .r extension to inform Autolev that SUM returns a result (.r for result). The #1# and #2# denote the two arguments of SUM. The comments at the top of the file contain text that appears on the screen when HELP SUM is typed at the Autolev line prompt. Typing PLUS = SUM(3,5) results in PLUS = 8 .

Next, suppose you want to create a command which makes assignments. For example, to create a command called POWER, which squares and cubes an expression and assigns the results to new variables called *name*SQ and *name*CUBE, compose a file named power.a with the following contents:

```
%POWER.A
%
%Function:  Put explanatory information about POWER here.
%
%Syntax:    POWER(expression,name)
%
#2#SQ = (#1#)^2
#2#CUBE = (#1#)^3
```

The filename has a .a extension to inform Autolev that POWER makes assignments (.a for assignment). Note the use of parentheses around the #1# argument. Liberal use of parentheses is helpful in making bug-free .a and .r files. To produce efficient results when AUTOZ is ON, use the AUTOZ() command which introduces intermediate symbols. Use of the POWER command is demonstrated below.

```
   (1) POWER(3,A)
 ->(2) ASQ = 9
 ->(3) ACUBE = 27
   (4) Constants A,B
   (5) POWER(A+B,ED)
 ->(6) EDSQ = (A+B)^2
 ->(7) EDCUBE = (A+B)^3
```

There are three special symbols that are helpful in creating .a and .r files. The symbol #NUM_ARGS# evaluates to the number of arguments passed to the .a or .r command; #Newtonian# evaluates to the name declared in the Newtonian declaration; and #DEGREES# evaluates to ON or OFF, depending on the current setting of DEGREES. In addition, one may use the logical if and else statements, the ECHO command, and the special symbols \k, \a, \p, \n that are used with ECHO (type HELP ECHO for more information). For example,

```
%SUM.R
%
%Function:  Returns the sum of two expressions.
%
%  Syntax:  SUM(x,y)
%
if( #NUM_ARGS# != 2 )
   { ECHO(\k\a"Error: wrong number of arguments to the SUM command"\p\n); }
else
   { #1# + #2# }
```

6.5 Default Settings

The manner in which Autolev responds to online commands or when executing a batch file depends on certain "settings". For example, the factoring of expressions depends on whether FACTORING is ON or OFF, as seen by comparing the two Autolev sessions below:

```
     (1) FACTORING OFF                    (1) FACTORING ON
     (2) Constants A,B,C                  (2) Constants A,B,C
     (3) X = A*B + A*C + A^2              (3) X = A*B + A*C + A^2
  -> (4) X = A*B + A*C + A^2           -> (4) X = A*(A+B+C)
     (5) Y = A/C + B/C                    (5) Y = A/C + B/C
  -> (6) Y = A/C + B/C                 -> (6) Y = (A+B)/C
```

AUTORHS is another example of a settting. AUTORHS may be set to ALL, ON, or OFF. With AUTORHS set to ALL, the right-hand side of all quantities are automatically substituted for the quantity. With AUTORHS ON, substitutions are made when the right-hand side of quantities are "simple". With AUTORHS OFF, no substitutions are made. One way to see the effect of AUTORHS is to compare the files below.

```
   (1) AUTORHS  OFF          (1) AUTORHS  ON           (1) AUTORHS  ALL
   (2) Constants B           (2) Constants B           (2) Constants B
   (3) A = 7                 (3) A = 7                 (3) A = 7
-> (4) A = 7              -> (4) A = 7              -> (4) A = 7
   (5) C = A + B             (5) C = A + B             (5) C = A + B
-> (6) C = A + B          -> (6) C = 7 + B          -> (6) C = 7 + B
   (7) D = C*SIN(C)          (7) D = C*SIN(C)          (7) D = C*SIN(C)
-> (8) D = C*SIN(C)       -> (8) D = C*SIN(C)       -> (8) D = (7+B)*SIN(7+B)
```

Whenever Autolev is invoked, several settings are assigned default values in the file `DEFAULTS.AL`. The user can employ a text editor to change each of the values in this file. For example, `ON` can be replaced with `OFF` as the value assigned to `FACTORING`. The value assigned to most of the settings in the file `DEFAULTS.AL` can be changed online. For example, if the value assigned in `DEFAULTS.AL` to `AUTORHS` is `ON`, then entering the line `AUTORHS OFF` changes the setting from `ON` to `OFF` in the current Autolev session. Such changes can be made repeatedly. However, changes online do not alter the file `DEFAULTS.AL`.

During an Autolev session, one can review the values assigned to all settings by typing `DEFAULTS` at any line prompt. For help with a particular setting, type `HELP SettingName`, e.g., `HELP FACTORING`.

Note: To accommodate multi-user computers, Autolev checks the current directory for the file `DEFAULTS.AL` before checking the directories specified in the environment variable AUTOLEVPATH or the `TOOLBOX` directory or the directories specified in the environment variable PATH.

6.6 Special Symbols (see also Reserved Names in Section 2.4)

%	Comment delimiter. Input following this character is ignored
%%	Inserts a comment in MATLAB, C, or FORTRAN code
&	Line continuation character (Autolev input files only)
' (prime)	Implies total differentiation with respect to T
>	The last symbol in the name of a vector
>>	The last two symbols in the name of a dyadic
>>>	The last three symbols in the name of a triadic or higher order polyadic
[ ]	Used to denote a matrix or designate an element of a matrix
{ }	Encloses indices in declarations
,	Separates arguments of a function and elements of a row in a matrix
;	Separates rows of a matrix
:	Designates a range, e.g., 1:3
()	Encloses mathematical expressions and function arguments
#	Delimits arguments in .A and .R files
"	Delimits string literals
.	Decimal point
+ - * / ^	Mathematical operators: addition, subtraction, multiplication, division, and exponentiation
= := += -=	Assignment operators: normal, overwrite, addition, and subtraction
*= /= ^ =	Assignment operators: multiplication, division, and exponentiation

6.7 Editing Keys for Online Editing with a Windows PC

HOME	Moves cursor to the beginning of input line
END	Moves cursor to the end of input line
DEL	Deletes current character in line
BACKSPACE	Deletes previous character in line
RIGHT ARROW →	Moves cursor one space to the right
LEFT ARROW ←	Moves cursor one space to the left
UP ARROW ↑	Recovers previous line
INSERT	Toggles between insert and overstrike editing modes
Control-C	Abruptly terminates program execution

7 Summary of Commands

Interfacing with Autolev and the operating system

`al`	Invokes Autolev from the operating system prompt
`al filename`	Invokes Autolev from the operating system prompt and executes the commands in `filename`
`CLEAR`	Deletes all lines from the workspace and restarts Autolev at line (1)
`CLEAR k`	Deletes lines k and all lines following line k from the workspace and reruns Autolev from line(1) to line (k-1)
`CONTROL-C`	Terminates program execution
`ECHO`	Enables information to be transmitted to the user during execution of user-created .a and/or .r files
`EXIT or QUIT`	Ends Autolev session
`HELP`	Displays general information about Autolev and OnLine Dynamics, Inc.
`HELP commandname`	Displays information on commands, declarations, and defaults
`HELP syntax_forms`	Displays information on syntactical forms
`HELP update`	Displays information on new commands, declarations, and defaults
`LIST`	Displays commands and responses beginning with line (1)
`LIST 10:20`	Displays lines `10` through `20` on the screen
`MEMORY`	Displays memory allocation information
`PAUSE`	Temporarily suspends execution
`RUN`	Continues execution of an Autolev input file after an error occurs
`RUN filename`	Executes the Autolev commands listed in `filename`
`SAVE filename.al`	Creates `filename.al` containing all input lines
`SAVE filename.all`	Creates `filename.all` containing all input lines and response lines
`WHAT`	Displays a list of commands, declarations, and defaults
`!`	Suspends program execution and invokes operating system
`!abcd`	Issues operating system command `abcd` from within Autolev

Default settings

`AutoEpsilon 1.0E-7`	A number differing from an integer by `1.0E-7` or less is replaced with this integer. `AutoEpsilon` is useful when tiny deviations of numbers from their neighboring integer values adversely affects Autolev's ability to simplify expressions
`AutoExpress on/off`	Causes linear/angular velocities and linear/angular accelerations to be expressed in terms of certain basis vectors
`AUTOLEVPATH`	Controls the search on disk for files used by Autolev. `AUTOLEVPATH` is an environment variable which must be set in the file `AUTOEXEC.BAT` (Windows 95/98/ME) or in the file `.cshrc` or `.login` (Unix, Linux)
`AutoRHS on/off/all`	Controls substitution for the right-hand sides of equations. See also `HELP EXPLICIT` and `HELP RHS`. Note: Normally, `AutoRHS` should not be set to `ALL` because this produces computationally inefficient formulas
`AutoZ on/off`	Controls automatic introduction of intermediate variables `Z1, Z2, ...` For large problems or to produce computationally efficient formulas, set `AutoZ ON`
`COMPLEX on/off/auto`	Controls some simplification of scalar quantities. Setting `COMPLEX ON` tells Autolev to treat scalar quantities as complex, whereas setting `COMPLEX OFF` allows Autolev to assume that scalar quantities have no imaginary parts. Setting `COMPLEX AUTO` tells Autolev that `COMPLEX` is `OFF` until Autolev encounters an imaginary number
`DEFAULTS`	Lists current settings of all defaults
`DEGREES on/off`	Determines whether the arguments of trigonometric functions (e.g., `SIN`, `COS`, `TAN`) and the value returned by inverse trigonometric (e.g., `ASIN`, `ACOS`, `ATAN`, and `ATAN2`) are regarded as representing degrees or radians. The setting of `DEGREES` does *not* affect degrees and radians conversion in Matlab, C, or Fortran code. See also `HELP UnitSystem`
`DIGITS n`	Controls the number of digits displayed for online calculations and in output files from Autolev generated Matlab, C, or Fortran code to be `n` ($1 \leq n \leq 17$)
`FACTORING on/off`	Controls the automatic factoring of expressions
`OVERWRITE on/off`	Controls whether users are queried when quantities are overwritten. Note: Quantities may also be overwritten, without query, with :=, the overwrite assignment operator
`PAUSE wait,0,1,..`	Controls delay time when a warning is issued
`SCRATCH_DIRECTORY`	Controls the location on disk of scratch files. `SCRATCH_DIRECTORY` can be set only in the file `defaults.al` and may be set to an empty string, the name of a directory, or, for fastest execution, a virtual RAM-drive
`SIMPLIFY`	Controls simplification of certain mathematical expressions involving inverse trigonometric functions
`SPACING 0,1,...`	Controls inter-line spacing
`STEPPING`	Enables execution of an input file one line at a time

Physical declarations

`Bodies A`	Declares the (massive) body `A`, the point `Ao` (mass center of A), and orthonormal vectors `A1>, A2>, A3>` fixed in `A`
`Frames B`	Declares the reference frame `B`, the point `Bo`, and orthonormal vectors `B1>, B2>, B3>` fixed in `B`
`Newtonian N`	Declares `N` to be a Newtonian (inertial) reference frame
`Particles C,D`	Declares the (massive) particles `C` and `D`
`Points E,F`	Declares the (massless) points `D` and `E`

Mathematical declarations

`Constants a,b+,c-`	Declares the constant `a`, the non-negative constant `b`, and the non-positive constant `c`
`Imaginary j`	Declares `j` as the imaginary number, $\sqrt{-1}$
`Specified f`	Declares `f` as a specified function of constants, time, and/or other variables
`Variables x'`	Declares the variable `x` and its first time-derivative `x'`

Mass and inertia declarations

`Inertia B,I1,I2,I3,I12,I23,I31`	Declares the central inertia scalars of body B
`Mass B=mB`	Declares `mB` as a non-negative constant and assigns it to the mass of the particle or body `B`

Mathematical operators and library functions

`+ - * / ^`	Mathematical operators for addition, subtraction, multiplication, division, and exponentiation
`+= -= *= /= ^=`	Cumulative mathematical operators for addition, subtraction, multiplication, division, and exponentiation
`abs(x)`	Absolute value
`cos(x) sin(x) tan(x)`	Trigonometric functions
`cosh(x) sinh(x) tanh(x)`	Hyperbolic functions
`acos(x) asin(x) atan(x)`	Inverse trigonometric functions
`atan2(y,x)`	Two argument inverse tangent function
`log(x) log10(x)`	Natural logarithm and base 10 logarithm
`exp(x)`	Exponential function
`sqr(x) sqrt(x)`	Square or square-root function
`factorial(x)`	Factorial of non-negative integer
`ceil(x) floor(x)`	Rounds floating point number up or down to nearest integer
`int(x) round(x)`	Truncates or rounds floating point number to integer
`max(x,y) min(x,y)`	Returns the maximum or minimum of two numbers
`sign(x)`	Returns +1 if $x > 0$, 0 if x=0, or -1 if $x < 0$

Mathematical commands

`AutoTaylor(0:2,w=a,x=0)`	Governs automatic Taylor series expansion of expressions about `w=a,x=0`. Useful for linearization
`D(y,x)`	Returns the partial derivative of `y` with respect to `x`
`Dt(y)`	Returns the total derivative of `y` with respect to `t` (time)
`Evaluate(y,a=x,b=2)`	Returns the expression for `y` evaluated with `a=x` and `b=2`
`Polynomial(y,x,n)`	Returns a matrix whose elements are the coefficients of $x^n, x^{n-1}, ..., x^1, x^0$ in `y`
`Roots(A)`	Returns the roots of the polynomial whose coefficients are stored in the matrix `A`
`Roots(y,x,n)`	Returns an $n \times 1$ matrix of roots of `y`, where `y` is a polynomial in `x`
`Solve(A,x1,x2)`	Solves for the variables `x1` and `x2` that appear linearly in the equations represented by the matrix `A`
`Solve(option,A,X)`	Solves the linear equations in matrix `A` for the variables in matrix `X`. `Option` is `Gauss`, `Minors` or `Implicit`
`Spline(...)`	Returns a step, bell, transition, pulse, or spline function
`Taylor(y,0:2,w=a,x=0)`	Returns the 0th, 1st, and 2nd degree terms of the Taylor series expansion of `y` about `w=a,x=0`.
`Units(n*m,lbf*ft)`	Returns the conversion factor between newton*meters and pound-force*feet

Vector and dyadic commands

`+ - *`	Vector and dyadic operators
`0> 1>>`	Zero vector. Unit dyadic
`Cross(a>,b>)`	Returns the cross-product of the vector `a>` and the vector `b>`
`Dot(a>,b>)`	Returns the dot-product of the vector `a>` and the vector `b>`
`Dot(a>,b>>)`	Returns the dot-product of the vector `a>` and the dyadic `b>`
`Dyadic`	Constructs a dyadic from a frame (or body) and scalars or a matrix of scalars
`Express(v>,B)`	Expresses `v>` in terms of the unit vectors `B1>, B2>, B3>` fixed in body (or frame) `B`
`Mag(v>)`	Returns the magnitude of `v>`
`Magsq(v>)`	Returns the square of the magnitude of `v>`, frequently denoted by $\mathbf{v}^2$. Note: `MAGSQ(v>)` $\triangleq$ `DOT(v>,v>)`
`Trace(d>>)`	Returns the trace of the dyadic `d>>`
`Unitvec(v>)`	Returns a unit vector having the same direction as `v>`
`Vector`	Constructs a vector from a frame (or body) and three scalars or a matrix of three scalars

Matrix commands

`+ - *`	Matrix operators
`Cols(A)`	Returns the number of columns in matrix `A`
`Cols(A,1,3,4)`	Returns a matrix consisting of the 1^{st}, 3^{rd}, and 4^{th} columns of the matrix `A`
`Det(A)`	Returns the determinant of the square matrix `A`
`DiagMat(n,m,x)`	Returns the $n \times m$ matrix with `x` along the "diagonal" and all other elements equal to zero
`Eig(A)`	Returns a column matrix whose elements are the eigenvalues of `A`
`Eig(A,LAMBDA,V)`	Assigns the eigenvalues of `A` to the column matrix `LAMBDA`, and the eigenvectors of A to the *rows* of `V`
`Element(A,2,3)`	Returns A[2,3], the element in the 2^{nd} row and 3^{rd} column of matrix `A`
`Inv(A)`	Returns the inverse of the square matrix `A`
`Matrix`	Constructs a matrix from a reference frame and a vector, dyadic, or triadic
`Polynomial(A,x)`	Returns a polynomial in x whose coefficients are elements of the row or column matrix `A`
`Rows(A)`	Returns the number of rows in matrix `A`
`Rows(A,1:3,5:7)`	Returns a matrix consisting of the 1^{st}-3^{rd} and 5^{th}-7^{th} rows of the matrix `A`
`Trace(A)`	Returns the trace (sum of the elements on the "diagonal") of the matrix `A`
`Transpose(A)`	Returns the transpose of matrix `A`

Mass distribution commands

`CM(P)`	Returns the position vector from point `P` to the mass center of a system of particles and/or bodies
`Inertia(P,A,B)`	Returns the sum of the inertia dyadics of `A` and `B` for point `P`
`Mass(A,B,C)`	Returns the sum of the masses of `A`, `B`, and `C`

Kinematics commands

`A1pt(A,B,Bq,Q)`	Forms `A_Q_A>` by implementing the acceleration formula for one point moving on a rigid body
`A2pts(A,B,P,Q)`	Forms `A_Q_A>` by implementing the acceleration formula for two points fixed on a rigid body
`Angvel(A,B,...)`	Forms `W_B_A>`, the angular velocity of `B` in `A` associated with Euler angles, Rodrigues parameters, or Euler parameters
`Dircos(A,B,...)`	Forms the `A_B` direction cosine matrix associated with Euler angles, Rodrigues parameters, or Euler parameters
`Kindiffs(A,B,...)`	Forms kinematical differential equations for Euler angles, Rodrigues parameters, or Euler parameters
`Partials(V_P_N>)`	Returns a matrix of partial velocities of `P` in `N`
`Simprot(A,B,1,x)`	Forms the `A_B` direction cosine matrix associated with a simple rotation of amount `x` about `A1> or B1>`
`Remainder(V_P_N>)`	Returns the velocity remainder of `P` in `N`
`V1pt(A,B,Bq,Q)`	Forms `V_Q_A>` by implementing the velocity formula for one point moving on a rigid body
`V2pts(A,B,P,Q)`	Forms `V_Q_A>` by implementing the velocity formula for two points fixed on a rigid body

Kinetics commands

`Gravity(G*n1>)`	Adds a local gravitational force to each particle and body
`Force(P/Q,v>)`	Adds -v> to the force on `P`, adds +v> to the force on `Q`
`Fr()`	Returns a matrix of generalized active forces
`Torque(A/B,v>)`	Adds -v> to the torque on `A`, adds +v> to the torque on `B`

Dynamics commands

`Constrain(...)`	Solves for dependent and auxiliary generalized speeds and their time derivatives
`FrStar()`	Returns a matrix of generalized inertia forces
`Gyrostat(option)`	Facilitates formulation of expressions for `FRSTAR`, `KE`, and `ANGMOM` for gyrostats
`Kane()`	Brings Kane's dynamical equations into a form suitable for numerical integration; determines force/torque measure numbers associated with auxiliary generalized speeds
`KE()`	Forms kinetic energy
`Momentum(option)`	Forms `LINEAR`, `ANGULAR`, and `GENERALIZED` momentum
`NiCheck()`	Forms a function that should remain constant throughout numerical integration of equations of motion
`TStar(B,N)`	Forms inertia torque of body `B` in frame `N`

Code commands

`Animate`	Informs Autolev that certain frames and/or bodies are to be animated by means of the program Animake
`CODE Option()`	Generates a Matlab, C, or Fortran program. `Option` is `Dynamics`, `ODE`, `Nonlinear`, or `Algebraic`
`Digits`	Governs the number of digits displayed in output files of Matlab, C, or Fortran programs
`Input X=2,Y=3`	Assigns the value 2 to `X` and 3 to `Y` for use in a Matlab, C, or Fortran input file
`Input X=2 deg, Y=3 m`	Assigns the value 2 degrees to `X` and 3 meters to `Y` for use in a Matlab, C, or Fortran input file
`Input(X)`	Returns the input value assigned to `X`
`Input(X,UnitSystem)`	Returns the input value assigned to `X` multiplied by `conversionFactor`, where `conversionFactor` is the conversion factor from the units assigned to `X` to the units named in the UnitSystem declaration
`Output T,X+Y`	Specifies `T` and `X+Y` as output quantities for Matlab, C, or Fortran programs
`Output T sec, X+Y m`	Specifies `T` in seconds and `X+Y` in meters as output quantities for Matlab, C, or Fortran programs
`UnitSystem`	Declares a unit system for automatic conversion of units declared in Input or Output commands

Simplifications commands

`Arrange(y,n,x) *`	Arranges in groups those terms in `y` which are of degree `n` in `x` (`n` = 0, 1, or 2)
`AutoZ(y,x)`	If AutoZ is ON, returns `ZEE(y,x)`, else returns `y`
`Coef(y,x)`	Returns the coefficient of `x` in the expression for `y` (`y` must be linear in `x`)
`Epsilon(y,0.1)`	Returns an expression found by setting numbers in `y` which are within `0.1` of an integer equal to that integer
`Exclude(y,x)`	Returns the terms in `y` that do not contain `x` (`y` does *not* have to be linear in `x`)
`Expand(y,n:m) *`	Removes parentheses, e.g., `(a+b)*c` becomes `a*c + b*c`
`Explicit(y,x)`	Makes `y` an explicit function of `x`; e.g., if `a=x+t` and `y=a+t`, `explicit(y,t)` produces `x+2*t`
`Factor(y,x) *`	Returns `y` factored on `x`
`Include(y,x)`	Returns the terms in `y` that contain `x` (`y` must be linear in `x`)
`Replace(y,sin(x)=3)`	Replaces `sin(x)` with 3 in the expression for `y`
`RHS(y)`	Returns the right-hand side of `y`
`Zee(y) *`	Produces a computationally efficient expression for `y` by introducing intermediate variables `Z1, Z2, ....`
`Zee(y,x) *`	Produces a compact expression for `y` by introducing variables `Z1, Z2, ...` which do not explicitly contain `x`. (`y` must be linear in `x`)

* denotes dual function, i.e., a command that can be entered either on the right-hand side of an equation or immediately following an Autolev prompt.