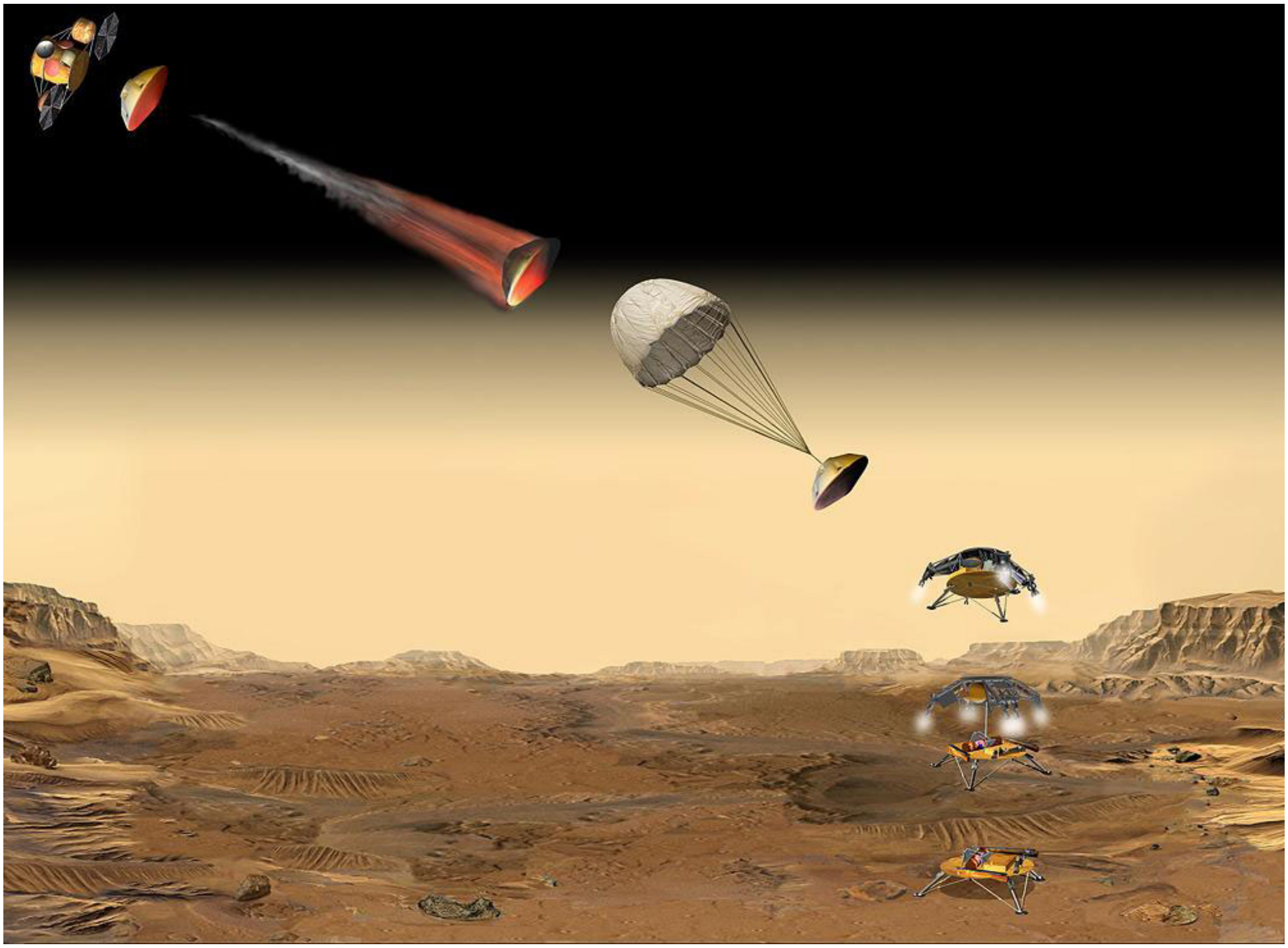
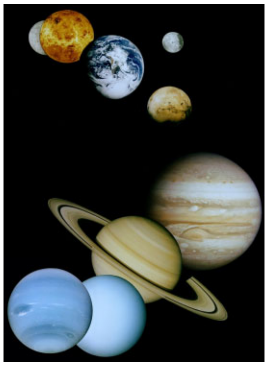
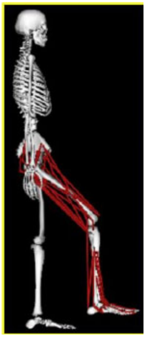
MotionGenesis™ Kane Tutorial

Software, textbooks, training, and consulting
Force, motion, and code-generation tools
www.MotionGenesis.com







English tutorial: March 26, 2015
Basic math, motion, & simulation

MotionGenesis Kane

End-user license agreement (EULA)

Last updated March 17, 2015. Subject to change without notice

This program is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability or fitness for a particular purpose. In no event shall the authors, contributors, or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the software or the use or other dealings in the software.

This program is protected by copyright law and international treaties.
Copyright includes, but is not limited to:
names of commands, methods, functions, and declarations;
responses, output, and code (C, FORTRAN, etc.) generated by the program;
documentation, help, examples, messages, warnings; and
website content associated with www.MotionGenesis.com.

Unauthorized use, reproduction, distribution, modification, translation, sale, derivative work, or reverse-engineering of this program, or any portion of it, may result in severe civil and criminal penalties.

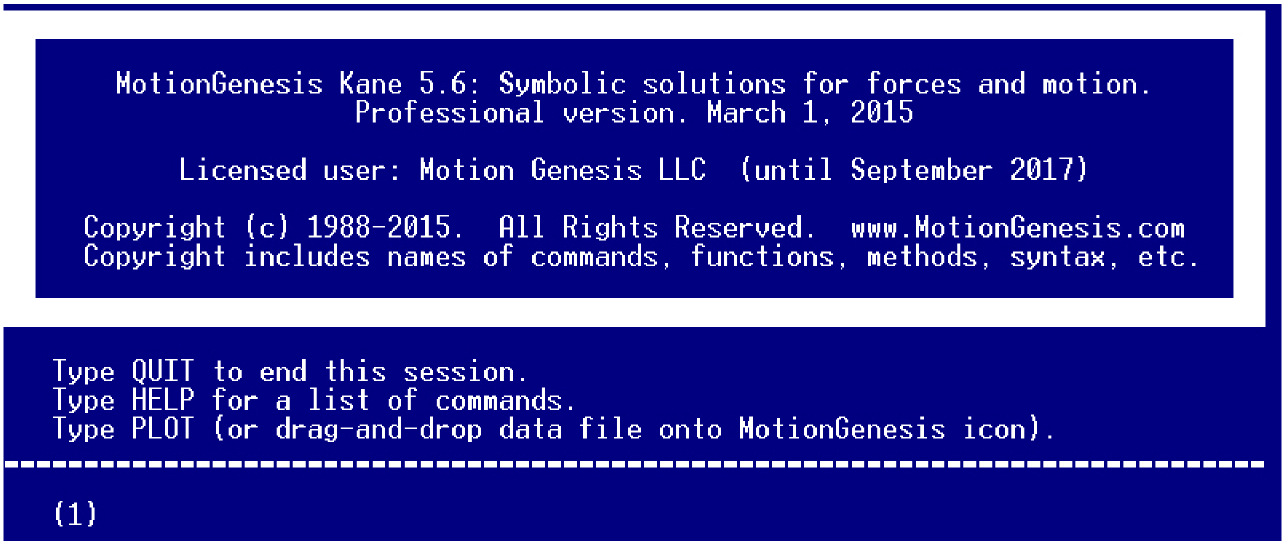
Disputes: This agreement shall be governed by the laws of the state of California, regardless of its place of execution or performance. I consent to exclusive jurisdiction and venue of the federal and state courts holding jurisdiction in California for any action brought in connection with this agreement.

Agreement in Parts: If one or more provisions in this agreement are deemed void by law, the remaining provisions will be in full force and effect.

Do you understand and agree to be legally bound by these terms and all U.S. and international copyright and patent law applicable to this program (Y/N)?

Contents

1 Get started & basic input/output



```
    MotionGenesis Kane 5.6: Symbolic solutions for forces and motion.
                Professional version. March 1, 2015

        Licensed user: Motion Genesis LLC  (until September 2017)

  Copyright (c) 1988-2015.  All Rights Reserved.  www.MotionGenesis.com
  Copyright includes names of commands, functions, methods, syntax, etc.

  Type QUIT to end this session.
  Type HELP for a list of commands.
  Type PLOT (or drag-and-drop data file onto MotionGenesis icon).
---------------------------------------------------------------------------

  (1)
```

Follow the download and install directions at www.MotionGenesis.com ⇒ Get Started.
Note: There are **significantly more** examples at www.MotionGenesis.com ⇒ Get Started.

On line (1), type

```
sum = 2 + 2
```

Press Enter and observe the response.
Note: Output lines are preceding with an arrow . → . Some commands do not produce output.

Next, enter the following **symbolic** expression and observe the **automatic simplification** to $1 + \sin(t)^2$.
Note: Since the program is case-insenstive, you may use upper-case or lower-case letters (or a mix).

```
someName = 2*sin(t)^2 + cos(t)^2
```

Saving input

To **save** input to the text file FirstDemo.al, enter

```
Save  FirstDemo.al
```

Exit the program by typing

```
Quit
```

Running files

Modify the file FirstDemo.al with a text-editor (e.g,, NotePad, SimpleText, TextEdit, Emacs).
Put the following comment line at the top of the file and ensure the editor saves the updated file.

```
% File: FirstDemo.al
```

To **run** the input file FirstDemo.al, invoke the program and type[1]

```
Run   FirstDemo.al
```

Saving input and output

To **save** input commands together with output responses in the file FirstDemo.all, enter

```
Save FirstDemo.all
```

Printing files

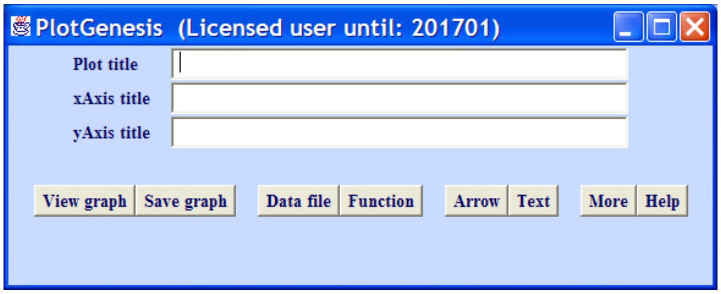
To **print** an input or output file (e.g., FirstDemo.all), open the file in a text editor (e.g,, NotePad, SimpleText, TextEdit, Emacs) or word-processing program with Courier Font (e.g,, Microsoft Word) and print it from within that program.

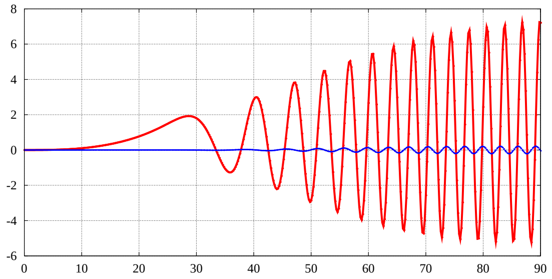
Online help

For general **help** and/or a list of commands, type HELP.
For help with a command, e.g., SOLVE, type Help SOLVE.

1.1 Running the PlotGenesis plotting program

To invoke PlotGenesis, double click on the PlotGenesis icon and follow the on-screen instructions. The Windows version of PlotGenesis supports drag and drop (i.e., drag a data file on top of the PlotGenesis icon to start PlotGenesis and load the data file). Alternately, invoke **M**otion**G**enesis and type **plot**.





---

[1] Instead of interactively entering commands, it is generally **easier** to use a text editor to create a text file (e.g., FirstDemo.al) and then execute the commands in that file. Additionally, lines read from a text file can be broken into multiple lines by using an ampersand `(&)` as the last non-blank character of each but the last input line – which informs **M**otion**G**enesis that the command continues on the next line. This is advantageous for entering lines longer than 512 characters.

2 Mathematical declarations

2.1 Scalars

Names of scalar quantities must start with a letter and may be followed by alphanumeric characters or underscores (_). For example, `x`, `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 (') at the end of a name. Each prime represents one differentiation (e.g., `x''` is the 2^{nd} 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 named in a **Constant**, **Specified**,[2] **Variable**, **SetMass**, **SetInertia**, or **SetImaginaryNumber** declaration, or is defined by appearing on the left-hand side of an equals sign in an assignment. For example,

```
Constant   a                     % Declares a as a constant
Constant   b = 3 meters          % Declares b as a constant with an input value of 3 meters
Constant   c+                    % Declares c to be a non-negative constant
Constant   d-                    % Declares d to be a non-positive constant
Specified  phi                   % Declares phi as a function of time, constants, and variables
Variable   q, s                  % Declares the variables q and s
Variable   x''                   % Declares the variables x,  x',  x''
Variable   u{3}'                 % Declares the variables u1, u2, u3, and u1', u2', u3'
SetImaginaryNumber( j )          % Declares j to be the imaginary number, i.e., j = sqrt(-1)
Tina = 2*pi                      % Creates the scalar Tina and assigns 2*pi to Tina
```

2.2 Vectors >, Dyadics >>, and Polyadics >>>

The "greater than" character > is used to distinguish scalars, vectors, dyadics, and polyadics,

- One "greater than" > is appended to the end of a vector's name, e.g., $a >$
- Two are appended to the end of a dyadic's name, e.g., $b >>$
- Three are appended to the end of a triadic or higher-order polyadic, e.g., $c >>>$

Names of vectors, dyadics, and polyadics must start with a letter. This first letter may be followed by alphanumeric characters or underscores (_). A total of 64 characters (including > symbols) may be used to form the name of a vector, dyadic, triadic, etc.

---

[2] A ***specified*** quantity varies in a **known way**, i.e., is prescribed as a function of constants, time, and other variables. Frequently, the action or motion of a **motor** is specified, e.g., a linear actuator whose force, length, or elongation is specified or a rotational motor whose torque, angle, or angular velocity is specified.

2.3 Matrices

Matrices start with a left bracket (`[`) and end with a right bracket (`]`). Elements in a row are separated 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 (_). Matrix names are less than 64 characters long.

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 `]` e.g., Ed[2,5]. Elements of a one-dimensional matrix can be referenced by in 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 and **1>>** is the unit dyadic.
- Unless otherwise declared, `imaginary = ` $\sqrt{-1}$.
- For those needing backward compatibility with Autolev:
  - **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.
  - **Variables U{n}** or **Variables U{n}'** introduce motion variables.
  - **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.
  - **ZEE_NOT** is reserved for a matrix of scalar quantities that are excluded from `Z1,Z2,...`.

3 Physical declarations

3.1 RigidBody, RigidFrame, NewtonianFrame, Point, Particle

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

```
RigidBody  A        % Declares the (massive) rigid body A.
                    % Introduces orthonormal vectors Ax>, Ay>, Az> fixed in A.
                    % Declares a point Ao  fixed on A.
                    % Declares a point Acm fixed on A and at A's center of mass.

RigidFrame B        % Declares the (massless) reference frame B.
                    % Introduces orthonormal vectors Bx>, By>, Bz> fixed in B.
                    % Declares a point Bo that is fixed on B

NewtonianFrame N    % Declares N as a Newtonian (inertial) reference frame.
                    % Introduces orthonormal vectors Nx>, Ny>, Nz> fixed in N.
                    % Declares a point No that is fixed on N

Particle   C, D     % Declares the (massive) particles C and D

Point      E, F     % Declares the (massless) points E and F
Points     G( A )   % Declares the (massless) point G that is fixed on A.
```

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 reference frame B in reference frame F
alf_E_A>        % Angular acceleration of reference frame E in reference frame A
Force_P>        % Force acting on point P
Force_P_Q>      % Force acting on point P from point Q
Torque_B>       % Torque of a couple acting on RigidFrame or RigidBody B
Torque_B_A>     % Torque of a couple acting on B from RigidFrame or RigidBody A
I_B_P>>         % Inertia dyadic of RigidBody B about point P
A_B             % Direction cosine matrix relating Ax>, Ay>, Az> to Bx>, By>, Bz>
```

Note: Element `A_B[i,j]` of `A_B` is defined as `Dot(Ai>,Bj>)` `(i,j = x, y, z)`. As a consequence,

$$\texttt{A\_B} = \begin{bmatrix} \text{Ax}> \cdot \text{Bx}> & \text{Ax}> \cdot \text{By}> & \text{Ax}> \cdot \text{Bz}> \\ \text{Ay}> \cdot \text{Bx}> & \text{Ay}> \cdot \text{By}> & \text{Ay}> \cdot \text{Bz}> \\ \text{Az}> \cdot \text{Bx}> & \text{Az}> \cdot \text{By}> & \text{Az}> \cdot \text{Bz}> \end{bmatrix}$$

3.3 Mass Declarations

The masses of bodies and particles are declared by the `SetMass` command. For example:

```
   (1) RigidBody  A, B
   (2) Particle   C
   (3) A.SetMass( 12.5 );   B.Setmass( mB );   C.SetMass( mC = 10 kg )
   (4) massA = A.GetMass()
-> (5) massA = 12.5
   (6) massAB = A.GetMass() + B.GetMass()
-> (7) massAB = 12.5 + mB
   (8) TotalMass = System.GetMass()
-> (9) TotalMass = 22.5 + mB + mC
```

3.4 Inertia Declarations

As previously mentioned, `I_B_P>>` denotes the inertia dyadic of RigidBody B about 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*Ax>*Ax> + 200*Ay>*Ay> + 250*Az>*Az>
  ```

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

  ```
  A.SetInertia( Acm, Ixx, Iyy, Izz, Ixy, Iyz, Izx )
  ```

  is equivalent to typing

  ```
  Constant   Ixx+, Iyy+, Izz+, Ixy, Iyz, Izx
  I_A_Acm>> = Ixx*Ax>*Ax> + Ixy*Ax>*Ay> + Izx*Ax>*Az>
            + Ixy*Ay>*Ax> + Iyy*Ay>*Ay> + Ixy*Ay>*Az>
            + Izx*Az>*Ax> + Iyz*Az>*Ay> + Izz*Az>*Az>
  ```

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

- If the inertia dyadic of a RigidBody B about a point P has already been entered, **M**otion**G**enesis can calculate the inertia dyadic of B about a different point, e.g., B_o if B's mass has been declared and appropriate position vectors have been entered. For example,

  ```
     (1) RigidBody  B
     (2) Point      P( B )
     (3) B.SetMass( mB )
     (4) B.SetInertia( P, I, I, J )
     (5) Constant   L
     (6) p_Bo_P> = L*Bx>
  -> (7) p_BO_P> = L*Bx>
     (8) I_B_Bo>> = B.GetInertia( Bo )
  -> (9) I_B_Bo>> = I*Bx>*Bx> + (I-mB*L^2)*By>*By> + (J-mB*L^2)*Bz>*Bz>
  ```

3.5 Forces and torques

`Force_P>`	is the resultant force acting on a point or particle `P`.
`Force_P_Q>`	is the resultant force acting on point or particle `P` from point or particle Q.
`Torque_B>`	is the resultant torque acting on a RigidFrame or RigidBody B.
`Torque_B_A>`	is the resultant torque acting on reference frame B from RigidFrame or RigidBody A.

There are multiple ways to assign values to a `Force` or `Torque` vector:

- `P.AddForce( 5*Az> )      % Adds 5*Az> to the previous value of Force_P>`
- `Force_P> -= 6*Az>        % Subtracts 6*Az> from the previous value of Force_P>`
- `Q.AddForce(P, Vec> )     % Adds Vec> to the force on Q from P`
- `Force_Q> := 7*Ax>        % Explicit assignment overwrites previous value for Force_Q>`
- `A.AddTorque( Vec> )      % Adds Vec> to the previous value of Torque_A>`
- `Torque_A> -= Vec>        % Subtracts VEC> from the previous value of Torque_A>`
- `B.AddTorque( A, Vec> )   % Adds Vec> to the torque on B from A`
- `Torque_B> := 7*Bz>       % Explicit assignment overwrites previous value for Torque_B>`

4 Procedures for solving problems

4.1 Solving ordinary differential equations (ODEs)

To solve ODEs with **M**otion**G**enesis or write MATLAB®, C, or Fortran code to solve ODEs:

- Declare all variables and their derivatives.
- Enter the equations governing the highest derivative of each variable.
- Use `Input` statements to specify integration parameters and initial values of variables.
- Use `Output` statements to specify the quantities to be output.
- Type `ODE() Filename.ext` (`ext` is `m`, `c`, `for`, `f`, or missing).
  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`
  When `ext` is missing, immediately solves the ODEs and outputs the results in `Filename.1`

```
Variable  x', y'
x' = -x - 2*y
y' = -y - t*x^2
Input  x = 2 meters,  y = 3 meters,   tFinal = 5 second,  tStep = 0.1 second
Output  t seconds,  x m,  y m,  x' m/s,  y' m/s,  sin(x)^2 noUnits
ODE() test
```

Note: Changing "test" to "test.m" creates ready-to-compile MATLAB® code in the file `test.m`. Changing "test" to "test.c" creates ready-to-compile C code in the file `test.c` and an input file `test.in` that is read by the executable program (e.g., `test.exe`). These programs numerically solve the ODEs for `x` and `y` from `t=0` to `t=5` with integration steps of 0.1 s.

4.2 Solving nonlinear algebraic equations

- Declare unknowns (e.g., as variables).
- Create a matrix whose elements represent the nonlinear equations.
- Use `Input` statements to assign values to constants (and perhaps the convergence parameter `absError`).
- Use the `Solve` command with guessed solutions for the unknowns.

```
% Example: Solve the following nonlinear equations for x and y.
Constant   a = 1.0 m, r = 1.0 m
Variable   x, y
Eqn[1] = x^2 + y^2 - r          % Equation of circle:    x^2 + y^2 - r = 0
Eqn[2] = y - a*sin(x)           % Equation of sinusoid:  y - a*sin(x) = 0
Input  x = 0.5 m,  y = 0.5 m
Solve( Eqn,  x= 0.5 m,  y = 0.5 m )
```

Alternately: Type `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. Edit `katy.m` to try a different initial guess for `x` or `y` or to use a different value for `a` or `b`. Change `katy.m` to `katy.c` or `katy.f` for C or Fortran code.

4.3 Solving linear algebraic equations

- Declare unknowns (e.g., as variables).
- Create a matrix whose elements represent the linear equations.
- Use the `Solve` command.

```
Variable  x,  y
Constant  a{1:2, 1:2},  b{1:2}
Eqn[1] = a11*x + a12*y - b1              %   a11*x + a12*y = b1
Eqn[2] = a21*x + a22*y - b2              %   a21*x + a22*y = b2
Solve( Eqn,  x, y )
```

Alternately: Type `Code Algebraic(Eqn, x, y) becky.for`. This creates the ready-to-compile Fortran code `becky.for` and an input file `becky.in` 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` in `becky.in`. Note: Edit `becky.in` with a text editor to try different values of `a11`, `a12`, `a21`, `a22`. Change `becky.for` to `becky.c` or `becky.m` for C or MATLAB® code.

4.4 Forming equations of motion (examples in Chapter 9).

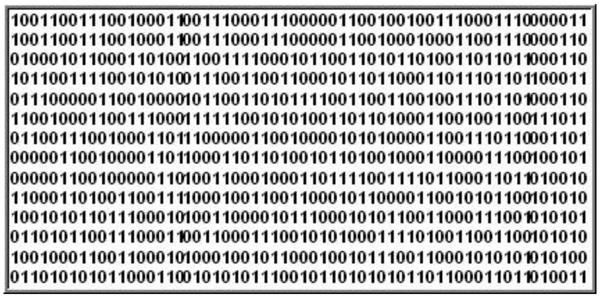
MotionGenesis can help form equations of motion with many methods. It is especially good for dynamics with Newton/Euler or Kane's method (textbooks can be purchased via www.MotionGenesis.com ⇒ Textbooks).

- Declare a **NewtonianFrame**.
- Use the **RigidBody**, **RigidFrame**, **Point**, and **Particle** declarations.
- Declare generalized speeds, e.g., `SetGeneralizedSpeed( wx, wy, wz, x', y', z' )`
- Declare generalized coordinates and their time-derivatives with a declaration of the form `Variable   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 rotation matrices, $\vec{\boldsymbol{\omega}}$, and $\vec{\boldsymbol{\alpha}}$, e.g., with the **Rotate** command.
  As needed, form other angular velocities/accelerations with **SetAngularVelocityAcceleration**
- Form position vectors, $\vec{\mathbf{v}}$, and $\vec{\mathbf{a}}$, e.g., with the **Translate** and/or **SetPosition** commands.
  As needed, form additional velocities/accelerations with **SetVelocityAcceleration**
- If necessary, impose motion constraints with a variation of the **Solve** or **SolveDt** commands
- Add forces and torques with **AddForce** or **AddTorque** commands.
- Form particle Q's translational dynamics with `Zero = Q.GetDynamics()`
  Form rigid body B's rotational dynamics about B_{cm} with `Zero = B.GetDynamics( Bcm )`
  Form equations of motion with Kane's method with `Zero = System.GetDynamicsKane()`

  Note: For large problems, or problems which require real-time solution, type **SetAutoZ(ON)**.

Chapter 5

Computer techniques



Do the basic two minute exercises at www.MotionGenesis.com ⇒ Get Started

5.1 Declaration of scalars (constant, variable, specified) in MotionGenesis

Declaration	Description
`Constant a`	Declares `a` as a constant
`Constant b, c, Fred`	Declares `b`, `c`, and `Fred` as constants.
`Variable x`	Declares `x` as a variable (**unknown**)
`Variable y'`	Declares `y` and `y'` (i.e., $\dot{y}$) as variables (**unknowns**)
`Variable z''`	Declares `z`, `z'`, and `z''` as variables (**unknowns**)
`Variable z'' = 2*pi*t + z`	Declares `z`, `z'`, and `z''` as variables and assigns $\ddot{z} = 2\,\pi\,z + z$
`Specified s`	Declares `s` as specified (**known** or **prescribed**)
`Specified motorSpeed'`	Declares `motorSpeed` and `motorSpeed'` as specified (**known** or **prescribed**)
`Specified h'''`	Declares `h`, `h'`, `h''`, and `h'''` as specified (**known** or **prescribed**)
`Specified h' = sin(2*pi*t*h)`	Declares `h` and `h'` (i.e., $\dot{h}$) as specified and assigns $h' = \sin(2\,\pi\,t\,h)$
`SetImaginaryNumber( i )`	Declares `i` as the imaginary number, i.e., `i` $= \sqrt{-1}$

By default, MotionGenesis defines `t` as the independent variable, `Pi` as π, and `imaginary` as $\sqrt{-1}$.

5.2 Converting units with MotionGenesis

Note: MotionGenesis output results are marked with `->`

```
   (1) %--------------------------------------------------------------------
   (2) %Example 1: ConvertUnits
   (3) %--------------------------------------------------------------------
   (4) InchesToCentimeter = ConvertUnits( inch, cm )
-> (5) InchesToCentimeter = 2.54

   (6) OunceMassToMilligram = ConvertUnits( ozm, mg )
-> (7) OunceMassToMilligram = 28349.52

   (8) PoundForceToNewton = ConvertUnits( lbf, Newton )
-> (9) PoundForceToNewton = 4.448222

   (10) Convert60MPHToMetersPerSecond = 60 * ConvertUnits( MPH, m/sec )
-> (11) Convert60MPHToMetersPerSecond = 26.8224

   (12) %--------------------------------------------------------------------
   (13) %Example 2: ConvertUnits
   (14) %--------------------------------------------------------------------
   (15) Convert60MPHToMetersPerSecond := ConvertUnits( (30+30) MPH, m/sec )
-> (16) Convert60MPHToMetersPerSecond = 26.8224

   (17) ConvertTMinutesToSeconds = ConvertUnits( t minutes, seconds )
-> (18) ConvertTMinutesToSeconds = 60*t
```

5.3 Symbolic differentiation with MotionGenesis

MotionGenesis symbolically calculates ***partial derivatives*** and ***ordinary time-derivatives***.
Note: **MotionGenesis** output results are marked with `->`

```
   (1) Variable  x, y
   (2) z = y*cos(x) + 2*x^2*sin(y)
-> (3) z = y*cos(x) + 2*x^2*sin(y)

   (4) partialDerivativeOfZwithRespectToY = D( z, y )
-> (5) partialDerivativeOfZwithRespectToY = cos(x) + 2*x^2*cos(y)

   (6) partialDerivativeOfZWithRespectToX = D( z, x )
-> (7) partialDerivativeOfZWithRespectToX = 4*x*sin(y) - y*sin(x)

   (8) Variable  s'   % Declares s as a variable and s' as it's ordinary time-derivative
   (9) funct = log(s) + s*exp(s)
-> (10) funct = log(s) + s*exp(s)

   (11) ordinaryTimeDerivativeOfFunct = Dt( funct )
-> (12) ordinaryTimeDerivativeOfFunct = (1/s+exp(s)+s*exp(s))*s'
```

5.4 Optional**: Numerical calculation of integrals with MotionGenesis

The **MotionGenesis** `Integrate` command numerically calculates single, double, triple integrals.
The website www.Mathematica.com is a valuable resource for ***symbolically*** calculating integrals.

```
   (1) Variable  x, y
   (2) integralA = Integrate( 2*x, x=0:4 )
-> (3) integralA = 16

   (4) integralB = Integrate( 2*x*abs(cos(x))^3.4*sqrt(exp(x)), x=0:3 )
-> (5) integralB = 10.79699

   (6) integralC = Integrate( Integrate( x*y, x=0:y ), y=0:2 )
-> (7) integralC = 2

   (8) integralD = Integrate( y^2 * Integrate( sin(x*y), x=0:y ), y=0:2 )
-> (9) integralD = 2.378401
```

5.5 Solutions of *linear* algebraic equations

It is relatively easy to solve a single, uncoupled, ***linear algebraic equation***, e.g., solving for x in

$$3\,x \;+\; 9\,\sin(t) \;-\; 12 \;=\; 0 \qquad \text{or} \qquad \begin{bmatrix} 3 \end{bmatrix} \begin{bmatrix} x \end{bmatrix} \;=\; \begin{bmatrix} \text{-}9\,\sin(t) \;+\; 12 \end{bmatrix}$$

Solving two ***coupled linear algebraic equations*** for y and z is a little more difficult, e.g.,

$$\begin{aligned} 3\,y \;+\; 2\,z \;+\; 9\,\sin(t) \;-\; 12 \;&=\; 0 \\ 2\,y \;+\; 4\,z \;+\; 5\,\cos(t) \;-\; 11 \;&=\; 0 \end{aligned} \qquad \text{or} \qquad \begin{bmatrix} 3 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} \;=\; \begin{bmatrix} \text{-}9\,\sin(t) \;+\; 12 \\ \text{-}5\,\cos(t) \;+\; 11 \end{bmatrix}$$

Solving four ***coupled linear algebraic equations*** for x_1, x_2, x_3, x_4 is more difficult, e.g.,

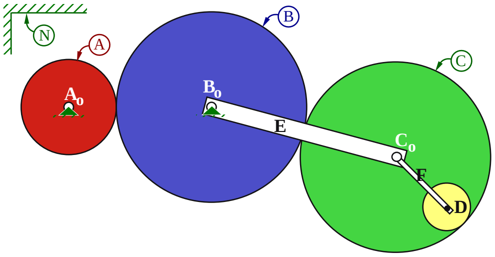
$$\begin{aligned} 3\,x_1 + 2\,x_2 + 2\,x_3 + 3\,x_4 &= 9\,\sin(t) \\ 2\,x_1 + 4\,x_2 + 2\,x_3 + 3\,x_4 &= 5\,\cos(t) \\ 4\,x_1 + 5\,x_2 + 6\,x_3 + 7\,x_4 &= 11 \\ 9\,x_1 + 8\,x_2 + 7\,x_3 + 6\,x_4 &= 15 \end{aligned} \qquad \text{or} \qquad \begin{bmatrix} 3 & 2 & 2 & 3 \\ 2 & 4 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 9 & 8 & 7 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 9\,\sin(t) \\ 5\,\cos(t) \\ 11 \\ 15 \end{bmatrix}$$

Solutions of previous *linear* algebraic equations with MotionGenesis (symbolic)

```
Variable  x
Equation = 3*x + 9*sin(t) - 12
Solve( Equation, x )
%------------------------------------------
Variable  y, z
Zero[1] = 3*y + 2*z + 9*sin(t) - 12
Zero[2] = 2*y + 4*z + 5*cos(t) - 11
Solve( Zero, y, z )
%------------------------------------------
Variable  x{1:4}
Eqn[1] = 3*x1 + 2*x2 + 2*x3 + 3*x4 - 9*sin(t)
Eqn[2] = 2*x1 + 4*x2 + 2*x3 + 3*x4 - 5*cos(t)
Eqn[3] = 4*x1 + 5*x2 + 6*x3 + 7*x4 - 11
Eqn[4] = 9*x1 + 8*x2 + 7*x3 + 6*x4 - 15
Solve( Eqn, x1, x2, x3, x4 )
%------------------------------------------
Save SolveLinearEquations.all
Quit
```

Example: Symbolic solution of linear equations for gear constraints with MotionGenesis

The following **MotionGenesis** file solves a set of seven coupled linear algebraic equations. This analysis regards ${}^N\omega^E$ and ${}^N\omega^F$ as free variables (associated with the two degrees of freedom) and ${}^N\omega^A$ as ***specified***.



```
% File:  GearTrainABCD.txt
%-----------------------------------------------------------------
Constant   rA, rB, rC, rD     % Gear radii
Specified  wAN                % Known function of time
Variable   wBN, wCN, wDN      % Unknown gear angular rates
Variable   wEN, wFN           % Unknown rigid rod angular rates
Variable   wBE, wCE           % Useful for analysis
Variable   wCF, wDF           % Useful for analysis
%-----------------------------------------------------------------
%       Motion constraints
Zero[1] = wAN*rA  + wBN*rB    % Rolling contact between A and B
Zero[2] = wBE*rB  + wCE*rC    % Rolling contact between B and C
Zero[3] = wCF*rC  - wDF*rD    % Rolling contact between C and D
Zero[4] = wBN - (wEN + wBE)   % Angular velocity addition theorem
Zero[5] = wCN - (wEN + wCE)   % Angular velocity addition theorem
Zero[6] = wCN - (wFN + wCF)   % Angular velocity addition theorem
Zero[7] = wDN - (wFN + wDF)   % Angular velocity addition theorem
Solve( Zero, wBN, wCN, wDN, wBE, wCE, wCF, wDF )
%-----------------------------------------------------------------
Save GearTrainABCD.all
Quit
```

5.6 Solution of *quadratic* and *polynomial* equations (roots)

Polynomial equations are a special class of nonlinear algebraic equations. Although there are closed-form solutions for linear, quadratic, cubic, and quartic polynomial equations, there are no general closed-form solutions for 5^{th} and higher-order polynomials.

Symbolic roots of quadratic equation $a\,x^2 + b\,x + c = 0$ with MotionGenesis

```
   (1) %--------------------------------------------------------------------
   (2) % Example 1: GetQuadraticRoots  (roots of quadratic equation)
   (3) %--------------------------------------------------------------------
   (4) Constant a, b, c
   (5) Variable x
   (6) rootsA = GetQuadraticRoots( a*x^2 + b*x + c, x )
-> (7) rootsA[1] = -0.5*(b-sqrt(b^2-4*a*c))/a
-> (8) rootsA[2] = -0.5*(b+sqrt(b^2-4*a*c))/a

   (9) positiveRootA = GetQuadraticPositiveRoot( a*x^2 + b*x + c, x )
-> (10) positiveRootA = -0.5*(b-sqrt(b^2-4*a*c))/a

   (11) negativeRootA = GetQuadraticNegativeRoot( a*x^2 + b*x + c, x )
-> (12) negativeRootA = -0.5*(b+sqrt(b^2-4*a*c))/a

   (13) %--------------------------------------------------------------------
   (14) % Example 2: GetQuadraticRoots  (roots of quadratic equation)
   (15) %--------------------------------------------------------------------
   (16) rootsB = GetQuadraticRoots( [a; b; c] )
-> (17) rootsB[1] = -0.5*(b-sqrt(b^2-4*a*c))/a
-> (18) rootsB[2] = -0.5*(b+sqrt(b^2-4*a*c))/a
```

Roots of 5^{th}-order polynomial $p^5 + 2\,p^4 + 3\,p^3 + 5\,p^2 + 9\,p + 17 = 0$ with MotionGenesis

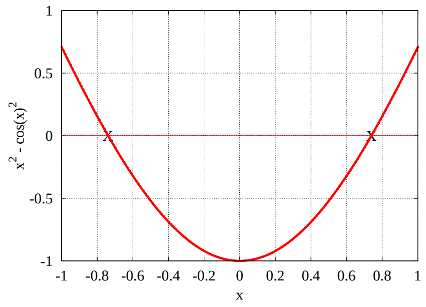
```
   (1) %--------------------------------------------------------------------
   (2) % Example 1: GetPolynomialRoots  (roots of 5th-order polynomial)
   (3) %--------------------------------------------------------------------
   (4) SetImaginaryNumber( i )
   (5) Variable  p
   (6) rootsA = GetPolynomialRoots( p^5 + 2*p^4 + 3*p^3 + 5*p^2 + 9*p + 17,  p, 5 )
-> (7) rootsA = [-1.857621; -0.9475112 - 1.507048*i; -0.9475112 + 1.507048*i;
       0.8763218 - 1.455989*i; 0.8763218 + 1.455989*i]

   (8) %--------------------------------------------------------------------
   (9) % Example 2: GetPolynomialRoots  (roots of 5th-order polynomial)
   (10) %--------------------------------------------------------------------
   (11) rootsB = GetPolynomialRoots( [1, 2, 3, 5, 9, 17] )
-> (12) rootsB = [-1.857621, -0.9475112 - 1.507048*i, -0.9475112 + 1.507048*i,
        0.8763218 - 1.455989*i, 0.8763218 + 1.455989*i]
```

5.7 Solutions of *nonlinear* algebraic equations

The graph of the function $x^2 - \cos^2(x)$ is **nonlinear** (i.e., it is **not a line**) and has two solutions, namely $x \approx 0.74$ and $x \approx -0.74$.

Motion**G**enesis solves ***nonlinear algebraic equations*** using an algorithm that requires a **guess** and iterates towards a solution (frequently the solution closest to the guess). For example, the following **M**otion**G**enesis commands produce the solution $x = 0.74$.

$$x^2 \;-\; \cos^2(x) \;=\; 0$$



```
Variable   x
Solve( x^2 - cos(x)^2,  x=2 )      % x=2 is a guess to a solution
Quit
```

5.7.1 Solutions of *coupled nonlinear* algebraic equations with MotionGenesis

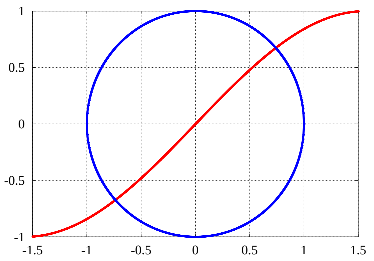
The coupled set of algebraic equations to the right is **nonlinear**[a] in x and y (a circle and sine curve are **not lines**). These two curves intersect at two locations (there are two solutions to these equations), namely $x \approx 0.74$, $y \approx 0.67$ and $x \approx -0.7391$, $y \approx -0.6736$.

In general, it is difficult to determine the ***number of solutions*** to nonlinear algebraic equations, and the solution process usually requires a numerical algorithm that starts with a **guess** and iterates towards a solution.

---

[a] Coupled nonlinear algebraic equations frequently arise in determining equilibrium configurations of static systems. Although nonlinear equations with **one** or **two** unknowns can be solved by **trial and error** or **graphing**, generally, ***Newton-Rhapson*** techniques are used to solve sets of nonlinear equations.

$$\begin{aligned} x^2 \;+\; y^2 \;&=\; 1 \\ y \;&=\; \sin(x) \end{aligned}$$



For example, the following **M**otion**G**enesis commands produce the solution $x = -0.7391$, $y = -0.6736$.

```
Variable   x, y
Zero[1] = x^2 + y^2 - 1            % x^2 + y^2 = 1   (unit circle)
Zero[2] = y - sin(x)               % y = sin(x)      (sine wave)
Solve( Zero, x = 1.5, y = 0 )      % x=1.5, y=0 is a guess to a solution
Quit
```

5.7.2 Starting guesses and solutions of coupled *nonlinear* algebraic equations

In the range $0 \le x \le 24$, the following **nonlinear** algebraic equation has **5** solutions.

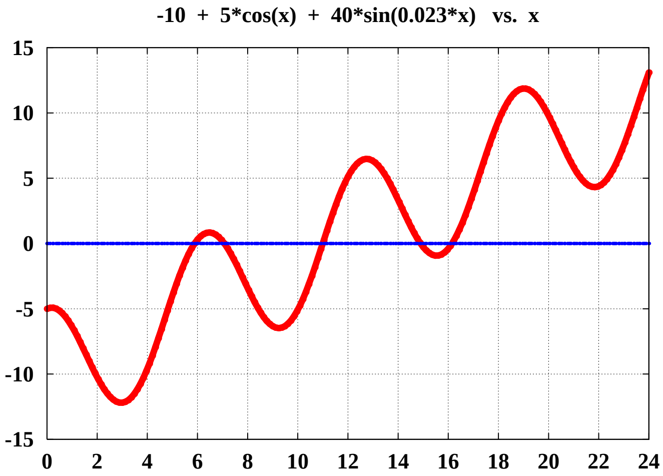
$$-10 \;+\; 5\cos(x) \;+\; 40\sin(0.023\,x) \;=\; 0 \qquad \underset{\text{(Solutions)}}{\Rightarrow} \qquad x \;\approx\; 5.88, \;\; 7.07, \;\; 11.0, \;\; 14.9, \;\; 16.15$$

As shown in the following **M**otion**G**enesis file, the solutions to this nonlinear equation is **highly sensitive** to the starting guess around $x = 6.45$, $x = 9.25$, $x = 12.75$, and $x = 15.55$. Guesses of $x < 2.5$ or $x > 20$ may not converge to a solution. Other guesses (especially near local maximum or minimum) may not converge to their closest solution.[1]

---

[1] This nonlinear equation is associated with the algorithm used for a radiation machine targeting a cancer cell.

```
   (1) % File:  HighlyNonlinearEquation.al
   (2) %------------------------------------
   (3) Variable  x
   (4) Eqn = -10 + 5*cos(x) + 40*sin(0.023*x)
-> (5) Eqn = -10 + 5*cos(x) + 40*sin(0.023*x)

   (6) ans0 = SolveNonlinear( Eqn, x = 3.2 )
-> (7) ans0 = [5.882719]

   (8) ans1 = SolveNonlinear( Eqn, x = 6.4 )
-> (9) ans1 = [5.882719]

   (10) ans2 = SolveNonlinear( Eqn, x = 6.5 )
-> (11) ans2 = [7.072387]

   (12) ans3 = SolveNonlinear( Eqn, x = 9.2 )
-> (13) ans3 = [7.072387]

   (14) ans4 = SolveNonlinear( Eqn, x = 9.3 )
-> (15) ans4 = [10.99414]

   (16) ans5 = SolveNonlinear( Eqn, x = 12.7 )
-> (17) ans5 = [10.99414]

   (18) ans6 = SolveNonlinear( Eqn, x = 12.8 )
-> (19) ans6 = [14.89506]

   (20) ans7 = SolveNonlinear( Eqn, x = 15.5 )
-> (21) ans7 = [14.89506]

   (22) ans8 = SolveNonlinear( Eqn, x = 15.6 )
-> (23) ans8 = [16.15025]

   (24) ans9 = SolveNonlinear( Eqn, x = 18.95 )
-> (25) ans9 = [16.15025]
```



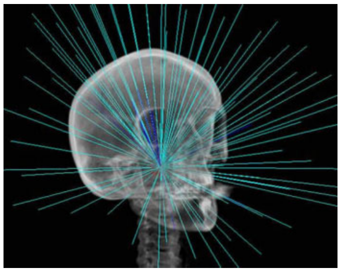
5.7.3 Continuous solutions of *nonlinear* algebraic equations

One way to find a continuous solution for x in the range $0 \le t \le 8$ for

$$x^2 \;-\; \cos^2(x) \;\;=\; 0.3\,\sin(t)$$

is to differentiate this ***nonlinear*** equation with respect to t and then solve the derivative equation that is ***linear*** in $\dot{x}$ as

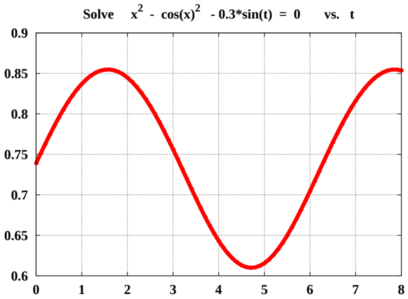
$$2\,x\,\dot{x} + 2\,\cos(x)\,\sin(x)\,\dot{x} \;=\; 0.3\,\sin(t) \qquad \Rightarrow \qquad \dot{x} \;=\; \frac{0.3\,\sin(t)}{2\,x \;+\; 2\,\cos(x)\,\sin(x)}$$



Courtesy Accuray Inc.

Solving the nonlinear equation **once** at $t = 0$ gives $x(t\!=\!0) \;\approx\; 0.74$. With this "initial" value for x and a continuous formula for $\dot{x}$, ODE techniques can be used to numerically integrate $\dot{x}(t)$ to solve for $x(t)$.

```
% File:  NonlinearSolveContinuous.al
%-----------------------------------------
Variable  x'
eqn = x^2 - cos(x)^2 - 0.3*sin(t)
Solve( Dt(eqn),  x' )
SolveSetInput(  Evaluate(eqn, t=0),   x = 1 )
Input  tFinal = 8 sec,  tStep = 0.1 sec
Output  t,  x
ODE() NonlinearSolveContinuous
Save NonlinearSolveContinuous.all
Quit
```



5.8 Solution of ordinary differential equations (ODEs)

Computer languages and software such as **M**otion**G**enesis and MATLAB® have revolutionized the **numerical solution** of ODEs. Frequently, **compiled** C and Fortran codes optimize code for a specific operating system, microprocessor, and cache and can be 100x **faster** than **interpreted** codes. This difference is significant for embedded systems that require real-time operation or when compiled code requires more than a minute to execute (which means the interpreted code may require many hours).

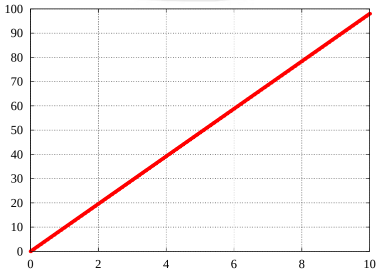
5.8.1 Solution of 1^{st}-order ODE (numerical integration)

The figure to the right shows a parachutist in vertical free-fall. When air-resistance and other forces than gravity are neglected, the parachutist's downward speed v is governed by the 1^{st}-order ODE

$$\frac{dv}{dt} = 9.8$$

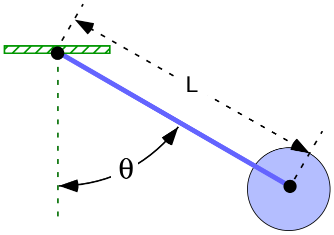
Although this ODE is easily solvable by ***separation of variables*** and integration as $v(t) = v(0) + 9.8\,t$, it can also be solved by computer numerical integration as shown in the following **M**otion**G**enesis file.

```
% File: ParachutistFreeFallSpeed.txt
%-------------------------------------
Variable v’ = 9.8
Input    v = 0        % Initial value
ODE() ParachutistFreeFallSpeed
Quit
```



Note: To generate MATLAB®, C, or Fortran code to solve the ODE, append the suffix .m, .c, or .for, to the filename. For example, to generate MATLAB® code, replace the last line with `ODE() ParachutistFreeFallSpeed.m`

5.8.2 Solution of 2^{nd}-order ODEs (numerical integration)

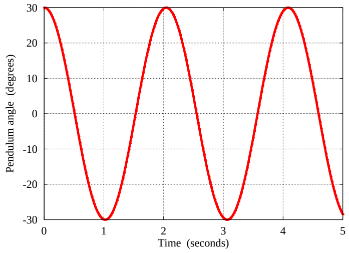


The figure to the right shows a 1 m pendulum swinging on Earth's surface. The pendulum's motion is governed by the **nonlinear** 2^{nd}-order ODE

$$\ddot{\theta} = \text{-}9.8\,\sin(\theta)$$

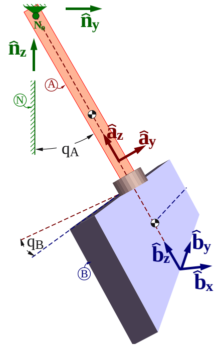
The **M**otion**G**enesis solution to this **nonlinear** ODE is shown below.

```
% File: ClassicParticlePendulumShort.al
%-----------------------------------------------------
Variable theta’’ = -9.8*sin(theta)
Input  theta = 30 deg,  theta’ = 0,  tFinal = 5, tStep = 0.02
Output  t sec,  theta deg
ODE() ClassicParticlePendulum
Quit
```



Note: To generate MATLAB®, C, or Fortran code to solve the ODE, append the suffix .m, .c, or .for, to the filename. For example, to generate MATLAB® code, replace the last line with `ODE() ClassicParticlePendulum.m`

5.8.3 Solution of *coupled* nonlinear 2^{nd}-order ODEs

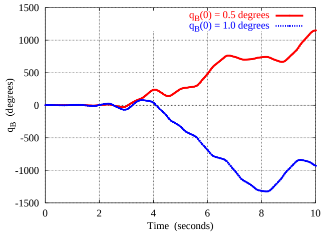


The motion of the system to the right is governed by ODEs that can exhibit "chaotic" behavior (small changes in initial values, physical parameters, or numerical integration accuracy lead to dramatically different behavior).

$$\ddot{q}_A \;=\; \frac{2\,[508.89\,\sin(q_A)\;-\;\sin(q_B)\,\cos(q_B)\,\dot{q}_A\,\dot{q}_B]}{\text{-}21.556+\sin^2(q_B)}$$

$$\ddot{q}_B \;=\; \text{-}\sin(q_B)\,\cos(q_B)\,\dot{q}_A^2$$

The following **M**otion**G**enesis commands solve these ODEs.



```
Variable  qA'', qB''      % Angles and first/second time-derivatives.
%--------------------------------------------------------------------
qA'' = 2*( 508.89*sin(qA) - sin(qB)*cos(qB)*qA'*qB' ) / (-21.556 + sin(qB)^2)
qB'' = -sin(qB)*cos(qB)*qA'^2
%--------------------------------------------------------------------
Input  tFinal = 10 sec, tStep = 0.02 sec, absError = 1.0E-07
Input  qA = 90 deg,  qB = 1.0 deg,  qA' = 0.0 rad/sec,  qB' = 0.0 rad/sec
OutputPlot  t sec,  qA degrees,  qB degrees
%--------------------------------------------------------------------
ODE() solveBabybootODE
Quit
```

Note: To generate MATLAB®, C, or Fortran code to solve the ODE, append the suffix .m, .c, or .for, to the filename. For example, to generate MATLAB® code, replace the last line with `ODE() solveBabybootODE.m`

5.8.4 Solution of coupled ODEs with additional output (spinning rigid body)

The ODEs governing 3D rotational motions of a torque-free rigid body B are:

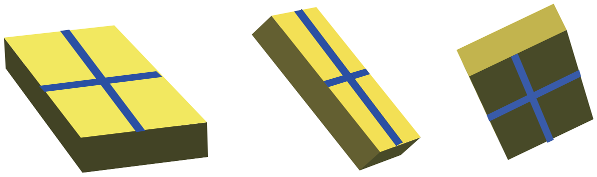
Quantity	Symbol	Value
B's central moment of inertia for $\hat{\mathbf{b}}_x$	I_{xx}	$1\ \text{kg m}^2$
B's central moment of inertia for $\hat{\mathbf{b}}_y$	I_{yy}	$2\ \text{kg m}^2$
B's central moment of inertia for $\hat{\mathbf{b}}_z$	I_{zz}	$3\ \text{kg m}^2$
$\hat{\mathbf{b}}_x$ measure of ${}^N\vec{\boldsymbol{\omega}}^B$	ω_x	Variable
$\hat{\mathbf{b}}_y$ measure of ${}^N\vec{\boldsymbol{\omega}}^B$	ω_y	Variable
$\hat{\mathbf{b}}_z$ measure of ${}^N\vec{\boldsymbol{\omega}}^B$	ω_z	Variable

$$\dot{\omega}_x \;=\; \frac{(\text{I}_{yy}-\text{I}_{zz})}{\text{I}_{xx}}\,\omega_z\,\omega_y$$

$$\dot{\omega}_y \;=\; \frac{(\text{I}_{zz}-\text{I}_{xx})}{\text{I}_{yy}}\,\omega_x\,\omega_z$$

$$\dot{\omega}_z \;=\; \frac{(\text{I}_{xx}-\text{I}_{yy})}{\text{I}_{zz}}\,\omega_y\,\omega_x$$

A **M**otion**G**enesis solution[2] to these ODEs for $0 \le t \le 4$ with initial values of $\omega_x = 7,\ \omega_y = 0.2,\ \omega_z = 0.2$ is provided below. The output from this program includes time, kinetic energy, and various measures of angular momentum, i.e., $t,\ \omega_x,\ \omega_y,\ \omega_z,\ H_x,\ H_y,\ H_z,\ H_{mag} \triangleq |\vec{\text{H}}|$, and K.



```
% File: SpinningBookODE.al (solve coupled odes)
%---------------------------------------------
Ixx = 1;   Iyy = 2;   Izz = 3;
Variable  wx', wy', wz'
wx' = ( (Iyy - Izz)*wz*wy ) / Ixx
wy' = ( (Izz - Ixx)*wx*wz ) / Iyy
wz' = ( (Ixx - Iyy)*wy*wx ) / Izz
%-- Angular momentum and rotational kinetic energy --
Hx = Ixx*wx;   Hy = Iyy*wy;    Hz = Izz*wz
Hmag = sqrt( Hx^2 + Hy^2 + Hz^2 )
K = 1/2*(Ixx*wx^2 + Iyy*wy^2 + Izz*wz^2)
%---------------------------------------------
Input   wx=7.0, wy=0.2, wz=0.2, tFinal=4
Output  t, wx, wy, wz, Hx, Hy, Hz, Hmag, K
ODE()  SpinningBook
Save   SpinningBookODE.all
Quit
```

---

[2] To produce a MATLAB® file to solve these ODEs, change the `ODE` command to `ODE() SpinningBook.m`

5.9 Matrix calculations with MotionGenesis

Matrices in MotionGenesis

```
RowMatrix    = [ 1, 2, 3 ]
ColumnMatrix = [ 1; 2; 3 ]
MatrixWithTwoRowsAndThreeColumns = [ 1, 2, 3; 4, 5, pi ]
MatrixWithThreeRowsAndTwoColumns = [ 1, 2; 3, 4; 5, pi ]
```

Matrix addition

```
A  =  [ 1, 2, 3;  4, 5, 6 ]
B  =  [ 7, 8, 9;  pi, i, t ]
AddMatrices  =  A + B
```

Multiplication of a matrix with a scalar

```
ScalarMultiplicationExample  =  7 * [ 1, 2, 3;  4, 5, 6 ]
```

Multiplication of two matrices

```
A  =  [ 11, 12, 13;   21, 22, 23 ]
B  =  [ 11, 12;   21, 22;   31, 32 ]
C  =  A * B
```

The zero matrix and identity matrix in MotionGenesis

```
A = GetZeroMatrix( 3 )         % 3x3 matrix of zeros
B = GetZeroMatrix( 2, 3 )      % 2x3 matrix of zeros
C = GetIdentityMatrix( 3 )     % 3x3 identity matrix
D = GetIdentityMatrix( 2, 3 )  % 2x3 matrix with 1 along the diagonal and 0 elsewhere
```

Partial and ordinary derivative of a matrix with MotionGenesis

```
Variables  x, y, z
A = [ x^2; x*sin(y); exp(x)*cosh(y) ]
PartialDerivativeOfAWithRespectToX = D( A, x )
PartialDerivativeOfAWithRespectToXandY = D( A, [x,y] )
```

Transpose of a matrix with MotionGenesis

```
A  =  [ 1, 2, 3;  4, 5, 6 ]
B  =  GetTranspose( A )
```

Submatrices with MotionGenesis

```
A = [1, 2, 3, 4;  5, 6, 7, 8;  9, 10, 11, 12]
B = GetRows( A, 2 )                   % 1x4 matrix with row 2 of A
C = GetRows( A, 3,1 )                 % 2x4 matrix with row 3 and row 1 of A
D = GetRows( A, 1:3, 3:2 )            % 5x4 matrix with rows 1 to 3 and rows 3 to 2 of A
F = GetColumn( A, 2 )                 % 3x1 matrix with column  2 of A
G = GetColumns(A, 2:4)                % 3x3 matrix with columns 2 to 4 of A
H = GetColumns( GetRows(A,2:3), 2 )   % 1x2 matrix with elements 2,2 and 3,2 of A
```

Determinant and inverse of a matrix with MotionGenesis

```
A  =  [ 1, 2, 3;  4, 5, 6;  7, 8, 9 ]
DeterminantOfA  =  GetDeterminant( A )
InverseOfA  =  GetInverse( A )
```

Solving linear algebraic equations with MotionGenesis (symbolic or numerical)

```
Variable  x1, x2, x3
Constant  b1, b2, b3
Zero[1] = 2*x1 + 3*x2 + 4*x3 - b1
Zero[2] = 3*x1 + 4*x2 + 5*x3 - b2
Zero[3] = 6*x1 + 7*x2 + 9*x3 - b3
Solve( Zero, x1, x2, x3 )
```

Forming matrices from linear algebraic equations with MotionGenesis

```
Constant  b1, b2, b3
Variable  x1, x2, x3
Zero[1] = 2*x1 + 3*x2 + 4*x3 - b1
Zero[2] = 3*x1 + 4*x2 + 5*x3 - b2
Zero[3] = 6*x1 + 7*x2 + 9*x3 - b3
CoefficientMatrix = D( Zero, [x1, x2, x3] )         % Forms 3x3 matrix
RemainderMatrix = Exclude( Zero, [x1, x2, x3] )     % Forms [-b1; -b2; -b3]
```

Eigenvalues and eigenvectors with MotionGenesis

```
A = [ 1, 2, 3;  4, 5, 6;  7, 8, 9 ]
eigenValuesOfA = GetEigen( A, eigenVectorsOfA )
eigenVector1 = GetColumn( eigenVectorsOfA, 1 )
eigenVector3 = GetColumn( eigenVectorsOfA, 3 )
```

Note: More at www.MotionGenesis.com ⇒ Get Started ⇒ Matrices and matrix commands.

5.10 Miscellaneous mathematical examples

Note: MotionGenesis file at www.MotionGenesis.com ⇒ Get Started ⇒ Sample Mathematics.

```
   (1) % File: MiscMathExamples.al
   (2) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (3) %------------------------------------------------------------------
   (4) %       Mathematical declarations: variables, constants
   (5) Variable   x', y'
   (6) Constant   a, b, c, d
   (7) SetImaginaryNumber( 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 = GetTaylorSeries( x*cos(y), 0:7, x=0,y=0)
-> (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 = GetPolynomial( [a,b,c], x )   % Creates a*x^2 +b*x +c
-> (27) Poly = c + b*x + a*x^2

   (28) Root1 = GetRoots( [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 = GetRoots( Poly, x, 2 )       % Some symbolic roots
-> (31) Root2[1] = -0.5*(b-sqrt(b^2-4*a*c))/a
-> (32) Root2[2] = -0.5*(b+sqrt(b^2-4*a*c))/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 = GetTranspose( M0 )
-> (63) M4 = [a, c; b, d]

   (64) M5 = GetInverse( M0 )
-> (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 = GetIdentityMatrix( 3 )
-> (70) M6 = [1, 0, 0; 0, 1, 0; 0, 0, 1]

   (71) M7 = GetZeroMatrix( 3, 3 )
-> (72) M7 = [0, 0, 0; 0, 0, 0; 0, 0, 0]

   (73) M8 = GetDiagonalMatrix(3,4, 5 )
-> (74) M8 = [5, 0, 0, 0; 0, 5, 0, 0; 0, 0, 5, 0]

   (75) C0 = GetColumns( M0, 1 )     % Returns column 1 of M0
-> (76) C0 = [a; c]

   (77) R0 = GetRows( M0, 2, 1:2)    % Returns rows 2 and rows 1 through 2 of M0
-> (78) R0 = [c, d; a, b; c, d]

   (79) N1 = GetRows( M0 )           % Returns the number of rows in M0
-> (80) N1 = 2

   (81) det = GetDeterminant( M0 )
-> (82) det = a*d - b*c

   (83) M9 = Evaluate( M0, a=1, b=2, c=3, d=4 )
-> (84) M9 = [1, 2; 3, 4]

   (85) Lambda = GetEigen( M9 )      % Eigenvalues
-> (86) Lambda = [-0.3722813; 5.372281]

   (87) EigValue = GetEigen( M9, EigVec)   % Eigenvalues/eigenvectors
-> (88) EigVec = [-0.8245648, -0.4159736; 0.5657675, -0.9093767]
-> (89) EigValue = [-0.3722813; 5.372281]

   (90) %--------------------------------------------------------------------
```

Note: MotionGenesis file at www.MotionGenesis.com ⇒ Get Started ⇒ Sample Mathematics.

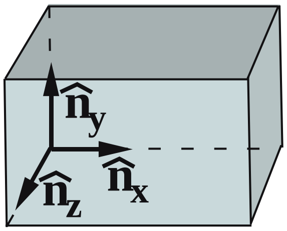
Chapter 6

Computing with vectors

6.1 MotionGenesis vector commands

Command	Description
`Cross( a>, b> )`	Returns $\vec{\mathbf{a}} \times \vec{\mathbf{b}}$
`Dot( a>, b> )`	Returns $\vec{\mathbf{a}} \cdot \vec{\mathbf{b}}$
`GetMagnitude( v> )`	Returns $\lvert\vec{\mathbf{v}}\rvert$
`GetMagnitudeSquared( v> )`	Returns $\lvert\vec{\mathbf{v}}\rvert^2$
`GetUnitVector( v> )`	Returns $\vec{\mathbf{v}} \,/\, \lvert\vec{\mathbf{v}}\rvert$
`GetAngleBetweenVectors( a>, b> )`	Returns the angle between vectors $\vec{\mathbf{a}}$ and $\vec{\mathbf{b}}$
`GetAngleBetweenVectors( Ax>, By> )`	Returns the angle between unit vectors $\widehat{\mathbf{A}}_x$ and $\widehat{\mathbf{B}}_y$
`Vector( A, x, y, z )`	Returns the vector $x\,\widehat{\mathbf{A}}_x + y\,\widehat{\mathbf{A}}_y + z\,\widehat{\mathbf{A}}_z$
`Vector( A, [x, y, z] )`	Returns the vector $x\,\widehat{\mathbf{A}}_x + y\,\widehat{\mathbf{A}}_y + z\,\widehat{\mathbf{A}}_z$

MotionGenesis vector dot-products and cross products



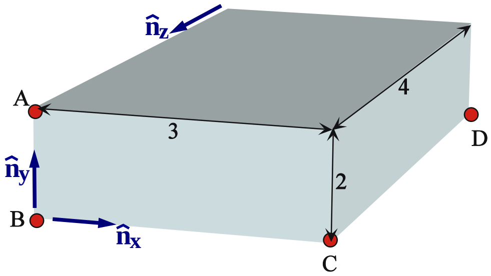
The figure to the right shows a right-handed set of orthogonal unit vectors $\widehat{\mathbf{n}}_x$, $\widehat{\mathbf{n}}_y$, $\widehat{\mathbf{n}}_z$. The vectors $\vec{\mathbf{u}}$, $\vec{\mathbf{v}}$, $\vec{\mathbf{w}}$ are defined as:

$$\begin{aligned}
\vec{\mathbf{u}} &= 2\,\widehat{\mathbf{n}}_x + 3\,\widehat{\mathbf{n}}_y + 4\,\widehat{\mathbf{n}}_z \\
\vec{\mathbf{v}} &= x\,\widehat{\mathbf{n}}_x + y\,\widehat{\mathbf{n}}_y + z\,\widehat{\mathbf{n}}_z \\
\vec{\mathbf{w}} &= 5\,\widehat{\mathbf{n}}_x - 6\,\widehat{\mathbf{n}}_y + 7\,\widehat{\mathbf{n}}_z
\end{aligned}$$

The following commands calculate $\vec{\mathbf{u}} \cdot \vec{\mathbf{v}}$, $\vec{\mathbf{u}} \cdot \vec{\mathbf{w}}$, $\vec{\mathbf{u}} \times \vec{\mathbf{v}}$, and $\vec{\mathbf{v}} \times \vec{\mathbf{w}}$.

```
% File: CalculateDotCrossProductsWithBasis.al
%---------------------------------------------
RigidFrame  N
Variable    x, y, z
u> = 2*Nx> + 3*Ny> + 4*Nz>
v> = x*Nx> + y*Ny> + z*Nz>
w> = 5*Nx> - 6*Ny> + 7*Nz>
uDotv = Dot( u>, v> )
uDotw = Dot( u>, w> )
uCrossv> = Cross( u>, v> )
vCrossw> = Cross( v>, w> )
Save CalculateDotCrossProductsWithBasis.all
Quit
```

Calculating angles with MotionGenesis

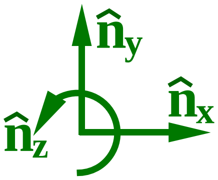


The figure to the right shows a rectangular parallelepiped (block) of sides 2, 3, and 4 with point A, B, C located at corners as shown. Right-handed orthogonal unit vectors $\widehat{\mathbf{n}}_x$, $\widehat{\mathbf{n}}_y$, $\widehat{\mathbf{n}}_z$ are directed with $\widehat{\mathbf{n}}_x$ directed from B to C and $\widehat{\mathbf{n}}_y$ from B to A.

The following commands calculate the angle between line AB and line AC.

```
% File: DotProductsToCalculateAngles.txt
%----------------------------------------
RigidFrame  N
Point       A, B, C, D
B.SetPosition( A, -2*Ny> )
C.SetPosition( B,  3*Nx> )
distanceFromAToC = C.GetDistance(A)
u> = C.GetPosition(A) / distanceFromAToC
angleBACRad = GetAngleBetweenVectors( B.GetPosition(A), C.GetPosition(A) )
angleBACDeg = angleBACRad * ConvertUnits( radians, degrees )
Save DotProductsToCalculateAngles.all
Quit
```

MotionGenesis Vector operations



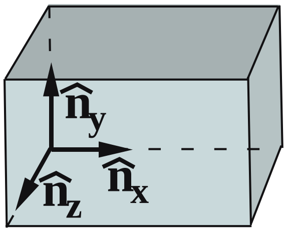
Vector operations such as addition, scalar multiplication, dot-products, and cross-products can be performed with **MotionGenesis** as shown below.

```
   (1) % File: VectorDemonstration.al
   (2) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (3) RigidFrame N                          % Create orthogonal unit vectors Nx>, Ny>, Nz>
   (4) v> = Vector( N, 7, 5, 4 )             % Construct a vector v>
-> (5) v> = 7*Nx> + 5*Ny> + 4*Nz>

   (6) w> = Vector( N, 2, 3, 2 )             % Construct a vector w>
-> (7) w> = 2*Nx> + 3*Ny> + 2*Nz>

   (8) addVW> = v> + w>                      % Vector addition
-> (9) addVW> = 9*Nx> + 8*Ny> + 6*Nz>

   (10) subtractVW> = v> - w>                 % Vector subtraction
-> (11) subtractVW> = 5*Nx> + 2*Ny> + 2*Nz>

   (12) vMultipliedBy5> = 5 * v>              % Scalar multiplication of vector v>
-> (13) vMultipliedBy5> = 35*Nx> + 25*Ny> + 20*Nz>

   (14) vDividedBy2> = v> / (-2)              % Divide vector v> by -2
-> (15) vDividedBy2> = -3.5*Nx> - 2.5*Ny> - 2*Nz>

   (16) dotVV = Dot( v>, v> )                 % Same as GetMagnitudeSquared( v> )
-> (17) dotVV = 90

   (18) dotWW = Dot( w>, w> )                 % Same as GetMagnitudeSquared( v> )
-> (19) dotWW = 17

   (20) magV = GetMagnitude( v> )             % Magnitude of v>
-> (21) magV = 9.486833

   (22) magW = GetMagnitude( w> )             % Magnitude of w>
-> (23) magW = 4.123106

   (24) unitV> = GetUnitVector( v> )          % Unit vector in the direction of v>
```

```
-> (25) unitV> = 0.7378648*Nx> + 0.5270463*Ny> + 0.421637*Nz>

   (26) unitW> = GetUnitVector( w> )           % Unit vector in the direction of w>
-> (27) unitW> = 0.4850713*Nx> + 0.7276069*Ny> + 0.4850713*Nz>

   (28) dotVW = Dot( v>, w> )                  % Dot product of v> and w>
-> (29) dotVW = 37

   (30) angleVW = GetAngleBetweenVectors( v>, w> )
-> (31) angleVW = 0.3303666

   (32) crossVW> = Cross( v>, w> )             % Cross product of v> and w>
-> (33) crossVW> = -2*Nx> - 6*Ny> + 11*Nz>

   (34) areaVW = 1/2*GetMagnitude( crossVW> )  % Area of triangle formed by v> an w>
-> (35) areaVW = 6.344289

   (36) crossVVW> = Cross( v>, Cross(v>,w>) )  % Vector triple cross product
-> (37) crossVVW> = 79*Nx> - 85*Ny> - 32*Nz>

   (38) dotVWithZeroVector  = Dot( v>, 0>  )   % Dot product of v> with the zero vector
-> (39) dotVWithZeroVector = 0

   (40) dotVWithUnitDyadic> = Dot( v>, 1>> )   % Dot product of v> with the unit dyadic
-> (41) dotVWithUnitDyadic> = 7*Nx> + 5*Ny> + 4*Nz>

   (42) multVW>> = v> * w>                     % Form a dyadic by multiplying v> and w>
-> (43) multVW>> = 14*Nx>*Nx> + 21*Nx>*Ny> + 14*Nx>*Nz> + 10*Ny>*Nx> + 15*Ny>*
        Ny> + 10*Ny>*Nz> + 8*Nz>*Nx> + 12*Nz>*Ny> + 8*Nz>*Nz>
```

6.2 More vector operations with MotionGenesis

Vector dot-products and cross-products are useful for kinematics (motion), mass-distribution calculations, forces/moments, statics, and dynamics. Use the **MotionGenesis** commands **Dot**, **Cross**, **Magnitude**, **UnitVector**, and **AngleBetweenVectors** to perform the following operations.



The figure to the left shows a right-handed set of orthogonal unit vectors $\hat{\mathbf{n}}_x$, $\hat{\mathbf{n}}_y$, $\hat{\mathbf{n}}_z$. Given below are vectors $\vec{\mathbf{v}}$ and $\vec{\mathbf{w}}$.

$$\vec{\mathbf{v}} = 2\,\hat{\mathbf{n}}_x + 3\,\hat{\mathbf{n}}_y + 4\,\hat{\mathbf{n}}_z \qquad \vec{\mathbf{w}} = 5\,\hat{\mathbf{n}}_x - 6\,\hat{\mathbf{n}}_y + 7\,\hat{\mathbf{n}}_z$$

Note: **Assign** each output quantity in **MotionGenesis** to a scalar or vector name, e.g., type:
`a> = 10*v>;` `b> = v> / 10;` `c> = v> + w>;` `d> = v> - w>`

$$
\begin{aligned}
10 * \vec{\mathbf{v}} &= 20\,\hat{\mathbf{n}}_x + 30\,\hat{\mathbf{n}}_y + 40\,\hat{\mathbf{n}}_z \\
\vec{\mathbf{v}}/10 &= 0.2\,\hat{\mathbf{n}}_x + 0.3\,\hat{\mathbf{n}}_y + 0.4\,\hat{\mathbf{n}}_z \\
\vec{\mathbf{v}} + \vec{\mathbf{w}} &= 7\,\hat{\mathbf{n}}_x - 3\,\hat{\mathbf{n}}_y + 11\,\hat{\mathbf{n}}_z \\
\vec{\mathbf{v}} - \vec{\mathbf{w}} &= \text{-}3\,\hat{\mathbf{n}}_x + 9\,\hat{\mathbf{n}}_y - 3\,\hat{\mathbf{n}}_z \\
\vec{\mathbf{v}} \cdot \vec{\mathbf{w}} &= 20 \\
\vec{\mathbf{v}} \times \vec{\mathbf{w}} &= 45\,\hat{\mathbf{n}}_x + 6\,\hat{\mathbf{n}}_y - 27\,\hat{\mathbf{n}}_z \\
\vec{\mathbf{w}} \times \vec{\mathbf{v}} &= \text{-}45\,\hat{\mathbf{n}}_x - 6\,\hat{\mathbf{n}}_y + 27\,\hat{\mathbf{n}}_z \\
|\vec{\mathbf{w}}| &= \sqrt{110} = 10.4881 \\
|\vec{\mathbf{v}}| &= \sqrt{29} = 5.38516 \\
\vec{\mathbf{v}}^2 &= 29 \\
\vec{\mathbf{v}}^3 &= 29^{3/2} = 156.1698 \\
\texttt{UnitVector}(\vec{\mathbf{v}}) &= \frac{2\,\hat{\mathbf{n}}_x + 3\,\hat{\mathbf{n}}_y + 4\,\hat{\mathbf{n}}_z}{\sqrt{29}} \\
&= 0.3714\,\hat{\mathbf{n}}_x + 0.5571\,\hat{\mathbf{n}}_y + 0.7428\,\hat{\mathbf{n}}_z \\
\angle(\vec{\mathbf{v}}, \vec{\mathbf{w}}) &= 1.20884 \text{ radians or } 69.2613^\circ \\
\vec{\mathbf{v}} * \vec{\mathbf{w}} &= 10\,\hat{\mathbf{n}}_x\,\hat{\mathbf{n}}_x - 12\,\hat{\mathbf{n}}_x\,\hat{\mathbf{n}}_y + 14\,\hat{\mathbf{n}}_x\,\hat{\mathbf{n}}_z \\
&+ 15\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_x - 18\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_y + 21\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_z \\
&+ 20\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_x - 24\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_y + 28\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_z \\
\vec{\mathbf{w}} * \vec{\mathbf{v}} &= 10\,\hat{\mathbf{n}}_x\,\hat{\mathbf{n}}_x + 15\,\hat{\mathbf{n}}_x\,\hat{\mathbf{n}}_y + 20\,\hat{\mathbf{n}}_x\,\hat{\mathbf{n}}_z \\
&- 12\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_x - 18\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_y - 24\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_z \\
&+ 14\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_x + 21\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_y + 28\,\hat{\mathbf{n}}_y\,\hat{\mathbf{n}}_z
\end{aligned}
$$

```
% File: VectorCalculations.txt
%---------------------------------
RigidFrame N
%---------------------------------
v> = 2*Nx> + 3*Ny> + 4*Nz>
w> = 5*Nx> - 6*Ny> + 7*Nz>
%---------------------------------
a> = 10 * v>
b> = v> / 10
c> = v> + w>
d> = v> - w>
e  = Dot( v>, w> )
f> = Cross( v>, w> )
g> = Cross( w>, v> )
h = GetMagnitude( w> )
i = GetMagnitude( v> )
j = GetMagnitudeSquared( v> )
k = GetMagnitude( v> )^3
l> = GetUnitvector( v> )
m = GetAngleBetweenVectors( v>, w> )
n>> = v> * w>
o>> = w> * v>
%---------------------------------
Save VectorCalculations.all
Quit
```

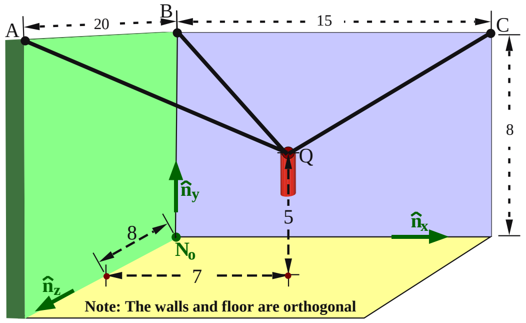
Note: More at www.MotionGenesis.com ⇒ Get Started ⇒ Vectors and vector commands.

6.3 MotionGenesis position vector commands

Command	Description and associated formula
`Q.GetPosition( No )`	Gets Q's position vector from N_o, i.e., $\vec{\mathbf{r}}^{\,Q/N_o}$.
`Q.SetPosition( P, posVector )`	Sets Q's position vector from P, i.e., $\vec{\mathbf{r}}^{\,Q/P} = \textbf{posVector}$
`Q.GetDistance( P )`	Gets Q's distance from P, i.e., $\lvert\vec{\mathbf{r}}^{\,Q/P}\rvert$.

6.3.1 MotionGenesis: Microphone cable lengths, angles, and area (orthogonal walls)

The following **M**otion**G**enesis commands determine: L_A (the length of the cable joining A and Q); the angle ϕ between line $\overline{AQ}$ and line $\overline{AB}$; the surface area $|\vec{\boldsymbol{\Delta}}|$ of the triangle formed by points A, B, Q; and a unit vector $\widehat{\mathbf{u}}$ perpendicular to the surface area.



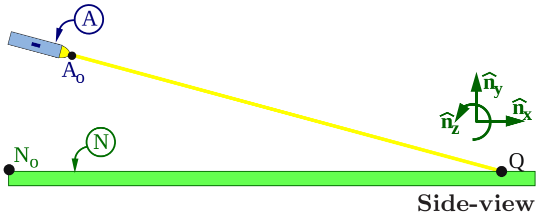
Note: The walls and floor are orthogonal

```
% File: MicrophoneCableLengthsOrthogonalWalls.al
%-----------------------------------------------------
RigidFrame  N          % Back Wall
Point       A, B, C    % Points attached to walls
Particle    Q          % Microphone attached to cables
%-----------------------------------------------------
%       Given position vectors
B.SetPosition( No,    8*Ny>  )
C.SetPosition( B,    15*Nx>  )
A.SetPosition( B,    20*Nz>  )
Q.SetPosition( No,   7*Nx> + 5*Ny> + 8*Nz>   )
%-----------------------------------------------------
LA = Q.GetDistance( A )
%-----------------------------------------------------
phiRadians = GetAngleBetweenVectors( Q.GetPosition(A), B.GetPosition(A) )
phiDegrees = phiRadians * ConvertUnits( radian, degree )
%-----------------------------------------------------
AreaABQ> = 1/2 * Cross( A.GetPosition(B), Q.GetPosition(B) )
AreaABQ = GetMagnitude( AreaABQ> )
UnitVectorPerpendicularToTriangleABQ> = GetUnitVector( AreaABQ> )
Save MicrophoneCableLengthsOrthogonalWalls.all
Quit
```

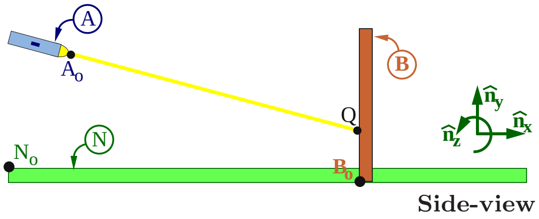
6.3.2 MotionGenesis: Position vectors and geometry (single basis)

Each problem below shows a laser A pointing at an object with a beam that passes through a point A_o in the direction of $3\,\widehat{\mathbf{n}}_x - \widehat{\mathbf{n}}_y + \widehat{\mathbf{n}}_z$ where $\widehat{\mathbf{n}}_x$, $\widehat{\mathbf{n}}_y$, $\widehat{\mathbf{n}}_z$ are right-handed orthogonal unit vectors fixed in a flat horizontal plane N with $\widehat{\mathbf{n}}_x$ horizontally right and $\widehat{\mathbf{n}}_y$ vertically upward (perpendicular to N). The position vector of A_o from N_o (a point fixed in N) is $\vec{\mathbf{r}}^{\,A_o/N_o} = \widehat{\mathbf{n}}_x + 2\,\widehat{\mathbf{n}}_y$.

Consider a laser beam that hits point Q of N.
Find the distance from A_o to Q.
Result: $\lvert\vec{\mathbf{r}}^{\,Q/A_o}\rvert = 6.63$

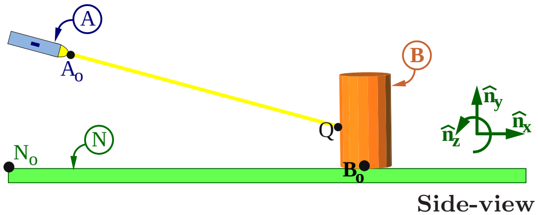


A laser beam hits point Q of a vertical wall B that is in the $\widehat{\mathbf{n}}_y$-$\widehat{\mathbf{n}}_z$ plane (the wall is perpendicular to $\widehat{\mathbf{n}}_x$). The wall is 5 units from N_o and its lower edge is parallel to $\widehat{\mathbf{n}}_z$. Find the distance from A_o to Q.
Result: $\lvert\vec{\mathbf{r}}^{\,Q/A_o}\rvert = 4.42$



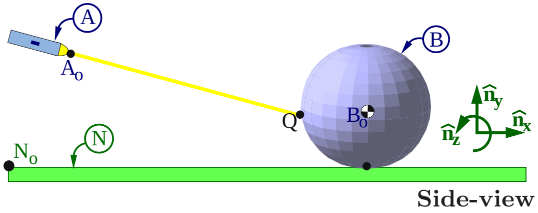
A laser beam hits point Q of a vertical cylinder B of radius 1. The point of B's symmetry axis in contact with N is denoted B_o and its position from N_o is $\vec{\mathbf{r}}^{\,B_o/N_o} = 5\,\hat{\mathbf{n}}_x + 2\,\hat{\mathbf{n}}_z$. Find the distance from A_o to Q.

Result: $\left|\vec{\mathbf{r}}^{\,Q/A_o}\right| = 3.83$

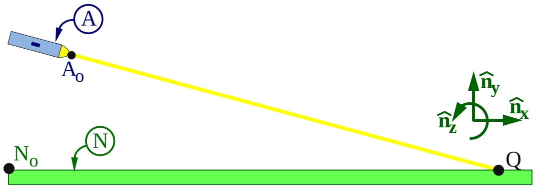


A laser beam hits point Q of a sphere B of radius 1. The position to the point of B in contact with N is $5\,\hat{\mathbf{n}}_x + 2\,\hat{\mathbf{n}}_z$. Find the distance from A_o to Q.

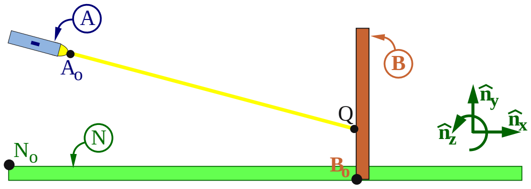
Result: $\left|\vec{\mathbf{r}}^{\,Q/A_o}\right| = 3.85$



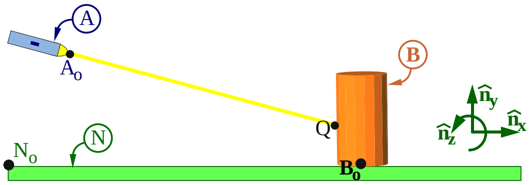
```
% File: LaserBeamOnHorizontalPlane.txt
%------------------------------------------------------
RigidFrame N   % Horizontal plane.
RigidFrame A   % Laser (also creates point Ao).
Point      Q   % Point of beam on horizontal plane.
%------------------------------------------------------
Ao.SetPosition( No, Nx> + 2*Ny> )     % Given info.
LaserDirection> = 3*Nx> - Ny> + Nz>   % Given info.
%------------------------------------------------------
Variable   d   % Introduce unknown to write position vector.
Q.SetPosition( Ao, d * GetUnitVector( LaserDirection> ) )
%------------------------------------------------------
ZeroVertical = Dot( P_No_Ao> + P_Ao_Q>, Ny> )
Solve( ZeroVertical, d )  % Solve for d (Q's distance from Ao).
%------------------------------------------------------
Save LaserBeamOnHorizontalPlane.all
Quit
```



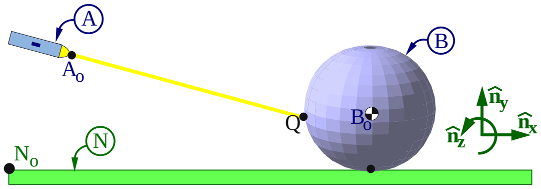
```
% File: LaserBeamOnVerticalWall.al
%------------------------------------------------------
RigidFrame N   % Horizontal plane.
RigidFrame A   % Laser (also creates point Ao).
RigidFrame B   % Vertical wall (also creates point Bo).
Point      Q   % Point of beam that hits vertical wall.
%------------------------------------------------------
%       Known position vector and direction of laser.
Ao.SetPosition( No, Nx> + 2*Ny> )
LaserDirection> = 3*Nx> - Ny> + Nz>
%------------------------------------------------------
%       Position vector to point Bo at base of vertical wall.
Bo.SetPosition( No, 5*Nx> )
%------------------------------------------------------
%       Introduce unknowns to be able to write position vectors.
Variable     Lambda, y, z
Q.SetPosition( Ao, Lambda * LaserDirection> )
Q.SetPosition( Bo, y*Ny> + z*Nz> )
%------------------------------------------------------
%       Solve for unknowns and determine Q's distance from Ao.
Loop> = P_No_Ao> + P_Ao_Q> + P_Q_Bo> + P_Bo_No>
Zero[1] = Dot( Loop>, Nx> )
Zero[2] = Dot( Loop>, Ny> )
Zero[3] = Dot( Loop>, Nz> )
Solve( Zero, Lambda, y, z )
DistanceBetweenQAndAo = Explicit( Q.GetDistance(Ao) )
%------------------------------------------------------
%       Record input and program responses
Save LaserBeamOnVerticalWall.all
Quit
```



```
% File: LaserBeamOnVerticalCylinder.al
%---------------------------------------------------------
RigidFrame N   % Horizontal plane.
RigidFrame A   % Laser (also creates point Ao).
RigidFrame B   % Vertical cylinder (also creates point Bo).
Point      Q   % Point of beam that hits vertical cylinder.
%---------------------------------------------------------
%       Known position vector and direction of laser.
Ao.SetPosition( No, Nx> + 2*Ny> )
LaserDirection> = 3*Nx> - Ny> + Nz>
%---------------------------------------------------------
%       Position vector to point Bo from No.
Bo.SetPosition( No, 5*Nx> + 2*Nz> )
%---------------------------------------------------------
%       Introduce unknowns to be able to write position vectors.
Variable     Lambda, x, y, z
Q.SetPosition( Ao, Lambda * LaserDirection> )
Q.SetPosition( Bo, x*Nx> + y*Ny> + z*Nz> )
%---------------------------------------------------------
%       Solve for unknowns and determine Q's distance from Ao.
Loop> = P_No_Ao> + P_Ao_Q> + P_Q_Bo> + P_Bo_No>
Zero[1] = Dot( Loop>, Nx> )
Zero[2] = Dot( Loop>, Ny> )
Zero[3] = Dot( Loop>, Nz> )
Solve( Zero, Lambda, y, z )
EqnOfVerticalCylinderOfRadius1 = x^2 + z^2 - 1
x = GetQuadraticNegativeRoot( EqnOfVerticalCylinderOfRadius1, x )
DistanceBetweenQAndAo = Explicit( Q.GetDistance(Ao) )
%---------------------------------------------------------
%       Record input and program responses
Save LaserBeamOnVerticalCylinder.all
Quit
```



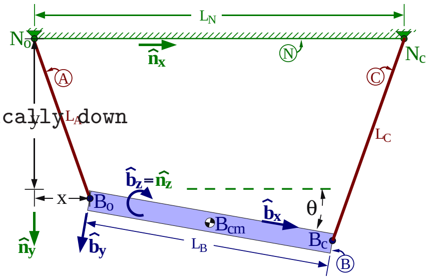
```
% File: LaserBeamOnSphere.al
%----------------------------------------------
RigidFrame N   % Horizontal plane.
RigidFrame A   % Laser (also creates point Ao).
RigidFrame B   % Sphere (also creates point Bo).
Point      Q   % Point of beam that hits sphere.
%----------------------------------------------
%       Known position vector and direction of laser.
Ao.SetPosition( No, Nx> + 2*Ny> )
LaserDirection> = 3*Nx> - Ny> + Nz>
%----------------------------------------------
%       Position vector to point Bo (center of sphere) from No.
Bo.SetPosition( No, 5*Nx> + Ny> + 2*Nz> )
%----------------------------------------------
%       Introduce unknowns to write position vectors.
Variable     Lambda, x, y, z
Q.SetPosition( Ao, Lambda * LaserDirection> )
Q.SetPosition( Bo, x*Nx> + y*Ny> + z*Nz> )
%----------------------------------------------
%       Solve for unknowns and determine Q's distance from Ao.
Loop> = P_No_Ao> + P_Ao_Q> + P_Q_Bo> + P_Bo_No>
Zero[1] = Dot( Loop>, Nx> )
Zero[2] = Dot( Loop>, Ny> )
Zero[3] = Dot( Loop>, Nz> )
Solve( Zero, Lambda, y, z )
EqnOfSphereOfRadius1 = x^2 + y^2 + z^2 - 1
x = GetQuadraticNegativeRoot( EqnOfSphereOfRadius1, x )
DistanceBetweenQAndAo = Explicit( Q.GetDistance(Ao) )
%-----------------------------------------------
%       Record input and program responses
Save LaserBeamOnSphere.all
Quit
```



6.4 MotionGenesis rotation commands

Command	Description
`B.GetRotationMatrix( A )`	Gets the ${}^{B}R^{A}$ rotation matrix
`B.SetRotationMatrixX( A, theta )`	Assigns ${}^{B}R^{A}$ for B rotated in A about $\widehat{\mathbf{b}}_x = \widehat{\mathbf{a}}_x$ by an angle θ
`B.SetRotationMatrixY( A, theta )`	Assigns ${}^{B}R^{A}$ for B rotated in A about $\widehat{\mathbf{b}}_y = \widehat{\mathbf{a}}_y$ by an angle θ
`B.SetRotationMatrixZ( A, theta )`	Assigns ${}^{B}R^{A}$ for B rotated in A about $\widehat{\mathbf{b}}_z = \widehat{\mathbf{a}}_z$ by an angle θ
`B.SetRotationMatrixNegativeX( A, theta )`	Assigns ${}^{B}R^{A}$ for B rotated in A about $-\widehat{\mathbf{b}}_x = -\widehat{\mathbf{a}}_x$ by an angle θ
`B.SetRotationMatrixNegativeY( A, theta )`	Assigns ${}^{B}R^{A}$ for B rotated in A about $-\widehat{\mathbf{b}}_y = -\widehat{\mathbf{a}}_y$ by an angle θ
`B.SetRotationMatrixNegativeZ( A, theta )`	Assigns ${}^{B}R^{A}$ for B rotated in A about $-\widehat{\mathbf{b}}_z = -\widehat{\mathbf{a}}_z$ by an angle θ
`B.RotateX( A, theta )`	Assigns ${}^{B}R^{A}$ and possibly ${}^{A}\vec{\boldsymbol{\omega}}^{B}$, ${}^{A}\vec{\boldsymbol{\alpha}}^{B}$, etc.
`B.RotateY( A, theta )`	Assigns ${}^{B}R^{A}$ and possibly ${}^{A}\vec{\boldsymbol{\omega}}^{B}$, ${}^{A}\vec{\boldsymbol{\alpha}}^{B}$, etc.
`B.RotateZ( A, theta )`	Assigns ${}^{B}R^{A}$ and possibly ${}^{A}\vec{\boldsymbol{\omega}}^{B}$, ${}^{A}\vec{\boldsymbol{\alpha}}^{B}$, etc.
`B.RotateNegativeX( A, theta )`	Assigns ${}^{B}R^{A}$ and possibly ${}^{A}\vec{\boldsymbol{\omega}}^{B}$, ${}^{A}\vec{\boldsymbol{\alpha}}^{B}$, etc.
`B.RotateNegativeY( A, theta )`	Assigns ${}^{B}R^{A}$ and possibly ${}^{A}\vec{\boldsymbol{\omega}}^{B}$, ${}^{A}\vec{\boldsymbol{\alpha}}^{B}$, etc.
`B.RotateNegativeZ( A, theta )`	Assigns ${}^{B}R^{A}$ and possibly ${}^{A}\vec{\boldsymbol{\omega}}^{B}$, ${}^{A}\vec{\boldsymbol{\alpha}}^{B}$, etc.

6.4.1 Cable lengths to position a beam (with rotation matrix).

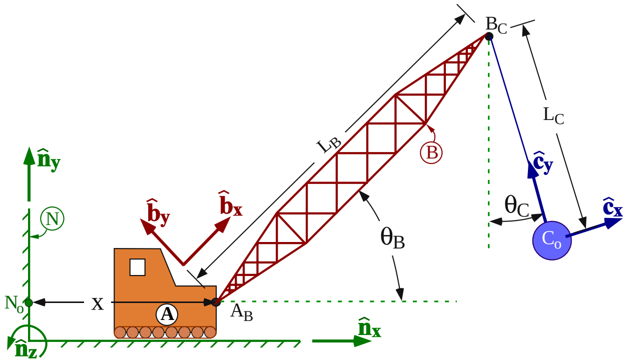


```
%    File: BeamOnTwoCablesLengths.al
% Problem: Cable lengths/rates for given beam position/orientation.
%--------------------------------------------------------------------
NewtonianFrame N         % Ceiling with Nx> horizontally right, Ny> vertically down
RigidBody      B         % Beam with Bx> pointing from Bo to Bc
Point          Nc( N )   % Point of N attached to cable C
Point          Bc( B )   % Point of B attached to cable C
%--------------------------------------------------------------------
Constant  LN = 6 m       % Distance between No and NC
Constant  LB = 4 m       % Distance between Bo and BC
Variable  x'             % Nx> measure of Bo's position from No
Variable  y'             % Ny> measure of Bo's position from No
Variable  q'             % Bz> measure of angle from Nx> to Bx>
%-----------------------------------------------------------
B.RotateZ( N, q )
Nc.SetPosition( No, LN*Nx> )
Bo.SetPosition( No, x*Nx> + y*Ny> )
Bc.SetPosition( Bo, LB*Bx> )
%-----------------------------------------------------------
LASquared = Bo.GetDistanceSquared( No )
LCSquared = Bc.GetDistanceSquared( Nc )
LA = EvaluateAtInput( sqrt(LASquared), x=1, y=2.5, q=15 deg )
LC = EvaluateAtInput( sqrt(LCSquared), x=1, y=2.5, q=15 deg )
%-----------------------------------------------------------
DtLASquared = Dt( LASquared )
DtLCSquared = Dt( LCSquared )
LAdt = EvaluateAtInput( DtLASquared / (2*LA), x=1, y=2.5, q=15 deg, x'=0, y'=2, q'=0.2 rad/sec )
LCdt = EvaluateAtInput( DtLCSquared / (2*LC), x=1, y=2.5, q=15 deg, x'=0, y'=2, q'=0.2 rad/sec )
Save BeamOnTwoCablesLengths.all
Quit
```

6.4.2 Calculating rotation matrices for a crane and wrecking ball with MotionGenesis

The following **M**otion**G**enesis commands calculate rotation matrices relating unit vectors $\widehat{\mathbf{n}}_x$, $\widehat{\mathbf{n}}_y$, $\widehat{\mathbf{n}}_z$ and $\widehat{\mathbf{b}}_x$, $\widehat{\mathbf{b}}_y$, $\widehat{\mathbf{b}}_z$ and $\widehat{\mathbf{c}}_x$, $\widehat{\mathbf{c}}_y$, $\widehat{\mathbf{c}}_z$.

```
% File:  CraneRotationMatrices.al
%------------------------------------
RigidFrame N, B, C
Variable   qB, qC
B.RotateZ( N, qB )
C.RotateZ( N, qC )
BC = B.GetRotationMatrix( C )
Save CraneRotationMatrices.all
Quit
```



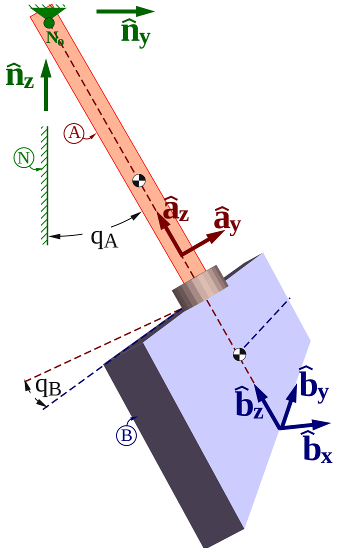
MotionGenesis output results

```
   (1) % File:  CraneRotationMatrices.al
   (2) %------------------------------------
   (3) RigidFrame N, B, C
   (4) Variable   qB, qC
   (5) B.RotateZ( N, qB )
-> (6) B_N = [cos(qB), sin(qB), 0; -sin(qB), cos(qB), 0; 0, 0, 1]

   (7) C.RotateZ( N, qC )
-> (8) C_N = [cos(qC), sin(qC), 0; -sin(qC), cos(qC), 0; 0, 0, 1]

   (9) BC = B.GetRotationMatrix( C )
-> (10) BC = [cos(qB-qC), sin(qB-qC), 0; -sin(qB-qC), cos(qB-qC), 0; 0, 0, 1]
```

6.4.3 Calculating rotation matrices for a chaotic double-pendulum with MotionGenesis

The following **M**otion**G**enesis commands calculate rotation matrices relating unit vectors $\widehat{\mathbf{n}}_x$, $\widehat{\mathbf{n}}_y$, $\widehat{\mathbf{n}}_z$ and $\widehat{\mathbf{a}}_x$, $\widehat{\mathbf{a}}_y$, $\widehat{\mathbf{a}}_z$ and $\widehat{\mathbf{b}}_x$, $\widehat{\mathbf{b}}_y$, $\widehat{\mathbf{b}}_z$.

```
   (1) % File: BabybootRotationMatrices.txt
   (2) %---------------------------------------
   (3) RigidFrame N        % Reference frame
   (4) RigidBody  A        % Upper rod
   (5) RigidBody  B        % Lower plate
   (6) Variable   qA       % Pendulum angle
   (7) Variable   qB       % Plate angle
   (8) %---------------------------------------
   (9) A.RotateX( N, qA )   % A rotates "about +x" in N  by qA
-> (10) A_N = [1, 0, 0; 0, cos(qA), sin(qA); 0, -sin(qA), cos(qA)]

   (11) B.RotateZ( A, qB )   % B rotates "about +z" in A  by qB
-> (12) B_A = [cos(qB), sin(qB), 0; -sin(qB), cos(qB), 0; 0, 0, 1]

   (13) B_N = B.GetRotationMatrix( N )
-> (14) B_N[1,1] = cos(qB)
-> (15) B_N[1,2] = sin(qB)*cos(qA)
-> (16) B_N[1,3] = sin(qA)*sin(qB)
-> (17) B_N[2,1] = -sin(qB)
-> (18) B_N[2,2] = cos(qA)*cos(qB)
-> (19) B_N[2,3] = sin(qA)*cos(qB)
-> (20) B_N[3,1] = 0
-> (21) B_N[3,2] = -sin(qA)
-> (22) B_N[3,3] = cos(qA)
```



Related: Sections 6.5 and 9.12 and www.MotionGenesis.com ⇒ Get Started ⇒ Chaotic pendulum.

6.4.4 Rotation matrices and vertical displacement of a bifilar pendulum.

Bifilar and trifilar pendulum are used to determine inertia properties of rigid bodies (e.g., aircraft, spacecraft, and biological structures such as humans limbs). The following shows a rigid human bone B suspended by two inextensible cables A_1 and A_2, each of which is attached to a flat ceiling N.

- Cable A_1 attaches to the ceiling at point N_1 of N and to the bone at point B_1 of B.
- Cable A_2 attaches to the ceiling at point N_2 of N and to the bone at point B_2 of B.
- Point N_o of N is centered between N_1 and N_2.
- Point B_o of B is centered between B_1 and B_2.
- Point B_{cm} (B's center of mass) and point B_o ***always*** lie directly below N_o.
- Initially, B_i lies directly below N_i (i=1, 2), respectively.
- B is rotated by an angle θ about the vertical line through B_o and N_o.

Draw right-handed sets of orthogonal unit vectors $\widehat{\mathbf{b}}_x$, $\widehat{\mathbf{b}}_y$, $\widehat{\mathbf{b}}_z$ and $\widehat{\mathbf{n}}_x$, $\widehat{\mathbf{n}}_y$, $\widehat{\mathbf{n}}_z$ are fixed in B and N, with:

- $\widehat{\mathbf{b}}_x$ directed from B_1 to B_2
- $\widehat{\mathbf{n}}_x$ directed from N_1 to N_2
- $\widehat{\mathbf{n}}_y = \widehat{\mathbf{b}}_y$ directed vertically upward from B_o to N_o

Knowing B is subjected to a right-handed rotation by an angle θ about $+\widehat{\mathbf{n}}_y$, complete the ${}^bR^n$ rotation matrix.

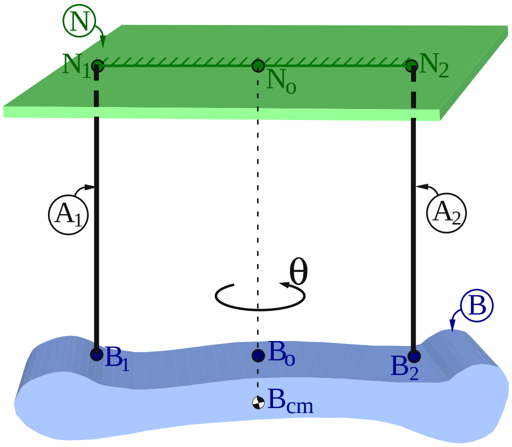
${}^bR^n$	$\widehat{\mathbf{n}}_x$	$\widehat{\mathbf{n}}_y$	$\widehat{\mathbf{n}}_z$
$\widehat{\mathbf{b}}_x$	$\cos(\theta)$	0	$-\sin(\theta)$
$\widehat{\mathbf{b}}_y$	0	1	0
$\widehat{\mathbf{b}}_z$	$\sin(\theta)$	0	$\cos(\theta)$

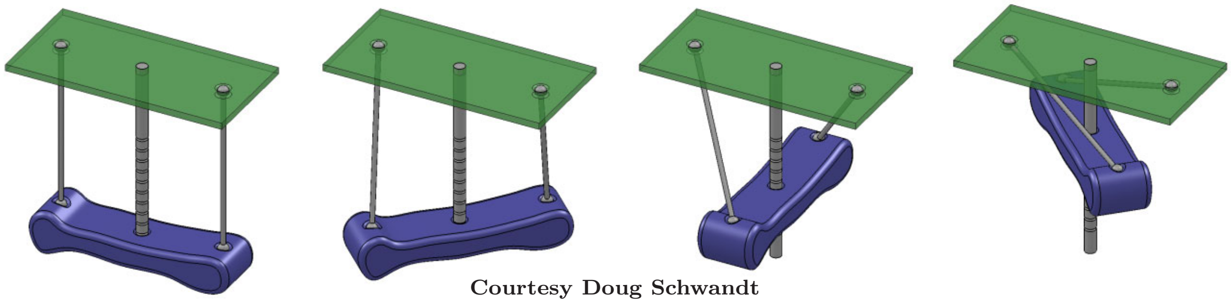
Relate y to L, h, and θ (defined in the following table).

Result: $y^2 + \frac{1}{2} L^2 [1 - \cos(\theta)] - h^2 = 0$

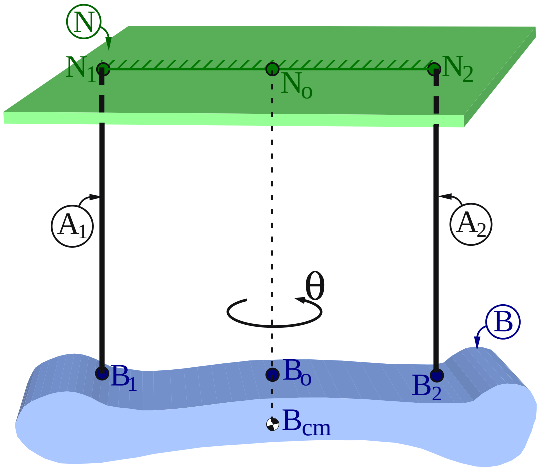
Calculate numerical values for y and $\dot{y}$ (3 significant digits).

Description	Symbol	Value
Distance between N_1 and N_2	L	1 m
Distance between N_i and B_i (i=1, 2)	h	1 m
B's rotation angle in N	θ	135°
B's rotation rate in N	$\dot{\theta}$	0.5 $\frac{\text{rad}}{\text{sec}}$
Distance between N_o and B_o	y	0.383 m
Time-derivative of y	$\dot{y}$	-0.231 $\frac{\text{m}}{\text{s}}$

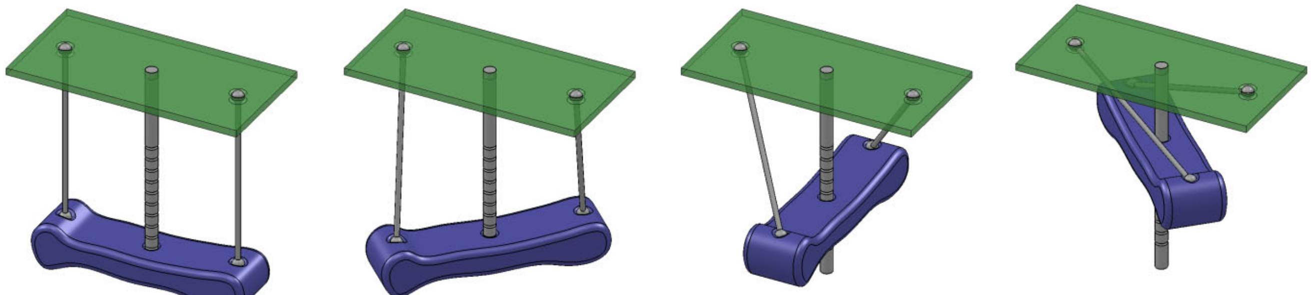




Courtesy Doug Schwandt

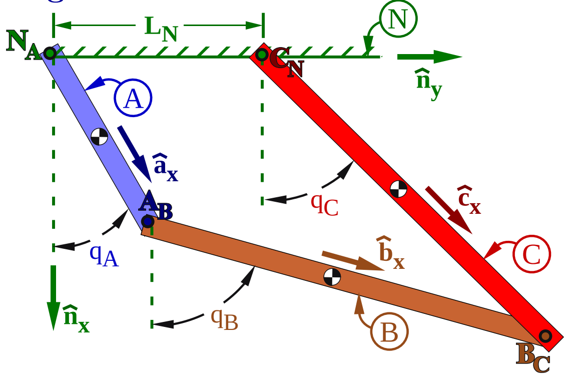


```
% File: BifilarPendulumYInTermsOfTheta.al
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%----------------------------------------
NewtonianFrame  N
RigidBody       B
Point           N1(N),  N2(N)     % N1 and N2 are fixed on N.
Point           B1(B),  B2(B)     % B1 and B2 are fixed on B.
%----------------------------------------
Constant   L = 1.0 m
Constant   h = 1.0 m
Variable   y'
Specified  theta'
%----------------------------------------
B.RotateY( N, theta )
N1.SetPosition( No,  -L/2*Nx> )   % Nx> points from No to N2
Bo.SetPosition( No,    -y*Ny> )   % Ny> = By> is vertical
B1.SetPosition( Bo,  -L/2*Bx> )   % Bx> points from Bo to B2
%--------------------------------------------
zeroDistanceSquared1 = B1.GetDistanceSquared( N1 ) - h^2
SolveQuadraticPositiveRootDt( zeroDistanceSquared1, y )
%--------------------------------------------
%    Determine y and y' for given values of theta, theta'
yValue = EvaluateAtInput( y,  theta = 135 deg )
yDtValue = EvaluateAtInput( y',  y = yValue,  theta = 135 deg,  theta' = 0.5 rad/sec )
%--------------------------------------------
N2.SetPosition( No,  L/2*Nx> )
B2.SetPosition( Bo,  L/2*Bx> )
zeroDistanceSquared2 = B2.GetDistanceSquared( N2 ) - h^2
ShouldBeEqual = Rhs( zeroDistanceSquared2 - zeroDistanceSquared1 )
Save BifilarPendulumYInTermsOfTheta.all
Quit
```



Courtesy Doug Schwandt

6.4.5 Rotation matrices and four-bar linkage configuration.



The figure to the right is a planar four-bar linkage consisting of uniform rigid links A, B, 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 $\hat{\mathbf{a}}_i$, $\hat{\mathbf{b}}_i$, $\hat{\mathbf{c}}_i$, $\hat{\mathbf{n}}_i$ $(i = x, y, z)$ are fixed in A, B, C, N, with $\hat{\mathbf{a}}_x$ directed from N_A to A_B, $\hat{\mathbf{b}}_x$ from A_B to B_C, $\hat{\mathbf{c}}_x$ from C_N to B_C, $\hat{\mathbf{n}}_x$ vertically downward, $\hat{\mathbf{n}}_y$ from N_A to C_N, and $\hat{\mathbf{a}}_z = \hat{\mathbf{b}}_z = \hat{\mathbf{c}}_z = \hat{\mathbf{n}}_z$ parallel to the axes of the revolute joints.

Quantity	Symbol	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
Angle from $\hat{\mathbf{n}}_x$ to $\hat{\mathbf{a}}_x$	q_A	Variable
Angle from $\hat{\mathbf{n}}_x$ to $\hat{\mathbf{b}}_x$	q_B	Variable
Angle from $\hat{\mathbf{n}}_x$ to $\hat{\mathbf{c}}_x$	q_C	Variable

Engineering convention: Angles are drawn **positive**.

Create a vector "***loop equation***" using a sum of position vectors that start and end at point N_A.

Result:

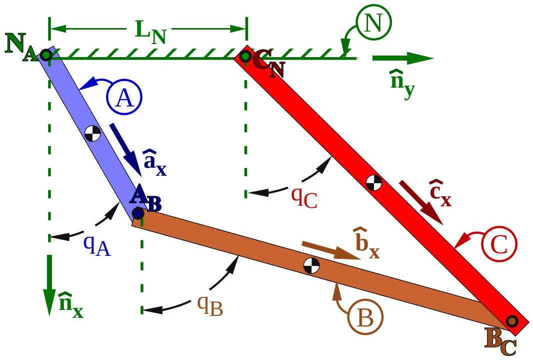
$$L_A\,\hat{\mathbf{a}}_x \;+\; L_B\,\hat{\mathbf{b}}_x \;+\; \text{-}L_C\,\hat{\mathbf{c}}_x \;+\; \text{-}L_N\,\hat{\mathbf{n}}_y \;=\; \vec{0}$$

Dot the loop equation with $\hat{\mathbf{n}}_x$ and $\hat{\mathbf{n}}_y$ to create two equations $f_i = 0$ $(i = 1, 2)$ that relate q_A, q_B, and q_C.[1] Next, determine values of q_B and q_C that satisfy these two equations when $q_A = 30°$.

Result:

Equations relating q_A, q_B, q_C.	Values when $q_A = 30°$
$f_1 = L_A * \cos(q_A) + L_B * \cos(q_B) - L_C * \cos(q_C)$	$q_B = 74.4775°$
$f_2 = L_A * \sin(q_A) + L_B * \sin(q_B) - L_C * \sin(q_C) - L_N$	$q_C = 45.5225°$

If $L_A < 1$ m, link A can be driven completely around, whereas if $L_A > 1$ m, it can only be driven 90°.



```
%    File: FourBarConfiguration.al
% Problem: Solve for qB and qC from given value of qA
%--------------------------------------------
RigidBody  A, B, C, N
Variable   qA, qB, qC
Constant   LA = 1 m,  LB = 2 m,  LC = 2 m,  LN = 1 m
%--------------------------------------------
%       Rotational kinematics
A.RotateZ( N, qA )
B.RotateZ( N, qB )
C.RotateZ( N, qC )
%--------------------------------------------
%       Configuration constraints
Loop> = LA*Ax> + LB*Bx> - LC*Cx> - LN*Ny>
Loop[1] = Dot( Loop>, Nx> )
Loop[2] = Dot( Loop>, Ny> )
%--------------------------------------------
%       Solve constraints with given constants, variables, etc.
Input  qA = 30 deg
Solve( Loop, qB = 60 deg, qC = 20 deg )
%--------------------------------------------
%       Save input together with program responses
Save FourBarConfiguration.all
Quit
```

Related: Statics/dynamics in Section 9.22 and www.MotionGenesis.com ⇒ Get Started ⇒ Four-bar linkage.

---

[1] Dot-products can be calculated by definition (inspection of the figure) or with rotation matrices.

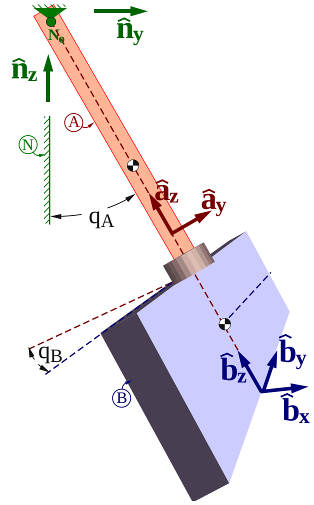
6.5 MotionGenesis angular velocity and angular acceleration commands

Command	Description
`A.SetAngularVelocity(N, theta'*Ax> )`	Assigns A's angular velocity in N. ${}^N\vec{\boldsymbol{\omega}}^A = \dot{\theta}\,\widehat{\mathbf{a}}_x$
`B.GetAngularVelocity(N)`	Calculates ${}^N\vec{\boldsymbol{\omega}}^B$, B's angular velocity in N
`A.SetAngularAcceleration(N,  theta''*Ax> )`	Assigns A's angular acceleration in N. ${}^N\vec{\boldsymbol{\alpha}}^A = \ddot{\theta}\,\widehat{\mathbf{a}}_x$
`B.GetAngularAcceleration(N)`	Calculates ${}^N\vec{\boldsymbol{\alpha}}^B$, B's angular acceleration in N
`A.SetAngularVelocityAcceleration(N, theta'*Ax> )`	Assigns ${}^N\vec{\boldsymbol{\omega}}^A = \dot{\theta}\,\widehat{\mathbf{a}}_x$ and ${}^N\vec{\boldsymbol{\alpha}}^A = \ddot{\theta}\,\widehat{\mathbf{a}}_x$
Note: N, A, and B have been named in a RigidFrame, RigidBody, or NewtonianFrame declaration.	

Calculating angular velocity and angular acceleration with MotionGenesis

The following **M**otion**G**enesis commands calculate the angular velocity and angular acceleration of rigid bodies A and B in reference frame N.

```
   (1) % File: BabybootAngularVelocityAngularAcceleration.txt
   (2) %--------------------------------------------------------
   (3) RigidFrame N      % Reference frame
   (4) RigidBody  A      % Upper rod
   (5) RigidBody  B      % Lower plate
   (6) Variable   qA''   % Pendulum angle and its time-derivatives
   (7) Variable   qB''   % Plate angle and its time-derivative
   (8) %--------------------------------------------------------
   (9) %       Angular velocity and angular acceleration
   (10) A.SetAngularVelocityAcceleration( N, qA'*Ax> )
-> (11) w_A_N> = qA'*Ax>
-> (12) alf_A_N> = qA''*Ax>

   (13) B.SetAngularVelocityAcceleration( A, qB'*Az> )
-> (14) w_B_A> = qB'*Az>
-> (15) alf_B_A> = qB''*Az>

   (16) w_B_N> = B.GetAngularVelocity( N )
-> (17) w_B_N> = qA'*Ax> + qB'*Az>

   (18) alf_B_N> =  B.GetAngularAcceleration( N )
-> (19) alf_B_N> = qA''*Ax> - qA'*qB'*Ay> + qB''*Az>
```

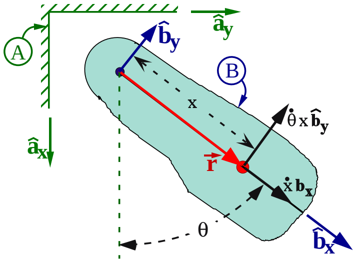


Related: Dynamics in Section 9.12 and www.MotionGenesis.com ⇒ Get Started ⇒ Chaotic pendulum.

6.6 MotionGenesis: Calculating vector derivatives

The **M**otion**G**enesis `Dt` and `D` commands calculate ordinary and partial derivatives. As shown below, `Dt` calculates the ordinary time-derivative of the vector $x\,\hat{\mathbf{b}}_\mathrm{x}$ in rigid body B and in reference frame A.

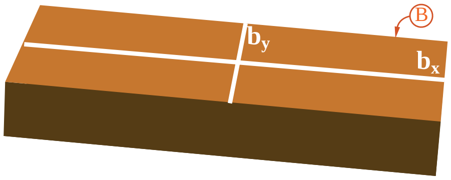
```
   (1) % File: VectorDerivativeExample.txt
   (2) %--------------------------------------
   (3) RigidFrame A         % Plane
   (4) RigidBody  B         % Rigid body
   (5) Variable   x'        % Declares x and its time-derivative x'
   (6) Variable   theta'    % Declares the angle theta and its time-derivative
   (7) %--------------------------------------
   (8) B.RotateZ( A, theta )
-> (9) B_A = [cos(theta), sin(theta), 0; -sin(theta), cos(theta), 0; 0, 0, 1]
-> (10) w_B_A> = theta'*Bz>

   (11) %--------------------------------------
   (12) r> = x*Bx>
-> (13) r> = x*Bx>

   (14) DerivativeOfrInB> = Dt( r>, B )
-> (15) DerivativeOfrInB> = x'*Bx>

   (16) DerivativeOfrInA> = Dt( r>, A )
-> (17) DerivativeOfrInA> = x'*Bx> + x*theta'*By>
```

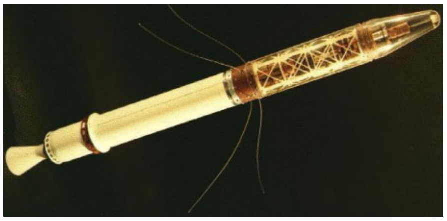

A
$\hat{\mathbf{b}}_\mathrm{y}$
$\hat{\mathbf{a}}_\mathrm{y}$
B
x
$\dot{\theta}\,x\,\hat{\mathbf{b}}_\mathrm{y}$
$\hat{\mathbf{a}}_\mathrm{x}$
$\vec{\mathbf{r}}$
$\dot{x}\,\hat{\mathbf{b}}_\mathrm{x}$
θ
$\hat{\mathbf{b}}_\mathrm{x}$

6.6.1 Time-derivative of angular momentum with MotionGenesis

The following **M**otion**G**enesis commands calculate the time-derivative in reference frame N of the angular momentum of a spinning book (rigid body B) and creates an matrix with the $\hat{\mathbf{b}}_\mathrm{x}$, $\hat{\mathbf{b}}_\mathrm{y}$, $\hat{\mathbf{b}}_\mathrm{z}$ measures.

```
   (1) % File: DerivativeOfAngularMomentum.txt
   (2) %--------------------------------------
   (3) NewtonianFrame  N
   (4) RigidBody       B
   (5) Variable wx', wy', wz'
   (6) B.SetInertia( Bcm, Ixx, Iyy, Izz )
   (7) %-----------------------------------------------
   (8) B.SetAngularVelocity( N, wx*Bx> + wy*By> + wz*Bz> )
-> (9) w_B_N> = wx*Bx> + wy*By> + wz*Bz>

   (10) H> = Vector( B, Ixx*wx, Iyy*wy, Izz*wz )
-> (11) H> = Ixx*wx*Bx> + Iyy*wy*By> + Izz*wz*Bz>

   (12) %-----------------------------------------------
   (13) Zero> = Dt( H>, N )
-> (14) Zero> = (Izz*wy*wz+Ixx*wx'-Iyy*wy*wz)*Bx> + (Ixx*wx*wz+Iyy*wy'-Izz*wx*
        wz)*By> + (Iyy*wx*wy+Izz*wz'-Ixx*wx*wy)*Bz>

   (15) Zero = Matrix( B, Zero> )
-> (16) Zero[1] = Izz*wy*wz + Ixx*wx' - Iyy*wy*wz
-> (17) Zero[2] = Ixx*wx*wz + Iyy*wy' - Izz*wx*wz
-> (18) Zero[3] = Iyy*wx*wy + Izz*wz' - Ixx*wx*wy
```

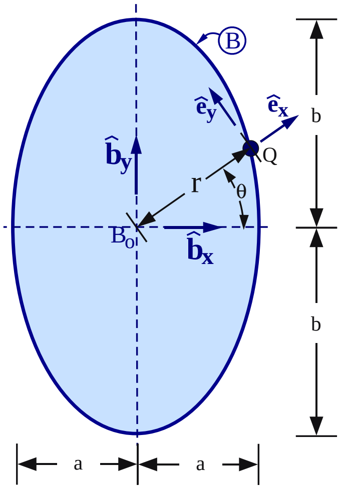




6.6.2 Differential geometry: Ellipse circumference, area, normal, tangent with MotionGenesis

The following figure shows a point Q on the periphery of an ellipse B whose center is point B_o.
Right-handed orthogonal unit vectors $\widehat{\mathbf{b}}_x$, $\widehat{\mathbf{b}}_y$, $\widehat{\mathbf{b}}_z$ are fixed in B with

- $\widehat{\mathbf{b}}_x$ horizontally-right and aligned with the ellipse's minor axis
- $\widehat{\mathbf{b}}_y$ vertically-upward and aligned with the ellipse's major axis
- $\widehat{\mathbf{b}}_z$ perpendicular to the ellipse.



Right-handed orthogonal unit vectors $\widehat{\mathbf{e}}_x, \widehat{\mathbf{e}}_y, \widehat{\mathbf{e}}_z$ are initially directed with $\widehat{\mathbf{e}}_i = \widehat{\mathbf{b}}_i$ $(i = x, y, z)$ and then are subjected to a right-handed rotation in B characterized by $\theta\,\widehat{\mathbf{b}}_z$ so $\widehat{\mathbf{e}}_x$ points from B_o to Q and $\widehat{\mathbf{e}}_z = \widehat{\mathbf{b}}_z$.

Description	Symbol	Type	Value
Half-diameter of ellipse minor axis	a	+Constant	2
Half-diameter of ellipse major axis	b	+Constant	4
Distance between N_o and Q	r	+Variable	Varies
Angle from $\widehat{\mathbf{b}}_x$ to $\widehat{\mathbf{e}}_x$ with $+\widehat{\mathbf{b}}_z$ sense	θ	Variable	Varies

When Q's position vector from B_o is expressed in terms of scalars x and y as shown below-left, the equation of the ellipse can be written as shown below-right.

$$\vec{\mathbf{r}} \triangleq \vec{\mathbf{r}}^{\,Q/B_o} = x\,\widehat{\mathbf{b}}_x + y\,\widehat{\mathbf{b}}_y \qquad\qquad \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

1. Denoting $\vec{\mathbf{r}}$ as Q's position vector from B_o, form $\left|{}^B d\vec{\mathbf{r}}\right|$ (the magnitude of the ***vector differential*** of $\vec{\mathbf{r}}$ in B). Next, write an integral (with appropriate limits) for the ellipse's circumference in terms of $d\theta$.[2] Then, express r and $\frac{dr}{d\theta}$ in terms of θ.
   Result:
   $$\left|{}^B d\vec{\mathbf{r}}\right| = \sqrt{(r\,d\theta)^2 + dr^2} \qquad \text{Circumference} = \int_{\theta=0}^{2\pi} \left|{}^B d\vec{\mathbf{r}}\right| = \int_{\theta=0}^{2\pi} \sqrt{r^2 + \left(\frac{dr}{d\theta}\right)^2}\, d\theta$$
   $$r = \frac{a\,b}{\sqrt{a^2\sin^2(\theta) + b^2\cos^2(\theta)}} \qquad \frac{dr}{d\theta} = \frac{-a\,b\,(a^2 - b^2)\sin(\theta)\cos(\theta)}{\sqrt{a^2\sin^2(\theta) + b^2\cos^2(\theta)}^{\,3}}$$

2. Find a formula for the circumference when $a = b$ is **constant** (the ellipse is a circle).
   Calculate the circumference of an ellipse (4^+ significant digits) when $a = 2$ m and $b = 4$ m.
   Express the area $d\Delta$ of the triangle formed by points B_o, $Q(\theta)$, and $Q(\theta + d\theta)$: first as a function of $\vec{\mathbf{r}}$ and $d\vec{\mathbf{r}}$; then as a function of r and θ.
   Next, calculate the area of an ellipse (4^+ significant digits) when $a = 2$ m and $b = 4$ m.[3]
   Result:
   $$\text{Circumference of circle} = 2\,\pi\,b \qquad \text{Circumference of ellipse} = 19.377 \text{ m}$$
   $$d\Delta = \frac{1}{2}\left|\vec{\mathbf{r}} \times d\vec{\mathbf{r}}\right| d\theta = \frac{1}{2} r^2\, d\theta = \frac{1}{2}\,\frac{a^2\,b^2}{a^2\sin^2(\theta) + b^2\cos^2(\theta)}\, d\theta \qquad \text{Area of ellipse} = 25.133 \text{ m}^2$$

3. There is an exact closed-form solution for the circumference of an ellipse. **True/$\boxed{\textbf{False}}$**.
   There is an exact closed-form solution for the area of an ellipse. **$\boxed{\textbf{True}}$/False**.

4. **Draw** an outward normal vector $\vec{\mathbf{n}}$ and tangent vector $\vec{\mathbf{t}}$ at point Q.
   Express $\vec{\mathbf{n}}$ and $\vec{\mathbf{t}}$ in terms of $\widehat{\mathbf{b}}_x$, $\widehat{\mathbf{b}}_y$, $\widehat{\mathbf{b}}_z$ when $x = \frac{a}{2}$.

Result:

Ellipse: $a = 2$ and $b = 4$	Circle: $a = 2$ and $b = 2$
$\vec{\mathbf{n}} = 1.0\,\widehat{\mathbf{b}}_x + 0.433\,\widehat{\mathbf{b}}_y$	$\vec{\mathbf{n}} = 0.5\,\widehat{\mathbf{b}}_x + 0.866\,\widehat{\mathbf{b}}_y$
$\vec{\mathbf{t}} = -0.433\,\widehat{\mathbf{b}}_x + 1.0\,\widehat{\mathbf{b}}_y$	$\vec{\mathbf{t}} = -0.866\,\widehat{\mathbf{b}}_x + 0.5\,\widehat{\mathbf{b}}_y$

---

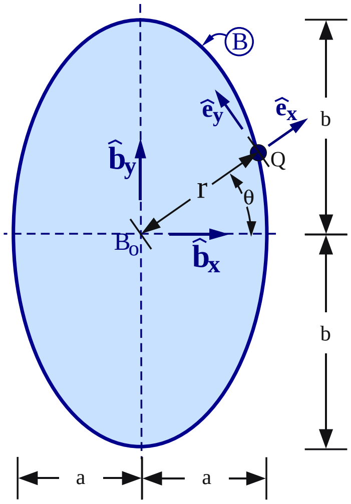
[2] The integral simplifies by noting that $d\theta$ is **positive** when the integral's upper-limit is larger than the integral's lower-limit.

[3] Verify your numerical integration result with the following formula for the area of ellipse: Area $= \pi\,a\,b$.

When $a = b$ (the ellipse is a circle), $\vec{\mathbf{n}}$ is always parallel to $\vec{\mathbf{r}}^{\,/Q/B_o}$ **[✓ True]/False**

When $a \neq b$ (the ellipse is not a circle), $\vec{\mathbf{n}}$ is always parallel to $\vec{\mathbf{r}}^{\,/Q/B_o}$ **True/[✓ False]**

5. **Optional******:** Show how the definition of an ***ellipse*** results in $F(x, y) \;=\; \dfrac{x^2}{a^2} \;+\; \dfrac{y^2}{b^2} \;-\; 1 \;=\; 0.$

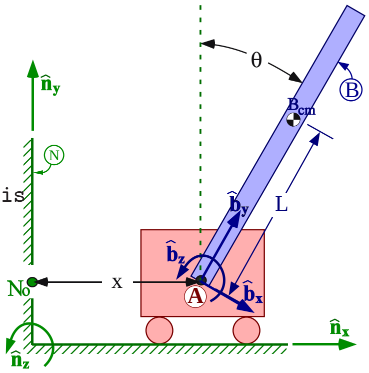


```
% File:  EllipseCircleNormalGradientCircumferenceArea.al
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%-----------------------------------------------------------------
RigidBody   B           % Ellipse Bx> and By> are directed along ellipse axes
Point       Q( B )      % Point on periphery of ellipse B
RigidFrame  E           % Ex> points from Bo to Q and Ez> = Bz>.
Variable    r'          % Distance between Bo and Q and dr/dtheta
Specified   x, y        % Locates Q's position from Bo
Constant    a+, b+      % Dimensions of ellipse's minor/major axes
IndependentVariable theta
%-----------------------------------------------------------------
%       Equation for an ellipse
EllipseCurve = (x/a)^2 + (y/b)^2 - 1
%--------------------------------------------------------------
%    The outward normal at Q is parallel to B's gradient at Q.
Gradient> = D( EllipseCurve, x )*Bx>  +  D( EllipseCurve, y )*By>
Tangent> = Cross( Bz>, Gradient> )
%--------------------------------------------------------------
%    Solve for positive value of y when x = a/2
CurveEvaluated = Evaluate( EllipseCurve, x = a/2 )
yAtX = SolveQuadraticPositiveRoot( CurveEvaluated, y )
%--------------------------------------------------------------
%    Gradient and tangent when x = a/2,  a = 2, and  b = 4
EllipseGradient> = Evaluate( Gradient>,  x = 4/2,  y = yAtX,  a = 2,  b = 4 )
EllipseTangent>  = Evaluate(  Tangent>,  x = 4/2,  y = yAtX,  a = 2,  b = 4 )
%--------------------------------------------------------------
%    Circle gradient and tangent when  x = a/2  and  a = b = 2
CircleGradient> = Evaluate( Gradient>,   x = 2/2,  y = yAtX,  a = 2,  b = 2 )
CircleTangent>  = Evaluate(  Tangent>,   x = 2/2,  y = yAtX,  a = 2,  b = 2 )
%-----------------------------------------------------------------
%    Rotation matrix relating Exyz> to Bxyz>
E.SetRotationMatrixZ( B, theta )
r> = r*Ex>              % Q's position from point Bo
x = Dot( r>, Bx> )
y = Dot( r>, By> )
%-----------------------------------------------------------------
%       Solve ellipse equation for positive r in terms of theta.
positiveRoot = GetQuadraticPositiveRoot( EllipseCurve, r )
SetDt( r = a*b / sqrt(a^2*sin(theta)^2 + b^2*cos(theta)^2) )
%-----------------------------------------------------------------
%       Calculate derivative of r with respect to theta in B - and its magnitude
DpDTheta> = Dt( r>, B )
magDpDTheta = GetMagnitude( DpDTheta> )
%-----------------------------------------------------------------
%       Calculate circumference of circle and ellipse
magDpDThetaForCircle = Evaluate( magDpDtheta,  a = b )
CircumferenceOfCircle = magDpDThetaForCircle * Integrate( 1, theta=0 : 2*pi )
magDpDThetaForEllipse = Evaluate( magDpDTheta, a=2, b=4 )
CircumferenceOfEllipse = Integrate( magDpDThetaForEllipse, theta = 0 : 2*pi )
%-----------------------------------------------------------------
%       Differential area of ellipse (uses area of triangle)
dAreaDTheta> = 1/2 * Cross( r>,  r> + DpDtheta> )
dAreaDTheta = Dot( dAreaDTheta>, Bz> )
%-----------------------------------------------------------------
%       Calculate area of ellipse
dAreaDThetaForEllipse = Evaluate( dAreaDTheta, a=2, b=4 )
AreaOfEllipse = Integrate( dAreaDThetaForEllipse, theta = 0 : 2*pi )
ShouldApproximateZero = AreaOfEllipse - Evaluate( pi*a*b,  a = 2, b = 4 )
%--------------------------------------------------------------
Save EllipseCircleNormalGradientCircumferenceArea.all
If( abs(ShouldApproximateZero) < 1.0E-7 ) {Quit}
```

6.7 MotionGenesis translation commands (position, velocity, and acceleration)

Command	Description and associated formula
`Q.Translate( P, posVector )`	Sets Q's position from P. Sets Q's velocity and acceleration.
`Q.Translate( P, posVector, B )`	Sets Q's position from P. Sets Q's velocity and acceleration. (Both P and Q are fixed on reference frame B).
`Q.GetPosition( No )`	Gets Q's position vector from N_o, i.e., $\vec{\mathbf{r}}^{\,Q/N_o}$.
`Q.SetPosition( P, posVector )`	Sets Q's position vector from P, i.e., $\vec{\mathbf{r}}^{\,Q/P} = \mathbf{posVector}$
`Q.GetDistance( P )`	Gets Q's distance from P, i.e., $\lvert\vec{\mathbf{r}}^{\,Q/P}\rvert$.
`Q.GetElongation( P )`	Gets Q's elongation from P, i.e., the time-derivative of $\lvert\vec{\mathbf{r}}^{\,Q/P}\rvert$
`Q.GetSpeed( N )`	Gets Q's speed in N, i.e., $\lvert{}^{N}\vec{\mathbf{v}}^{Q}\rvert$
`Q.GetVelocity( N )`	Gets Q's velocity in N, i.e., ${}^{N}\vec{\mathbf{v}}^{Q}$
`Q.SetVelocity( N, velVector )`	Sets Q's velocity in N, i.e., ${}^{N}\vec{\mathbf{v}}^{Q} = \mathbf{velVector}$
`Q.SetVelocity( N, P )`	Sets Q's velocity in N via ${}^{N}\vec{\mathbf{v}}^{Q} = {}^{N}\vec{\mathbf{v}}^{P} + \dfrac{{}^{N}d\,\vec{\mathbf{r}}^{\,Q/P}}{dt}$
`Q.SetVelocity( N, Bo, B )`	Velocity of two points (Q and B_o) ***fixed*** on rigid object B. ${}^{N}\vec{\mathbf{v}}^{Q} = {}^{N}\vec{\mathbf{v}}^{B_o} + {}^{N}\vec{\boldsymbol{\omega}}^{B} \times \vec{\mathbf{r}}^{\,Q/B_o}$
`Q.GetAcceleration( N )`	Gets Q's acceleration in N, i.e., ${}^{N}\vec{\mathbf{a}}^{Q}$
`Q.SetAcceleration( N, accelVector)`	Sets Q's acceleration in N, i.e., ${}^{N}\vec{\mathbf{a}}^{Q} = \mathbf{accelVector}$
`Q.SetAcceleration( N, P )`	Sets Q's acceleration in N via ${}^{N}\vec{\mathbf{a}}^{Q} = {}^{N}\vec{\mathbf{a}}^{P} + \dfrac{{}^{N}d^2\,\vec{\mathbf{r}}^{\,Q/P}}{dt^2}$
`Q.SetAcceleration( N, Bo, B )`	Acceleration of two points (B_o and Q) ***fixed*** on rigid object B. ${}^{N}\vec{\mathbf{a}}^{Q} = {}^{N}\vec{\mathbf{a}}^{B_o} + {}^{N}\vec{\boldsymbol{\alpha}}^{B} \times \vec{\mathbf{r}}^{\,Q/B_o} + {}^{N}\vec{\boldsymbol{\omega}}^{B} \times \left({}^{N}\vec{\boldsymbol{\omega}}^{B} \times \vec{\mathbf{r}}^{\,Q/B_o}\right)$
`Q.SetVelocityAcceleration( N, velVector)`	Sets ${}^{N}\vec{\mathbf{v}}^{Q} = \mathbf{velVector}$ and ${}^{N}\vec{\mathbf{a}}^{Q} = \dfrac{{}^{N}d\,\mathbf{velVector}}{dt}$
`Q.SetVelocityAcceleration( N, P, ...  )`	Sets Q's velocity and acceleration (see previous related syntax).
Note: N and B are reference frames (or rigid bodies) whereas Q, N_o, B_o are points (or particles).	

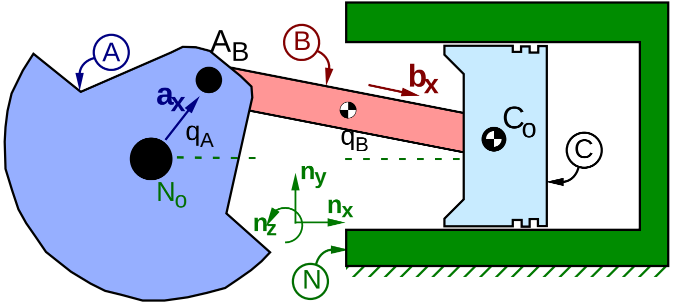
MotionGenesis translational kinematics (and dynamics) for an inverted pendulum on a cart


θ
$\hat{\mathbf{n}}_\mathbf{y}$
B
B_{cm}
N
$\hat{\mathbf{b}}_\mathbf{y}$
L
$\hat{\mathbf{b}}_\mathbf{z}$
N_o
x
A
$\hat{\mathbf{b}}_\mathbf{x}$
$\hat{\mathbf{n}}_\mathbf{x}$
$\hat{\mathbf{n}}_\mathbf{z}$

```
   (1) %  File: InvertedPendulumOnCartDynamics.txt
   (2) %--------------------------------------------------------------------
   (3) NewtonianFrame  N
   (4) Particle        A              % Cart
   (5) RigidBody       B              % Inverted pendulum
   (6) %--------------------------------------------------------------------
   (7) Variable   x’’                 % Distance between No to A
   (8) Variable   theta’’             % Angle from local vertical to B’s long axis
   (9) Specified  Fc                  % Control force on cart
   (10) Constant   g+ = 9.81 m/s^2     % Gravitational constant
   (11) Constant   L+ = 0.5  m         % Distance between A and Bcm
   (12) A.SetMass( mA = 10 kg )
   (13) B.SetMassInertia( mB = 1 kg,  Izz = 1/12*mB*(2*L)^2,  0,  Izz )
-> (14) Izz = 0.3333333*mB*L^2

   (15) %--------------------------------------------------------------------
   (16) %       Rotational and translational kinematics
   (17) B.RotateNegativeZ( N, theta )
-> (18) B_N = [cos(theta), -sin(theta), 0; sin(theta), cos(theta), 0; 0, 0, 1]
-> (19) w_B_N> = -theta’*Bz>
-> (20) alf_B_N> = -theta’’*Bz>

   (21) A.Translate( No, x*Nx> )
-> (22) p_No_A> = x*Nx>
-> (23) v_A_N> = x’*Nx>
-> (24) a_A_N> = x’’*Nx>

   (25) Bcm.Translate( A, L*By> )
-> (26) p_A_Bcm> = L*By>
-> (27) v_Bcm_N> = L*theta’*Bx> + x’*Nx>
-> (28) a_Bcm_N> = L*theta’’*Bx> - L*theta’^2*By> + x’’*Nx>

   (29) %--------------------------------------------------------------------
   (30) %       Relevant contact and distance forces
   (31) System.AddForceGravity( -g*Ny> )
```

```
-> (32) Force_A> = -g*mA*Ny>
-> (33) Force_Bcm> = -g*mB*Ny>

   (34) A.AddForce( Fc*Nx> )
-> (35) Force_A> = Fc*Nx> - g*mA*Ny>

   (36) %---------------------------------------------------------------
   (37) %       Form and simplify equations of motion (via Newton/Euler)
   (38) EquationsOfMotion[1] = Dot(  Nx>,  A.GetDynamics() + B.GetDynamics() )
-> (39) EquationsOfMotion[1] = mA*x'' + mB*x'' + L*mB*cos(theta)*theta'' - Fc
        - L*mB*sin(theta)*theta'^2

   (40) EquationsOfMotion[2] = Dot( -Bz>,  B.GetDynamics(A) )
-> (41) EquationsOfMotion[2] = Izz*theta'' + mB*L^2*theta'' - L*mB*(g*sin(theta
       )-cos(theta)*x'')

   (42) FactorLinear( EquationsOfMotion,  theta'', x'', Fc, g )
-> (43) EquationsOfMotion[1] = (mA+mB)*x'' + L*mB*cos(theta)*theta'' - Fc - L*
       mB*sin(theta)*theta'^2
-> (44) EquationsOfMotion[2] = L*mB*cos(theta)*x'' + (mB*L^2+Izz)*theta'' - g*L
       *mB*sin(theta)

   (45) %---------------------------------------------------------------
```

6.8 Configuration constraints and straight slots with MotionGenesis

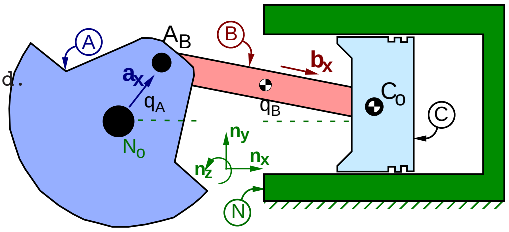


The figure to the right shows a piston C sliding in a cylinder that is fixed in a reference frame N. The piston is connected to a connecting rod B by a revolute joint at C_o. The connecting rod is connected to a crankshaft A by a second revolute joint at A_B. The center of the crankshaft is connected to N by a third revolute joint at N_o.

The point of this problem is to form a configuration constraint of the form $f(q_A, q_B) = 0$. After noticing that the configuration constraint is ***nonlinear*** in q_A and q_B, it is helpful to differentiate the configuration constraint to form a motion constraint $\dot{f}(\dot{q}_A, \dot{q}_B, q_A, q_B) = 0$ which is ***linear*** in $\dot{q}_A$ and $\dot{q}_B$.

Solving nonlinear configuration constraints with MotionGenesis

```
   (1) %    File:  PistonEngineConstraintKinematics.txt
   (2) % Problem:  Kinematic analysis of a piston engine
   (3) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (4) %--------------------------------------------------
   (5) NewtonianFrame  N
   (6) RigidBody  A                 % Crankshaft
   (7) RigidBody  B                 % Connecting rod
   (8) RigidBody  C                 % Piston
   (9) Point      AB                % Point connecting A and B
   (10) %--------------------------------------------------
   (11) Variable   qA', qB'          % qA and qB are angles
   (12) Constant   LA = 10 cm        % Length between No and AB
   (13) Constant   LB = 20 cm        % Length between AB and Ccm
   (14) Constant   kp =  0 noUnits   % For constraint stabilization
   (15) %--------------------------------------------------
   (16) %       Rotational kinematics
   (17) A.RotatePositiveZ(  N,  qA  )
-> (18) A_N = [cos(qA), sin(qA), 0; -sin(qA), cos(qA), 0; 0, 0, 1]
-> (19) w_A_N> = qA'*Az>

   (20) B.RotateNegativeZ(  N,  qB  )
-> (21) B_N = [cos(qB), -sin(qB), 0; sin(qB), cos(qB), 0; 0, 0, 1]
-> (22) w_B_N> = -qB'*Bz>
```

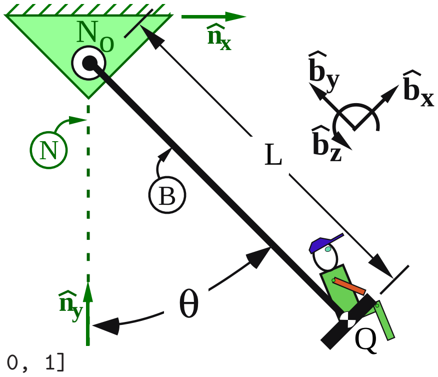


```
   (23) %--------------------------------------------------
   (24) %       Translational kinematics
   (25) AB.SetPositionVelocity(  No,  LA*Ax>  )
-> (26) p_No_AB> = LA*Ax>
-> (27) v_AB_N> = LA*qA'*Ay>

   (28) Co.SetPositionVelocity(  AB,  LB*Bx>  )
-> (29) p_AB_Co> = LB*Bx>
-> (30) v_Co_N> = LA*qA'*Ay> - LB*qB'*By>

   (31) %--------------------------------------------------
   (32) %       Position constraint f(qA,qB) = 0
   (33) f = Dot( Co.GetPosition(No), Ny> )
-> (34) f = LA*sin(qA) - LB*sin(qB)

   (35) %-----------------------------------------------------------------
   (36) %       Input constants, variables, etc.
   (37) Input   qA = 45 deg
   (38) %-----------------------------------------------------------------
   (39) %       Find initial value of qB for given input values (guess qB=0)
   (40) SolveSetInput(  f,  qB = 0 deg )

->   %  INPUT has been assigned as follows:
->   %   qB                       20.70481105463543       deg

   (41) %-----------------------------------------------------------------
   (42) %       Velocity constraint with constraint stabilization: Dt(f) + kp*f
   (43) velocityConstraint = Dot( Co.GetVelocity(N), Ny> ) + kp*f
-> (44) velocityConstraint = kp*f + LA*cos(qA)*qA' - LB*cos(qB)*qB'

   (45) Solve( velocityConstraint, qB' )
-> (46) qB' = (kp*f+LA*cos(qA)*qA')/(LB*cos(qB))

   (47) %-----------------------------------------------------------------
   (48) %       Simulate compressor with constant speed motor of 10 rad/sec
   (49) qA' = 10
-> (50) qA' = 10

   (51) %-----------------------------------------------------------------
   (52) %       Input constants, variables, etc. for ODE command (for motion)
   (53) Input  tFinal = 6 sec,  tStep = 0.02 sec,  absError = 1.0E-7
   (54) %-----------------------------------------------------------------
   (55) %       List output quantities and solve ODEs (or write MATLAB, C, ... code).
   (56) Output  t sec,  qA deg,  qB deg,  f cm
   (57) ODE()  PistonEngineConstraintKinematics
```

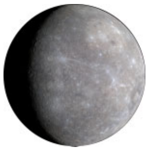
6.9 MotionGenesis commands for a particle

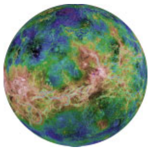
`Particle Q`	Declares Q as a particle
`Q.SetMass( mQ )`	Declares `mQ` as a non-negative constant (if `mQ` is not already defined) Assigns `mQ` to the mass of particle `Q`
`Q.GetMass()`	Returns `Q`'s mass
`Q.GetLinearMomentum()`	Returns `Q`'s linear momentum in the Newtonian reference frame
`Q.GetAngularMomentum(P)`	Returns `Q`'s angular momentum about point P in the Newtonian reference frame
`Q.GetKineticEnergy()`	Returns `Q`'s kinetic energy in the Newtonian reference frame
`Q.GetInertiaDyadic(P)`	Returns `Q`'s inertia dyadic about point `P`

Calculating a particle's momentum, kinetic energy, etc., with MotionGenesis



```
   (1) % File: InfantOnSwing.txt
   (2) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (3) %------------------------------------------------
   (4) NewtonianFrame N      % Newtonian reference frame
   (5) RigidFrame     B      % Rod connecting No to Q
   (6) Particle       Q      % Infant on swing
   (7) %------------------------------------------------
   (8) Variable  theta'
   (9) Constant  L
   (10) Q.SetMass( m )
   (11) %------------------------------------------------
   (12) %       Rotational and translational kinematics
   (13) B.RotateZ(  N,  theta  )
-> (14) B_N = [cos(theta), sin(theta), 0; -sin(theta), cos(theta), 0; 0, 0, 1]
-> (15) w_B_N> = theta'*Bz>

   (16) Q.SetPositionVelocity(  No,  -L*By>  )
-> (17) p_No_Q> = -L*By>
-> (18) v_Q_N> = L*theta'*Bx>

   (19) %------------------------------------------------
   (20) %       Q's linear momentum in N
   (21) LinearMomentum> = Q.GetLinearMomentum()
-> (22) LinearMomentum> = L*m*theta'*Bx>

   (23) %------------------------------------------------
   (24) %       Q's angular momentum about No in N
   (25) AngularMomentum> = Q.GetAngularMomentum( No )
-> (26) AngularMomentum> = m*L^2*theta'*Bz>

   (27) %------------------------------------------------
   (28) %       Q's kinetic energy in N
   (29) KineticEnergy = Q.GetKineticEnergy()
-> (30) KineticEnergy = 0.5*m*L^2*theta'^2
```

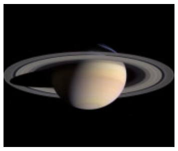

Mercury

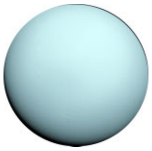

Venus

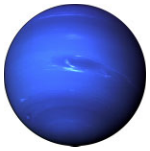
Earth

Mars

Jupiter


Saturn

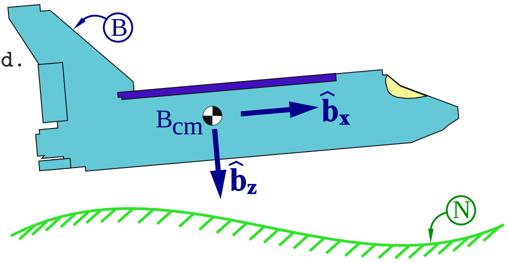

Uranus


Neptune

6.10 MotionGenesis commands for a rigid body

`RigidBody B`	Declares B as a rigid body
`B.SetMass( mB )`	Declares `mB` as a non-negative constant (if `mB` is not already defined) Assigns `mB` to the mass of `B`
`B.SetInertia(Bcm, Ix, Iy, Iz, Ixy, Iyz, Izx)`	Declares rigid body B's moments/products of inertia about B_{cm}
`B.GetMass()`	Returns `B`'s mass
`B.GetInertiaDyadic( P )`	Returns `B`'s inertia dyadic about point `P`
`B.GetLinearMomentum()`	Returns `B`'s linear momentum in the Newtonian reference frame
`B.GetAngularMomentum( P )`	Returns `B`'s angular momentum about point P in the Newtonian reference frame
`B.GetKineticEnergy()`	Returns `B`'s kinetic energy in the Newtonian reference frame

Calculating rigid-body's momentum, kinetic energy, etc., with MotionGenesis



```
   (1) % File:  AirplaneExample.txt
   (2) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (3) %------------------------------------------------------------
   (4) NewtonianFrame  N
   (5) RigidBody       B
   (6) B.SetMassInertia(  m,  Ixx, Iyy, Izz )
   (7) %------------------------------------------------------------
   (8) Variable   wx', wy', wz'
   (9) Variable   vx', vy', vz'
   (10) %------------------------------------------------------------
   (11) %      Set B's angular velocity and angular acceleration in N
   (12) B.SetAngularVelocityAcceleration( N, wx*Bx> + wy*By> + wz*Bz> )
-> (13) w_B_N> = wx*Bx> + wy*By> + wz*Bz>
-> (14) alf_B_N> = wx'*Bx> + wy'*By> + wz'*Bz>

   (15) %------------------------------------------------------------
   (16) %      Set Bcm's velocity and acceleration in N
   (17) Bcm.SetVelocityAcceleration( N,  vx*Bx> + vy*By> + vz*Bz> )
-> (18) v_Bcm_N> = vx*Bx> + vy*By> + vz*Bz>
-> (19) a_Bcm_N> = (vz*wy+vx'-vy*wz)*Bx> + (vx*wz+vy'-vz*wx)*By> + (vy*wx+vz'-
        vx*wy)*Bz>

   (20) %------------------------------------------------------------
   (21) %      Calculate B's linear momentum in N
   (22) LinearMomentum> = B.GetLinearMomentum()
-> (23) LinearMomentum> = m*vx*Bx> + m*vy*By> + m*vz*Bz>

   (24) %------------------------------------------------------------
   (25) %      Calculate B's angular momentum about Bcm in N
   (26) AngularMomentum> = B.GetAngularMomentum( Bcm )
-> (27) AngularMomentum> = Ixx*wx*Bx> + Iyy*wy*By> + Izz*wz*Bz>

   (28) %------------------------------------------------------------
   (29) %      Calculate B's kinetic energy in N
   (30) KineticEnergy = B.GetKineticEnergy()
-> (31) KineticEnergy = 0.5*Ixx*wx^2 + 0.5*Iyy*wy^2 + 0.5*Izz*wz^2 + 0.5*m*(vx^
        2+vy^2+vz^2)
```

Courtesy USAF

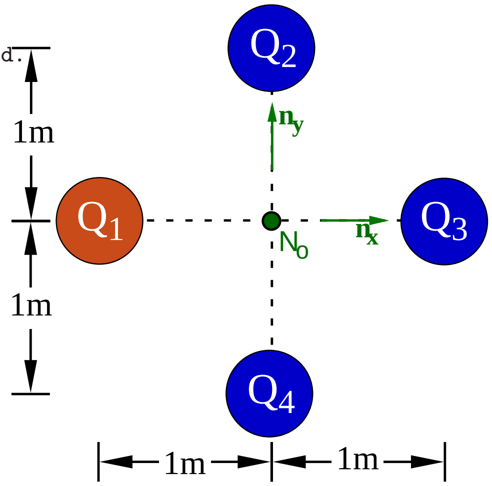
Chapter 7

Computing mass and inertia properties

7.1 MotionGenesis mass and center of mass commands

`Q.SetMass( mQ )`	Declares Q's mass as the non-negative constant `mQ` (if `mQ` is not already defined)
`B.SetMass( mB = 3 )`	Declares B's mass as `mB` which is assigned to `3`
`Q.GetMass() + B.GetMass()`	Returns the sum of `Q`'s mass and `B`'s mass
`System.GetMass()`	Returns the system's mass
`System.GetCMPosition( P )`	Returns the position vector of the system mass center from point `P`
`System.GetCMVelocity( N )`	Returns the velocity of the system's mass center in reference frame `N`
`System.GetCMAcceleration( N )`	Returns the velocity of the system's mass center in reference frame `N`

Example: Calculating mass and center of mass of four particles with MotionGenesis



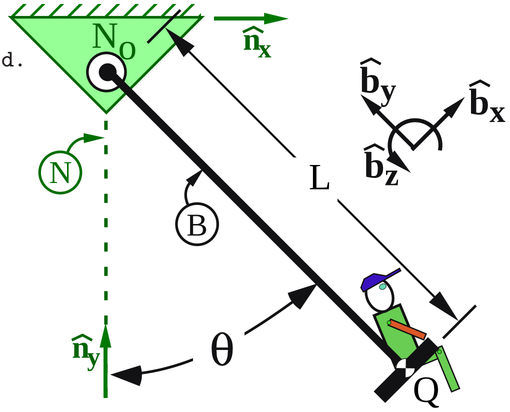
```
   (1) % File: CenterOfMassOfFourParticles.txt
   (2) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (3) %--------------------------------------
   (4) RigidFrame  N
   (5) Particle    Q1, Q2, Q3, Q4
   (6) %--------------------------------------
   (7) Q1.SetMass( 1 )
   (8) Q2.SetMass( 2 )
   (9) Q3.SetMass( 2 )
   (10) Q4.SetMass( 2 )
   (11) %--------------------------------------
   (12) Q1.SetPosition( No, -Nx> )
-> (13) p_No_Q1> = -Nx>

   (14) Q2.SetPosition( No,  Ny> )
-> (15) p_No_Q2> = Ny>

   (16) Q3.SetPosition( No,  Nx> )
-> (17) p_No_Q3> = Nx>

   (18) Q4.SetPosition( No, -Ny> )
-> (19) p_No_Q4> = -Ny>

   (20) %--------------------------------------
   (21) %   Mass, center of mass, and centroid
   (22) MassOfSystem = System.GetMass()
-> (23) MassOfSystem = 7

   (24) CMPositionFromNo> = System.GetCMPosition( No )
-> (25) CMPositionFromNo> = 0.1428571*Nx>

   (26) CentroidPositionFromNo> = 1/4*( Q1.GetPosition(No) + Q2.GetPosition(No) &
                               + Q3.GetPosition(No) + Q4.GetPosition(No) )
-> (27) CentroidPositionFromNo> = 0>
```

7.2 MotionGenesis inertia commands

`B.SetInertia( Bp, Ix,Iy,Iz, Ixy,Iyz,Izx)`	Declares rigid body *B*'s moments/products of inertia about *Bp*
`B.SetMassInertia( m, Ix,Iy,Iz, Ixy,Iyz,Izx)`	Same as: `B.SetMass(m); B.SetInertia( Bcm, Ix,Iy,Iz, Ixy,Iyz,Izx)`

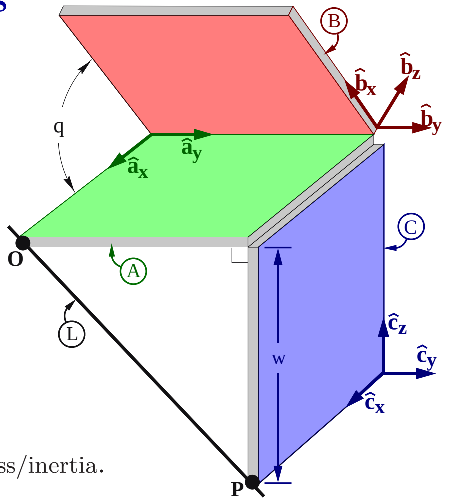
`B.GetInertiaDyadic( Bcm )`	Returns *B*'s inertia dyadics about point B_{cm}
`System.GetInertiaDyadic( P )`	Returns the system's inertia dyadic about point `P`
`System.GetInertiaDyadic( P, N )`	Returns the system's inertia dyadic about point `P` expressed in $\widehat{\mathbf{n}}_{\text{x}}$, $\widehat{\mathbf{n}}_{\text{y}}$, $\widehat{\mathbf{n}}_{\text{z}}$
`System.GetInertiaMatrix( P, N )`	Returns the system's inertia matrix about point `P` for $\widehat{\mathbf{n}}_{\text{x}}$, $\widehat{\mathbf{n}}_{\text{y}}$, $\widehat{\mathbf{n}}_{\text{z}}$
`System.GetMomentOfInertia( P, Nx> )`	Returns the system's moment of inertia about point `P` for $\widehat{\mathbf{n}}_{\text{x}}$
`System.GetRadiusOfGyration( P, Nx> )`	Returns the system's radius of gyration about point `P` for $\widehat{\mathbf{n}}_{\text{x}}$
`System.GetProductOfInertia( P, Nx>, Ny> )`	Returns the system's product of inertia about point `P` for $\widehat{\mathbf{n}}_{\text{x}}$ and $\widehat{\mathbf{n}}_{\text{y}}$

7.2.1 Example: Inertia properties for an infant on a swing with MotionGenesis



```
   (1) % File:  InertiaPropertiesOfInfantOnSwing.txt
   (2) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (3) %--------------------------------------------------------
   (4) RigidFrame  N
   (5) RigidBody   B
   (6) Particle    Q
   (7) Constant    L
   (8) Variable    theta
   (9) Q.SetMass( m )
   (10) %--------------------------------------------------------
   (11) %       Q's position vector from No
   (12) Q.SetPosition( No, -L*By> )
-> (13) p_No_Q> = -L*By>

   (14) %--------------------------------------------------------
   (15) %       Q's inertia dyadic about No expressed in B
   (16) QInertiaDyadicAboutNo>> = Q.GetInertiaDyadic( No, B )
-> (17) QInertiaDyadicAboutNo>> = m*L^2*Bx>*Bx> + m*L^2*Bz>*Bz>

   (18) %-----------------------------------------------------------------
   (19) %       B's rotation matrix in N
   (20) B.RotateZ( N, theta )
-> (21) B_N = [cos(theta), sin(theta), 0; -sin(theta), cos(theta), 0; 0, 0, 1]

   (22) %-----------------------------------------------------------------
   (23) %       Q's moments of inertia about No for various directions
   (24) IBxBx = Q.GetMomentOfInertia( No, Bx> )
-> (25) IBxBx = m*L^2

   (26) IByBy = Q.GetMomentOfInertia( No, By> )
-> (27) IByBy = 0

   (28) IBzBz = Q.GetMomentOfInertia( No, Bz> )
-> (29) IBzBz = m*L^2

   (30) INxNx = Q.GetMomentOfInertia( No, Nx> )
-> (31) INxNx = m*L^2*cos(theta)^2

   (32) %-----------------------------------------------------------------
   (33) %       Q's products of inertia about No for various directions
   (34) IBxBy = Q.GetProductOfInertia( No, Bx>, By> )
-> (35) IBxBy = 0

   (36) INxNy = Q.GetProductOfInertia( No, Nx>, Ny> )
-> (37) INxNy = m*L^2*sin(theta)*cos(theta)
```

7.3 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 $\hat{\mathbf{a}}_i$, $\hat{\mathbf{b}}_i$, and $\hat{\mathbf{c}}_i$ $(i = \text{x}, \text{y}, \text{z})$ are fixed in A, B, and C, respectively, with $\hat{\mathbf{a}}_y = \hat{\mathbf{b}}_y = \hat{\mathbf{c}}_y$ 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 B's orientation in A.

Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Plate mass/inertia.

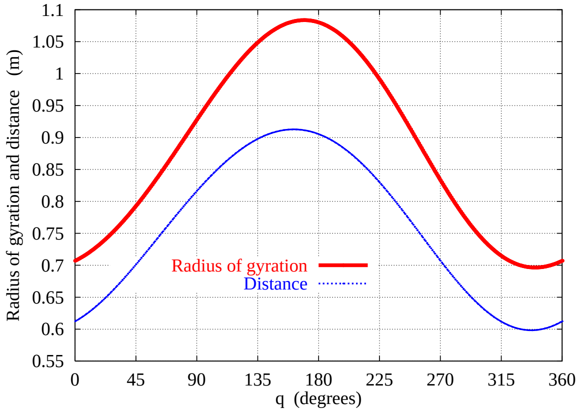
1. Find the distance between line $\overline{OP}$ and the center of mass S_{cm} of the system formed by A, B, C.
   Result: $$\text{Distance} \;=\; 0.1178511\sqrt{42 \;+\; \cos^2(q) \;+\; 6\sin(q) \;-\; 16\cos(q)}$$

2. For $q = 90°$, find λ_i (i=1, 2, 3), the system's principal moments of inertia about S_{cm}. Next, find the angle between $\hat{\mathbf{a}}_y$ and the principal axis associated with this system's **minimum** moment of inertia.
   Result: $\lambda_1 = 5\ \text{kg}\,\text{m}^2$ $\quad\lambda_2 = 13\ \text{kg}\,\text{m}^2$ $\quad\lambda_3 = 14\ \text{kg}\,\text{m}^2$ $\quad$ Angle $= 65.90516°$

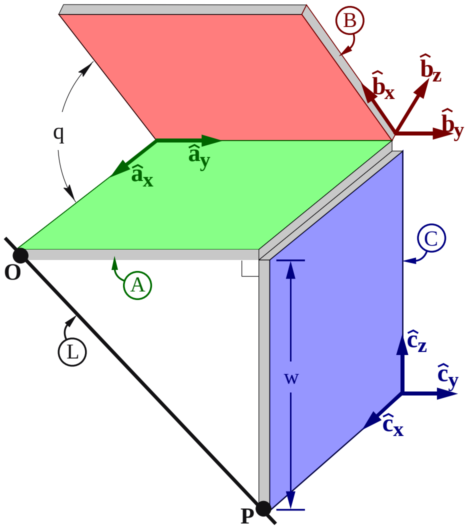
3. The system's radius of gyration about line $\overline{OP}$ is a measure of the how far the mass distribution is from line $\overline{OP}$ and is defined as $\sqrt{I/m}$ where I is the system's moment of inertia about $\overline{OP}$ and m is the system's mass. Using **physical intuition**, estimate the values of q that produce the smallest and largest radii of gyration about line $\overline{OP}$ and provide a reason for choosing these values.
   Result: $q_{\text{small}} \approx 340°$ $\qquad q_{\text{large}} \approx 160°$ $\qquad(\,0 \le q \le 360°\,)$
   Reason: These values minimize or maximize the distance from line OP to B's mass center.

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.

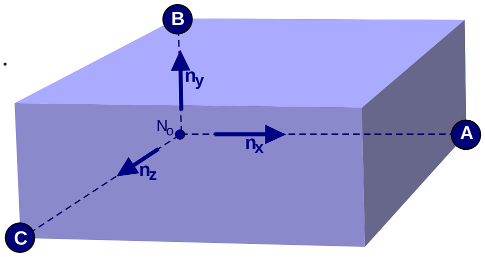
	Minimum Value	Minimum q	Maximum Value	Maximum q
Distance	0.598 m	337°	0.913 m	161°
Gyration	0.696 m	340°	1.084 m	169°



```
% File: ThreePlatesMassInertia.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%----------------------------------------------------------
%       Physical declarations
RigidBody  A, B, C           % Right, middle, top plate
Point      O, P              % Points on L
Point      SystemCM          % System's center of mass
%----------------------------------------------------------
%       Mathematical declarations
Variable   q                 % Angle between plates A and B
Constant   w = 1 m           % Width of plate
A.SetMassInertia( m = 12 kg,  I = m*w^2/12,  I, 2*I )
B.SetMassInertia( m,          I,             I, 2*I )
C.SetMassInertia( m,          I,           2*I,   I )
%----------------------------------------------------------
%       Geometry relating unit vectors
B.RotateNegativeY( A, q )
C.SetRotationMatrix( A, IdentityMatrix(3) )
%----------------------------------------------------------
%       Position vectors
Acm.SetPosition( O, -0.5*w*Ax> + 0.5*w*Ay> )
Bcm.SetPosition( O, -w*Ax> + 0.5*w*Ay> + 0.5*w*Bx> )
Ccm.SetPosition( O,  w*Ay> - 0.5*w*Cx> - 0.5*w*Cz> )
P.SetPosition( O, w*Ay> - w*Cz> )
%----------------------------------------------------------
%       System's center of mass position from point O.
P_O_SystemCM> = System.GetCMPosition( O )
uL> = P.GetUnitVector( O )     % Unit vector parallel to line L
DistanceFromLineLToSystemCM = GetMagnitude( Cross(uL>, SystemCM.GetPosition(O) ) )
%----------------------------------------------------------
%       System's inertia matrix about its center of mass.
SetAutoEpsilon( 1.0E-14 )      % Round to integers
SystemInertiaMatrix = EvaluateAtInput( System.GetInertiaMatrix( SystemCM, A ) )
%----------------------------------------------------------
%       Extract 1st eigenvector and calculate angle between Ay>
Lamba = GetEigen( Evaluate( SystemInertiaMatrix, q = 90 deg), EigenVecs )
EigenColumnMatrix1 = GetColumn( EigenVecs, 1 )
EigenVec1> = Vector( A,  EigenColumnMatrix1 )
AngleBetweenEigenVec1AndAy = GetAngleBetweenUnitVectorsDegrees( EigenVec1>, Ay> )
%----------------------------------------------------------
%       System's moment of inertia and radius of gyration of system about line OP.
MomentOfInertiaAboutLineL = System.GetMomentOfInertia(  O, uL> )
RadiusOfGyrationAboutL    = System.GetRadiusOfGyration( O, uL> )
%----------------------------------------------------------
%       Create Output and write MATLAB (or C or Fortran) program.
Output  q degs,  DistanceFromLineLToSystemCM m,  RadiusOfGyrationAboutL m
CODE Algebraic()  [ q deg = 0, 360, 1 ]  ThreePlatesMassInertia.m
%----------------------------------------------------------
Save ThreePlatesMassInertia.all
Quit
```



7.3.1 Example: Mass properties of three particles with MotionGenesis



```
   (1) % File: ThreeParticlesOnParellelpidedMassProperties.al
   (2) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (3) %-------------------------------------------------------
   (4) RigidFrame  N
   (5) Particle    A, B, C
   (6) Point       CM
   (7) A.SetMass( 1 )
   (8) B.SetMass( 1 )
   (9) C.SetMass( 1 )
   (10) %-------------------------------------------------------
   (11) %       Position vectors
   (12) A.SetPosition( No, 2*Nx> )
-> (13) p_No_A> = 2*Nx>

   (14) B.SetPosition( No, 2*Ny> )
-> (15) p_No_B> = 2*Ny>

   (16) C.SetPosition( No,   Nz> )
-> (17) p_No_C> = Nz>

   (18) %-------------------------------------------------------
   (19) %       Calculate and set CM's position from point No
   (20) CM.SetPosition( No, System.GetCMPosition(No) )
-> (21) p_No_CM> = 0.6666667*Nx> + 0.6666667*Ny> + 0.3333333*Nz>

   (22) %-------------------------------------------------------
   (23) %       System's inertia dyadic about No (expressed in N basis)
   (24) InertiaDyadicAboutNo>> = System.GetInertiaDyadic( No, N )
-> (25) InertiaDyadicAboutNo>> = 5*Nx>*Nx> + 5*Ny>*Ny> + 8*Nz>*Nz>

   (26) %-------------------------------------------------------
   (27) %       System moment of inertia and radius of gyration about diagonal of parallelpiped
   (28) Diagonal> = GetUnitVector( 2*Nx> + 2*Ny> + Nz> )
-> (29) Diagonal> = 0.6666667*Nx> + 0.6666667*Ny> + 0.3333333*Nz>

   (30) MomentOfInertiaAboutDiagonal = System.GetMomentOfInertia( No, Diagonal> )
-> (31) MomentOfInertiaAboutDiagonal = 5.333333

   (32) RadiusOfGyration = Sqrt( MomentOfInertiaAboutDiagonal / System.GetMass() )
-> (33) RadiusOfGyration = 1.333333

   (34) %-------------------------------------------------------
   (35) %       System's inertia dyadic about CM  expressed in N basis
   (36) InertiaDyadicAboutCM>> = System.GetInertiaDyadic( CM, N )
-> (37) InertiaDyadicAboutCM>> = 3.333333*Nx>*Nx> + 1.333333*Nx>*Ny> + 0.66666
        67*Nx>*Nz> + 1.333333*Ny>*Nx> + 3.333333*Ny>*Ny> + 0.6666667*Ny>*Nz> +
        0.6666667*Nz>*Nx> + 0.6666667*Nz>*Ny> + 5.333333*Nz>*Nz>
```

Chapter 8

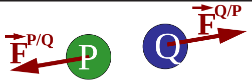
Forces, torques, moments, and statics

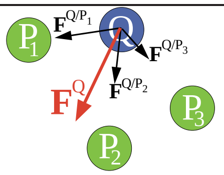
8.1 MotionGenesis commands and syntax for force, torque, and moments

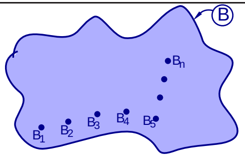
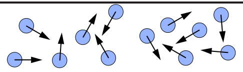
Motion**G**enesis command	Description and associated formula
`Q.AddForce( forceVector )`	Adds `forceVector` to the force on point Q.
`Q.AddForce( P, forceVector )`	Adds `forceVector` to the force on point Q from point P.
`S.AddForceGravity( gravityVector )`	Adds a uniform gravitational force to the particle, body, or system S.
`Q.AddForceGravity( P, G )`	Adds an inverse-square gravity force on particle Q from particle P.
`Q.AddForceSpring( P, k, Ln, ... )`	Adds a spring force to point Q from point P.
`Q.AddForceDamper( P, b, ... )`	Adds a damper force to point Q from point P.
`S.GetResultantForce()`	Gets the resultant of all forces on S.
`Q.GetResultantForce( P )`	Gets the resultant of all forces on point Q from point P.
`B.AddTorque( torqueVector )`	Adds `torqueVector` to the torque on RigidFrame B.
`B.AddTorque( A, torqueVector )`	Adds `torqueVector` to the torque on RigidFrame B from RigidFrame A.
`B.AddTorqueDamper( A, b, ... )`	Adds a damper torque to RigidFrame B from RigidFrame A.
`B.GetResultantTorque()`	Gets the resultant of all torques on RigidFrame B.
`B.GetResultantForce( A )`	Gets the resultant of all torques on RigidFrame B from RigidFrame A.
`S.GetMomentOfForces( P )`	Gets the moment of all forces on S about point P.

The **M**otion**G**enesis syntax `Force_Q>` denotes a force on point Q.
The **M**otion**G**enesis syntax `Force_Q_P>` denotes the force on point Q from point P.

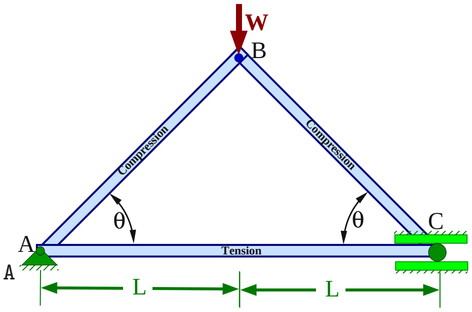
The **M**otion**G**enesis syntax `Torque_B>` denotes a torque on rigid frame (or rigid body) B.
The **M**otion**G**enesis syntax `Torque_B_A>` denotes a torque on B from a rigid frame (or rigid body) A.

Definition and description	Picture	How used & **M**otion**G**enesis commands
$\vec{\mathbf{F}}^{Q/P}$ is the force **on** Q **from** P. Law of action/reaction for forces: $\vec{\mathbf{F}}^{P/Q} = -\vec{\mathbf{F}}^{Q/P}$ (P and Q are points)		Gravity, spring, damper, and other forces have special **M**otion**G**enesis commands. `Q.AddForce( P, someVector )`

Resultant force on point Q:
$\vec{\mathbf{F}}^{Q} \triangleq \sum_{i=1}^{n} \vec{\mathbf{F}}^{Q/P_i}$
	 P_1
	F^{Q/P_1}
	Q
	F^{Q/P_3}
	F^Q
	F^{Q/P_2}
	P_3
	P_2
		$\vec{\mathbf{F}}^{Q} \underset{(\textit{Statics})}{=} \vec{\mathbf{0}}$ $\vec{\mathbf{F}}^{Q} \underset{(\textit{Dynamics})}{=} m^Q\ {}^N\vec{\mathbf{a}}^Q$
		`Q.AddForce( someVector )`
		`Q.GetResultantForce()`

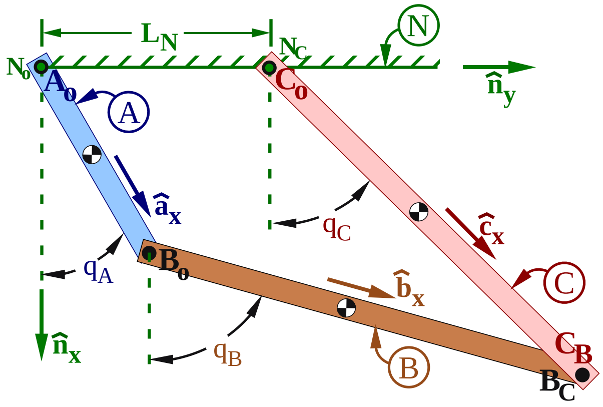
Resultant force on body B: $\vec{\mathbf{F}}^{B} \triangleq \sum_{j=1}^{n} \vec{\mathbf{F}}^{B_j}$		$\vec{\mathbf{F}}^{B} \underset{(\textit{Statics})}{=} \vec{\mathbf{0}}$ $\vec{\mathbf{F}}^{B} \underset{(\textit{Dynamics})}{=} m^B\ {}^N\vec{\mathbf{a}}^{B\text{cm}}$ `B.GetResultantForce()`
Resultant force on system S: $\vec{\mathbf{F}}^{S} \triangleq \sum_{k=1}^{n} \vec{\mathbf{F}}^{Q_k}$		$\vec{\mathbf{F}}^{S} \underset{(\textit{Statics})}{=} \vec{\mathbf{0}}$ $\vec{\mathbf{F}}^{S} \underset{(\textit{Dynamics})}{=} m^S\ {}^N\vec{\mathbf{a}}^{S\text{cm}}$ `System.GetResultantForce()`

8.2 Static truss analysis with MotionGenesis



```
   (1) % File: TrussABCTopLoad.al  (Truss analysis)
   (2) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (3) %--------------------------------------------------------
   (4) NewtonianFrame  N                  % Ground
   (5) RigidFrame      S                  % Entire truss
   (6) Point           A(S), B(S), C(S)   % Nodes on truss
   (7) %--------------------------------------------------------
   (8) Constant  L         % Twice the distance between A and C
   (9) Constant  theta     % Angle between AC and AB
   (10) Constant  W         % Weight applied to node B
   (11) Variable  FAx, FAy  % Nx>, Ny> measures of external force on A
   (12) Variable  FCy       % Ny> measure of external force on E
   (13) Variable  FAB       % Force on A from AB member directed from A to B
   (14) Variable  FAC       % Force on A from AC member directed from A to C
   (15) %--------------------------------------------------------
   (16) %       Relevant external contact and distance forces on S
   (17) A.AddForce( FAx*Nx> + FAy*Ny> )
-> (18) Force_A> = FAx*Nx> + FAy*Ny>

   (19) B.AddForce( -W*Ny> )
-> (20) Force_B> = -W*Ny>

   (21) C.AddForce( FCy*Ny> )
-> (22) Force_C> = FCy*Ny>

   (23) %--------------------------------------------------------
   (24) %       Static analysis of entire system
   (25) ResultantForceOnS> = S.GetResultantForce()
-> (26) ResultantForceOnS> = FAx*Nx> + (FAy+FCy-W)*Ny>

   (27) MomentOfSAboutA> = Cross( L*Nx>, -W*Ny> ) + Cross( 2*L*Nx>, FCy*Ny> )
-> (28) MomentOfSAboutA> = -L*(W-2*FCy)*Nz>

   (29) ZeroSystem[1]  = Dot( ResultantForceOnS>, Nx> )
-> (30) ZeroSystem[1]  = FAx

   (31) ZeroSystem[2]  = Dot( ResultantForceOnS>, Ny> )
-> (32) ZeroSystem[2]  = FAy + FCy - W

   (33) ZeroSystem[3]  = Dot( MomentOfSAboutA>,   Nz> )
-> (34) ZeroSystem[3]  = -L*(W-2*FCy)

   (35) Solve( ZeroSystem,  FAx, FAy, FCy )
-> (36) FAx = 0
-> (37) FAy = 0.5*W
-> (38) FCy = 0.5*W

   (39) %--------------------------------------------------------
   (40) %       Relevant external contact and distance forces on pin A
   (41) UnitVectorFromAToB> = cos(theta)*Nx> + sin(theta)*Ny>
-> (42) UnitVectorFromAToB> = cos(theta)*Nx> + sin(theta)*Ny>

   (43) A.AddForce( B,  FAB * UnitVectorFromAToB> )
-> (44) Force_A_B> = cos(theta)*FAB*Nx> + sin(theta)*FAB*Ny>

   (45) A.AddForce( C,  FAC * Nx> )
-> (46) Force_A_C> = FAC*Nx>

   (47) %--------------------------------------------------------
   (48) %       Static analysis of node A
   (49) ZeroA = Dot( A.GetResultantForce(), [Nx>; Ny>] )  % Creates 2x1 matrix
-> (50) ZeroA = [FAC + FAx + cos(theta)*FAB; FAy + sin(theta)*FAB]

   (51) Explicit( ZeroA )    % Ensure results are explicit in W, L, theta, etc.
-> (52) ZeroA = [FAC + cos(theta)*FAB; 0.5*W + sin(theta)*FAB]

   (53) Solve( ZeroA,  FAB, FAC )
-> (54) FAB = -0.5*W/sin(theta)
-> (55) FAC = 0.5*W*cos(theta)/sin(theta)
```

8.3 Four-bar linkage – static equilibrium (see dynamics in Section 9.22)



The figure to the right shows a planar four-bar linkage consisting of frictionless-pin-connected uniform rigid links A, B, and C and ground N.

- Link A connects to N and B at points A_o and A_B
- Link B connects to A and C at points B_o and B_C
- Link C connects to N and B at points C_o and C_B
- Point N_o of N is coincident with A_o
- Point N_C of N is coincident with C_o

Right-handed orthogonal unit vectors $\hat{\mathbf{a}}_i$, $\hat{\mathbf{b}}_i$, $\hat{\mathbf{c}}_i$, $\hat{\mathbf{n}}_i$ $(i = x, y, z)$ are fixed in A, B, C, N, with:

- $\hat{\mathbf{a}}_x$ directed from A_o to A_B
- $\hat{\mathbf{b}}_x$ directed from B_o to B_C
- $\hat{\mathbf{c}}_x$ directed from C_o to C_B
- $\hat{\mathbf{n}}_x$ vertically downward
- $\hat{\mathbf{n}}_y$ directed from N_o to N_C
- $\hat{\mathbf{a}}_z = \hat{\mathbf{b}}_z = \hat{\mathbf{c}}_z = \hat{\mathbf{n}}_z$ parallel to pin axes

Quantity	Symbol	Value
Length of link A	L_A	1 m
Length of link B	L_B	2 m
Length of link C	L_C	2 m
Distance between N_o and N_C	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
Earth's gravitational acceleration	g	9.81 $\frac{\text{m}}{\text{s}^2}$
$\hat{\mathbf{n}}_y$ measure of force applied to C_B	H	200 N
Angle from $\hat{\mathbf{n}}_x$ to $\hat{\mathbf{a}}_x$ with $+\hat{\mathbf{n}}_z$ sense	q_A	Variable
Angle from $\hat{\mathbf{n}}_x$ to $\hat{\mathbf{b}}_x$ with $+\hat{\mathbf{n}}_z$ sense	q_B	Variable
Angle from $\hat{\mathbf{n}}_x$ to $\hat{\mathbf{c}}_x$ with $+\hat{\mathbf{n}}_z$ sense	q_C	Variable

After creating a "***loop equation***" as

$$L_A\,\hat{\mathbf{a}}_x + L_B\,\hat{\mathbf{b}}_x - L_C\,\hat{\mathbf{c}}_x - L_N\,\hat{\mathbf{n}}_y = \vec{\mathbf{0}}$$

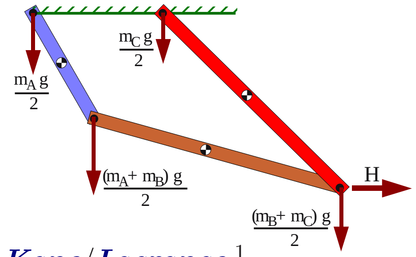
one can differentiate and solve the associated motion constraint equations for $\dot{q}_B$ and $\dot{q}_C$ in terms of $\dot{q}_A$:

$$\begin{aligned} -L_A \sin(q_A)\,\dot{q}_A - L_B \sin(q_B)\,\dot{q}_B + L_C \sin(q_C)\,\dot{q}_C &= 0 \\ L_A \cos(q_A)\,\dot{q}_A + L_B \cos(q_B)\,\dot{q}_B - L_C \cos(q_C)\,\dot{q}_C &= 0 \end{aligned}$$

$$\dot{q}_B = \frac{-L_A \sin(q_A - q_C)}{L_B \sin(q_B - q_C)}\,\dot{q}_A$$

$$\dot{q}_C = \frac{-L_A \sin(q_A - q_B)}{L_C \sin(q_B - q_C)}\,\dot{q}_A$$

The real gravitational forces on each rod can be replaced with equivalent sets. One way to do this is with a single force at each link's center of mass. A better alternative replaces the gravity forces on each rod with half the gravity force at each end. Then, the contribution of gravity forces to $\mathcal{F}_{\dot{q}_A}$ is calculated using a gravity force of $\frac{(m^A + m^B)\,g}{2}\,\hat{\mathbf{n}}_x$ at B_o and a gravity force of $\frac{(m^B + m^C)\,g}{2}\,\hat{\mathbf{n}}_x$ at C_B.



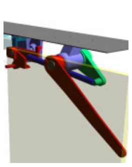
1. Form this system's generalized force $\mathcal{F}_{\dot{q}_A}$ for static equilibrium via ***Kane/Lagrange***.[1]
   Result: (Use "embedded method" which accounts for constraints and eliminates all constraint/reaction forces)

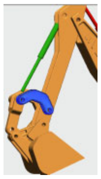
$$\mathcal{F}_{\dot{q}_A} = \frac{-L_A}{2}\left\{ (m^A + m^B)\,g\,\sin(q_A) + \frac{\sin(q_A - q_B)}{\sin(q_B - q_C)}\left[2\,H\,\cos(q_C) - (m^B + m^C)\,g\,\sin(q_C)\right]\right\}$$

2. Determine the **static equilibrium** values of q_A, q_B, q_C.
   Optional:** Using your intuition (or guessing), circle the ***stable*** solution.[2]

$\boxed{\text{Solution 1}}$	$q_A = 20.0°$	$q_B = 71.7°$	$q_C = 38.3°$
Solution 2	$q_A = 249.3°$	$q_B = 140.2°$	$q_C = 199.1°$
Solution 3	$q_A = 30.7°$	$q_B = 226.1°$	$q_C = 254.7°$

Solutions to 2^+ significant digits

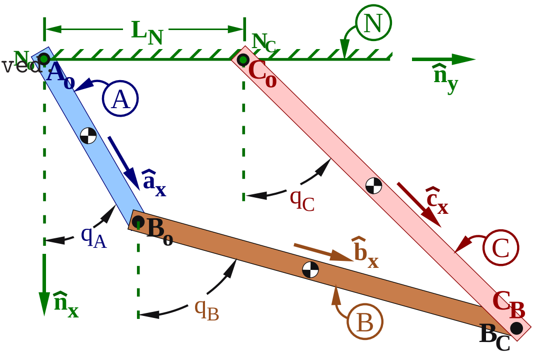




---

[1] The form of the expression for $\mathcal{F}_{\dot{q}_A}$ depends on how it is calculated. The result for $\mathcal{F}_{\dot{q}_A}$ presented here replaces gravity forces with equivalent sets at the end of the rods (shown above), and uses ${}^N\vec{\mathbf{v}}^{B_C} = L_C\,\dot{q}_C\,\hat{\mathbf{c}}_y$.

[2] **Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Four-bar linkage.**



```
% File: MGFourBarStaticsKaneEmbedded.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%--------------------------------------------------
%       Physical objects.
NewtonianFrame  N
RigidBody       A, B, C
Point           CB(C)
%--------------------------------------------------
%       Mathematical declarations.
Constant   LA = 1 m,  LB = 2 m,  LC = 2 m,  LN = 1 m
Constant   g = 9.81 m/s^2          % Gravity
Constant   H = 200 Newtons         % Horizontal force
Variable   qA', qB', qC'           % Angles
SetGeneralizedSpeed( qA' )
%--------------------------------------------------
A.SetMass( mA = 10 kg )
B.SetMass( mB = 20 kg )
C.SetMass( mC = 20 kg )
%--------------------------------------------------
%       Rotational kinematics
A.RotateZ( N, qA )
B.RotateZ( N, qB )
C.RotateZ( N, qC )
%--------------------------------------------------
%       Translational kinematics
Bo.SetPositionVelocity(  No,  LA*Ax> )
CB.SetPositionVelocity(  No,  LN*Ny> + LC*Cx> )
%--------------------------------------------------
%       Forces - replaces gravity forces with equivalent set
Bo.AddForce( 1/2*( A.GetMass()*g*Nx> + B.GetMass()*g*Nx> ) )
CB.AddForce( 1/2*( B.GetMass()*g*Nx> + C.GetMass()*g*Nx> ) )
CB.AddForce( H*Ny> )      % Horizontal force on CB
%--------------------------------------------------
%       Configuration constraints and motion constraints.
Loop> = LA*Ax> + LB*Bx> - LC*Cx> - LN*Ny>
Loop[1] = Dot( Loop>, Nx> )
Loop[2] = Dot( Loop>, Ny> )
MotionConstraint = Dt( Loop )
Solve( MotionConstraint,  qB', qC' )
%--------------------------------------------------
%       Equations of motion
Zero = System.GetStaticsKane()
Zero[2] = Loop[1]
Zero[3] = Loop[2]
%--------------------------------------------------
%       Solve nonlinear equations
Solve( Zero,  qA = 30 deg,  qB = 60 deg,  qC = 30 deg )
%--------------------------------------------------
Save MGFourBarStaticsKaneEmbedded.all
Quit
```

Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Four-bar linkage.

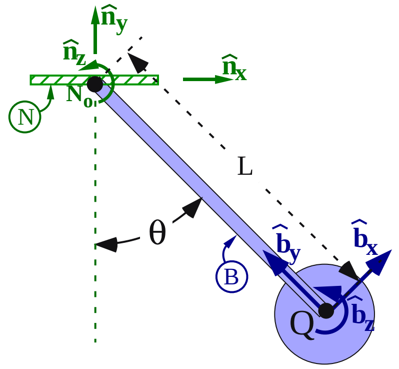
Chapter 9

Equations of motion

9.1 MotionGenesis statics and dynamics commands

Command	Description
`S.GetStatics()`	Forms the resultant of all forces on point, particle, body, or system S.
`S.GetStatics( P )`	Forms the moment of all forces on S about point P.
`S.GetDynamics()`	Forms $\vec{\mathbf{F}} = m\,\vec{\mathbf{a}}$ for particle, body, or systems S.
`S.GetDynamics( P )`	Forms Euler's equation for S about point P.

9.2 Motion simulation of classic particle pendulum

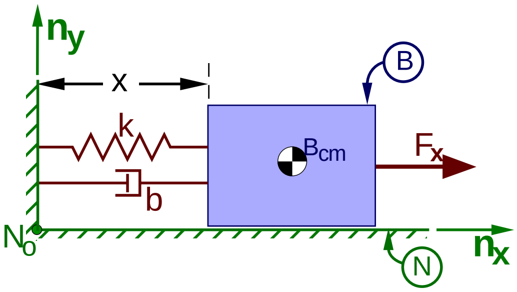


$\hat{\mathbf{n}}_\mathbf{y}$
$\hat{\mathbf{n}}_\mathbf{z}$
$\hat{\mathbf{n}}_\mathbf{x}$
$\mathbf{N_o}$
N
L
θ
B
$\hat{\mathbf{b}}_\mathbf{y}$
$\hat{\mathbf{b}}_\mathbf{x}$
Q
$\hat{\mathbf{b}}_\mathbf{z}$

```
% File:  ClassicParticlePendulumEuler.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%---------------------------------------------------------------
NewtonianFrame  N         % Newtonian reference frame (ground)
RigidFrame      B         % Massless inextensible (rigid) string
Particle        Q         % Particle at end of string
%---------------------------------------------------------------
Variable  theta''         % Pendulum angle
Constant  L = 50 cm       % Length of string
Constant  g = 9.8 m/s^2   % Earth's gravitational acceleration
Q.SetMass( m = 2 kg )
%---------------------------------------------------------------
%       Rotational/translational kinematics and relevant forces.
B.RotateZ( N, theta )
Q.Translate( No, -L*By> )
Q.AddForceGravity( -g*Ny> )
%---------------------------------------------------------------
%       Equations of motion (angular momentum principle)
Zero> = System.GetDynamics( No )
Zero = Dot(  Zero>,  Nz>  )
Solve( Zero,  theta'' )
%---------------------------------------------------------------
KE = System.GetKineticEnergy()
PE = Q.GetForceGravityPotentialEnergy(  -g*Ny>,  No  )
MechanicalEnergy = KE + PE
%---------------------------------------------------------------
%   Integration parameters and initial values of variables.
Input  tFinal = 10 sec,  tStep = 0.02 sec,  absError = 1.0E-08
Input  theta = 30 deg,  theta' = 0 deg/sec
%---------------------------------------------------------------
%       List output quantities and solve ODEs.
Output  t sec,  theta deg,  theta' deg/sec,  KE Joules,  PE Joules,  MechanicalEnergy Joules
ODE()  ClassicParticlePendulumEuler
Save ClassicParticlePendulumEuler.all
Quit
```

Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Pendulum.

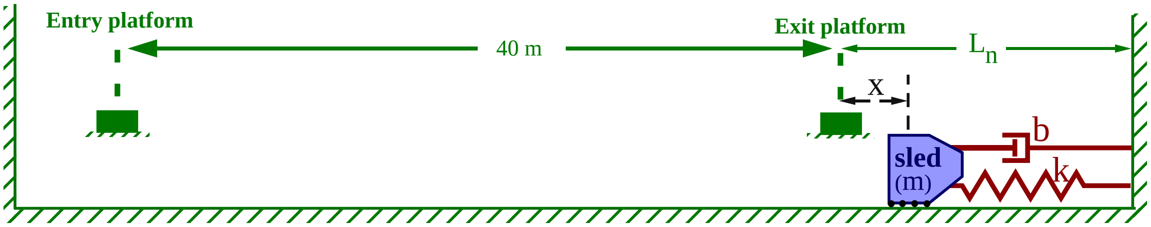
9.3 Motion simulation of a block on a rough surface



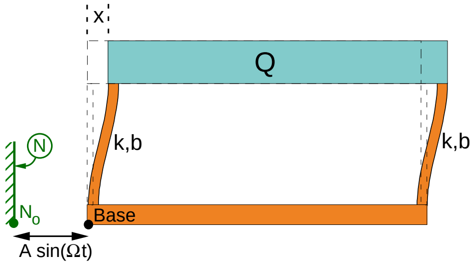
```
%    File: OneBlockMassSpringDamperSimple.al
% Purpose: Simulate mass-spring-damper on rough table.
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%-----------------------------------------------------------
NewtonianFrame  N
Particle        B            % Block
%-----------------------------------------------------------
Variable   x''               % B's horizontal displacement
Variable   Fn                % Resultant normal   force on B
Variable   Ff                % Resultant friction force on B
Specified  Fx = 2*cos(t)     % Specified horizontal force on B
Constant   g = 9.8 m/s^2     % Earth's gravitational acceleration
Constant   k = 10 N/m        % Linear spring constant
Constant   Ln = 0.1 m        % Natural length of spring
Constant   b = 1 N*sec/m     % Linear damper constant
Constant   muK = 0.1 noUnits % Coefficient of kinetic friction
Constant   epsilon = 1.0E-6 noUnits  % For Continuous Friction Law
B.SetMass( m = 1 kg )
%-----------------------------------------------------------
%       Position, velocity, acceleration
B.Translate( No, x*Nx> )
%-----------------------------------------------------------
%       Forces on B
B.AddForce( -m*g*Ny>  )               % Gravitational force
B.AddForce( -(k*(x-Ln) + b*x')*Nx> )  % Spring/damper force
B.AddForce( Fn*Ny> + Ff*Nx> )         % Normal and friction forces
B.AddForce( Fx*Nx> )                  % Specified force
%-----------------------------------------------------------
%       Equations of motion for B via F = m*a
Zero> = B.GetDynamics()
Zero[1] = Dot( Zero>, Nx> )
Zero[2] = Dot( Zero>, Ny> )
%-----------------------------------------------------------
%       Additional equation when B is sliding on N
%       Note: Use Continuous Friction Law to simulate sticking and sliding.
%       Set epsilon to a small positive number to avoid divide-by-zero problems.
magVelocityPlusEpsilon = B.GetSpeed(N) + epsilon
Ff = Dot( -muK*Fn*B.GetVelocity(N) / magVelocityPlusEpsilon, Nx> )
%-----------------------------------------------------------
%       Solve sliding equation of motion for x''
Solve( Zero, x'', Fn )
%-----------------------------------------------------------
%       Integration parameters and initial values for variables.
Input  tFinal = 10 sec,  tStep = 0.01 sec,  absError = 1.0E-08
Input  x = 0.1 m,  x' = 0 m/sec
%-----------------------------------------------------------
%       List output quantities and solve ODEs.
Output  t sec,  x m,  x' m/sec,  Ff Newton
ODE()  OneBlockMassSpringDamperSimple
%-----------------------------------------------------------
Save  OneBlockMassSpringDamperSimple.all
Quit
```

9.4 Equations of motion for a rocket sled ride

```
% File: DisneyRideEquationOfMotion.al
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%---------------------------------------------------------------
NewtonianFrame N        % Newtonian reference frame
Particle       Q        % Rocket-sled
Variable       x''      % Nx> measure of Q's position from No
Constant       k, b     % Spring/damper constants
Constant       Ln       % Natural length of spring
Q.SetMass( m )
%---------------------------------------------------------------
%       Q's position vector, velocity, and acceleration
Q.Translate( No, x*Nx> )
%---------------------------------------------------------------
%       Add relevant forces
SpringLength = Ln - x
SpringStretch = SpringLength - Ln
Q.AddForce( k*Explicit(SpringStretch)*Nx> )
Q.AddForce( -b*Q.GetVelocity(N) )
%---------------------------------------------------------------
%       Form and solve Newton's equation of motion (translation)
Zero> = Q.GetDynamics()
Zero = Dot( Zero>, Nx> )
Solve( Zero, x'' )
%---------------------------------------------------------------
%       Given adult values of m, b, k and initial values of x, x'
Input  m=200 kg, b=300 N*sec/meter, k=139.2 N/meter
Input  x=-40 meters, x'=0 m/sec
%---------------------------------------------------------------
%       Initial acceleration for given adult values
InitialAdultAcceleration> = EvaluateAtInput( Q.GetAcceleration(N) )
InitialAdultAccelerationMag = GetMagnitude( InitialAdultAcceleration> )
NumberOfGsForAdult = InitialAdultAccelerationMag / 9.8
%---------------------------------------------------------------
%       Initial acceleration for given child values
InitialChildAcceleration> = EvaluateAtInput( Q.GetAcceleration(N), m=100 kg )
NumberOfGsForChild = GetMagnitude( InitialChildAcceleration> ) / 9.8
%---------------------------------------------------------------
%       Plot x(t) for 10 seconds.
Input  tFinal = 10 seconds
OutputPlot  t sec,  x meters
Ode() DisneyRideEquationOfMotion
%---------------------------------------------------------------
%       Calculate zeta and wn for both adult and child
zeta = b / (2*sqrt(m*k) );    wn = sqrt(k/m)
zetaAdult = EvaluateAtInput(zeta);        wnAdult = EvaluateAtInput(wn)
zetaChild = EvaluateAtInput(zeta,m=100);  wnChild = EvaluateAtInput(wn,m=100)
%---------------------------------------------------------------
%       Save results
Save DisneyRideEquationOfMotion.all
Quit
```

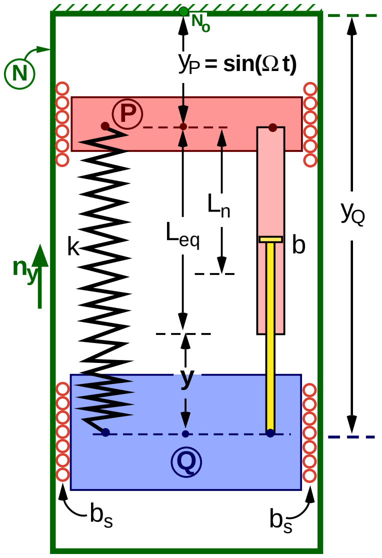


9.5 Motion simulation of a building in an earthquake



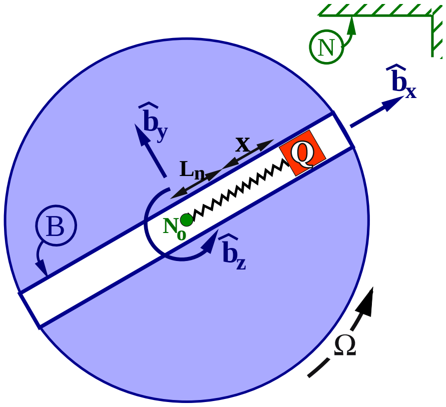
```
%    File:  SingleStoryBuildingInEarthquake.al
% Problem:  Forced base motion of building
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%---------------------------------------------------------------
NewtonianFrame N     % Newtonian reference frame
Point          Base  % Base of building (modeled as a point)
Particle       Q     % Roof of building (modeled as a particle)
%---------------------------------------------------------------
Variable  x’’                        % Spring stretch
Constant  k = 10000 N/m              % Linear spring constant
Constant  b = 500 N*sec/m            % Linear damping constant
Constant  Amp = 0.1 m                % Earthquake amplitude
Q.SetMass( m = 5000 kg )
wn = EvaluateAtInput( sqrt(2*k/m) )  % Building’s natural frequency
Constant  Omega = 0.8*wn rad/sec     % Earthquake forcing frequency
%---------------------------------------------------------------
%       Position vectors, velocities, and accelerations
Base.Translate( No, Amp*sin(Omega*t)*Nx> )
Q.Translate( Base,  x*Nx> )
%---------------------------------------------------------------
%       Spring and damper forces
SpringForce> = -2*k*x*Nx>
DamperForce> = -2*b*x’*Nx>
Q.AddForce( SpringForce> + DamperForce> )
%---------------------------------------------------------------
%       Form equations of motion via  F = m*a  and solve for x’’
Zero = Dot( Q.GetDynamics(), Nx> )
Solve( Zero, x’’ )
%---------------------------------------------------------------
%       Integration parameters and initial values of variables.
Input  tFinal = 40 sec,  tStep = 0.02 sec,  absError = 1.0E-08
Input  x = 0 m,  x’ = 0 m/sec
%---------------------------------------------------------------
%       List output quantities and solve ODEs.
Output  t sec,  x m,  x’ m/s,  x’’ m/s^2
ODE() SingleStoryBuildingInEarthquake
%---------------------------------------------------------------
%       Record input and program responses
Save SingleStoryBuildingInEarthquake.all
Quit
```

9.6 Equation of motion of Scotch-yolked mechanism



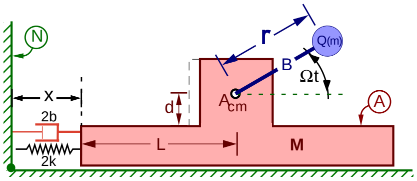
```
%    File: MassSpringDamperVerticalForcedScotchYolk.al
% Problem: Forced harmonic motion of mass/spring/damper system.
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%--------------------------------------------------------------------
NewtonianFrame N   % Newtonian reference frame
Point          P   % Platform (modeled as a moving point)
Particle       Q   % Roof of building (modeled as a moving particle)
%--------------------------------------------------------------------
Variable  y’’      % Spring stretch (from natural length)
Specified yP’’     % Known vertical motion of Scotch-yolk
Constant  k        % Linear spring/damper spring constant
Constant  Ln       % Natural length of spring
Constant  Leq      % Equilibrium length of spring
Constant  b        % Linear spring/damper damping constant
Constant  bs       % Linear viscous damping constant between Q and N
Constant  g        % Local gravitational acceleration
Q.SetMass( m )
%--------------------------------------------------------------------
%       Position vectors, velocities, and accelerations
P.Translate( No, -yP*Ny> )
Q.Translate( P,  -(y+Leq)*Ny> )
%--------------------------------------------------------------------
%       Relevant contact/distance forces
SpringStretch = y + Leq - Ln
SpringForce> = k * Explicit(SpringStretch) * Ny>
DamperForce> = b * Dt(SpringStretch) * Ny>
ViscousForce> = -2 * bs * Q.GetVelocity(N)
GravityForce> = -Q.GetMass() * g *Ny>
Q.AddForce( SpringForce> + DamperForce> + ViscousForce> + GravityForce> )
%--------------------------------------------------------------------
%       Downward measure of Newton’s equation of motion
Zero = Dot( Q.GetDynamics(), -Ny> )
%--------------------------------------------------------------------
%       Use static equilibrium to simplify equation of motion
ZeroStatics = Evaluate( Zero, y=0, y’=0, y’’=0, yP’=0, yP’’=0 )
ZeroDynamics = Explicit( Zero - ZeroStatics )
%--------------------------------------------------------------------
%       Examine bs = 0 and harmonic forcing for yP
Constant  A, Omega    % Amplitude/frequency of Scotch-yolk vertical motion
SetDt( yP = A*sin(Omega*t) )
ZeroHarmonicForcing = Evaluate( ZeroDynamics, bs=0, yP’’ = -A*Omega^2*sin(Omega*t) )
FactorLinear( ZeroHarmonicForcing, y’’, y’, y )
%--------------------------------------------------------------------
%       Record input and program responses
Save MassSpringDamperVerticalForcedScotchYolk.all
Quit
```

9.7 Equation of motion for a particle on spinning slot

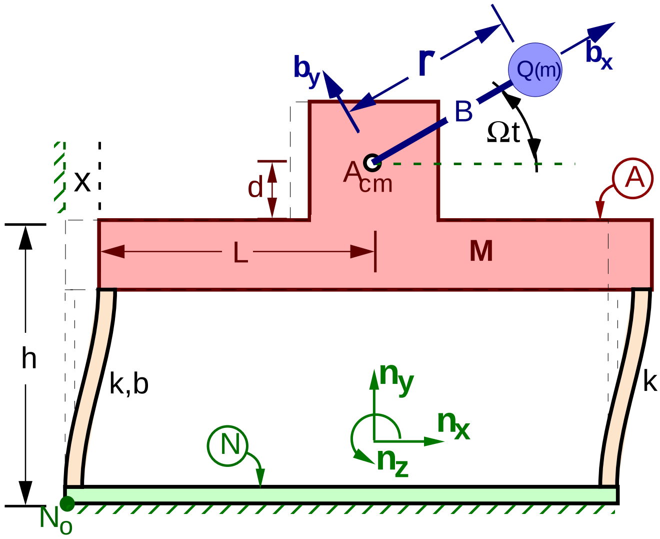


```
% File: ParticleOnSpinningSlotFMA.al
%---------------------------------------------------------------
NewtonianFrame  N
RigidBody       B
Particle        Q
%---------------------------------------------------------------
Variable   x’’      % Distance from No to Q; derivatives
Constant   b        % Linear damping constant
Constant   k, Ln    % Linear spring constant and natural length
Specified  Omega’   % Bz> measure of B’s angular velocity in N
Q.SetMass( m )
%---------------------------------------------------------------
%       Rotational and translational kinematics
B.SetAngularVelocityAcceleration( N, Omega*Bz> )
Q.Translate( No, (Ln+x)*Bx> )
%---------------------------------------------------------------
%       Relevant forces and torques
Q.AddForce( -(k*x + b*x’)*Bx> )
%---------------------------------------------------------------
%       Relevant translational/rotational equations of motion
Zero = Dot( Q.GetDynamics(), Bx> )
%---------------------------------------------------------------
%       Record input together with responses
Save ParticleOnSpinningSlotFMA.all
Quit
```

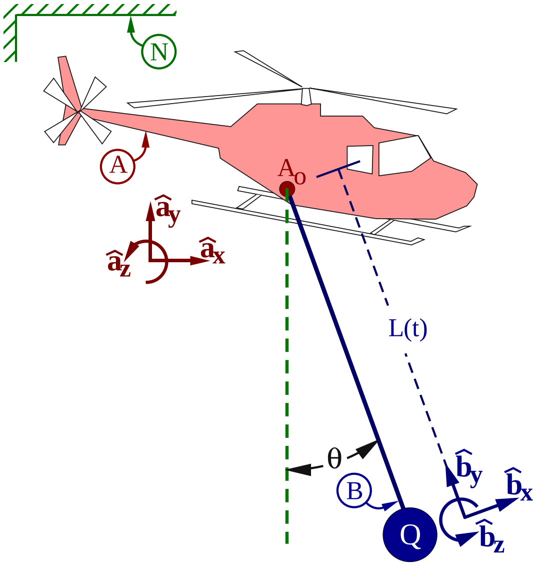
9.8 Equation of motion for unbalanced motor on roof



```
%    File: EccentricParticleAirConditionerForcedVibrationFBD1.al
% Purpose: Determine the effect on the motion of a one-story building
%          of an air-conditioner that is mounted on the roof - and
%          which has an unbalance motor, modeled as an eccentric particle.
%---------------------------------------------------------------------
NewtonianFrame  N
RigidFrame B          % Air-conditioner motor Frame
Particle   A          % The roof modeled as a particle (no rotation)
Particle   Q          % Eccentric particle
%---------------------------------------------------------------------
Variable   x’’        % Horizontal displacement of A
Constant   g          % Local gravitational acceleration
Constant   k          % Effective stiffness in each column
Constant   b          % Effective damping   in each column
Constant   h          % Height of roof
Constant   d          % Height of motor about roof
Constant   L          % Distance from left edge of roof to air-conditioner
Constant   r          % Eccentric distance (from A to Q)
Constant   W          % Motor spin rate
A.SetMass( mA )
Q.SetMass( mQ  )
%---------------------------------------------------------------------
%       Rotational and translational kinematics.
B.RotateZ( N, W*t )
A.Translate( No, (x+L)*Nx> + (h+d)*Ny> )
Q.Translate( A,   r*bx> )
%---------------------------------------------------------------------
%       Relevant forces on system A, B, Q
Q.AddForce( -mQ*g*Ny> )        % Gravity force on Q
SpringForce> = -2*k*x*Nx>      % Spring  force on A
DamperForce> = -2*b*x’*Nx>     % Damper  force on A
A.AddForce( SpringForce> + DamperForce> )
%---------------------------------------------------------------------
%       Equation of motion for system A, B, Q
Zero = Dot( Nx>, A.GetDynamics() + Q.GetDynamics() )
Solve( Zero, x’’ )
%---------------------------------------------------------------------
Save  EccentricParticleAirConditionerForcedVibrationFBD1.all
Quit
```



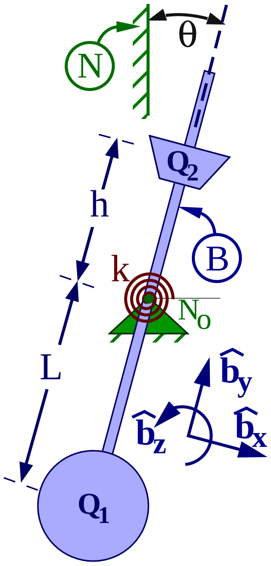
9.9 Motion simulation of helicopter rescue



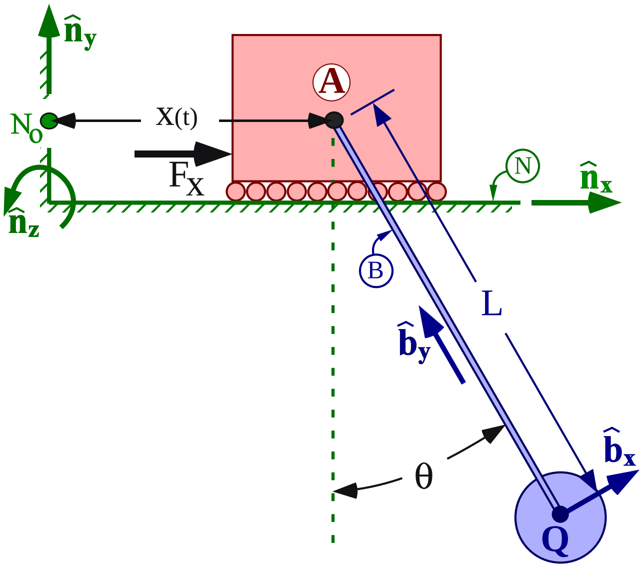
```
%     File: StationaryHelicopterRetrievalFMA.txt
%  Problem: Retrieval of capsized fishermen
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%--------------------------------------------------------------
NewtonianFrame N          % Earth
RigidFrame     B          % Cable
Particle       Q          % Rescue basket and fishermen
%--------------------------------------------------------------
Q.SetMass( m = 100 kg )
Constant   g = 9.8 m/s^2  % Local gravitational acceleration
Variable   theta''        % Pendulum swing angle
Variable   Tension        % Tension in cable
Specified  L''            % Cable length (varies)
SetDt( L = 50 - 2*t )
%--------------------------------------------------------------
%       Rotation and translation kinematics
B.RotateZ( N, theta )
Q.Translate( No, -L*By> )
%--------------------------------------------------------------
%       Relevant contact and distance forces
Q.AddForce( -m*g*Ny> + Tension*By> )
%--------------------------------------------------------------
%       Form equations of motion using F = m *a
ZeroFMA = Dot( Q.GetDynamics(), Bx> )
Solve( ZeroFMA, theta'' )
%--------------------------------------------------------------
%       Input initial values, and numerical integration parameters
Input   theta = 1 deg,  theta' = 0 deg/sec
Input   tFinal = 24.92 sec,  tStep = 0.02 sec
%--------------------------------------------------------------
%       List output quantities and solve ODEs.
OutputPlot  t sec,  theta deg
ODE() StationaryHelicopterRetrievalFMA
%--------------------------------------------------------------
%       Record input together with responses
Save StationaryHelicopterRetrievalFMA.all
Quit
```

9.10 Equations of motion of a metronome

```
% File: MetronomeEquationOfMotion.al
%---------------------------------------------------------------
NewtonianFrame N
RigidFrame     B        % Metronome rod
Particle       Q1, Q2   % Particles fixed at distal ends of B
%---------------------------------------------------------------
Variable  theta''       % Metronome angle and two derivatives
Constant  L             % Distance between No and Q1
Constant  h             % Distance between No and Q2
Constant  g             % Local gravitational acceleration
Constant  k             % Torsional spring constant
Q1.SetMass( m1 )
Q2.SetMass( m2 )
%---------------------------------------------------------------
%       Rotational and translational kinematics.
B.RotateNegativeZ( N, theta )
Q1.Translate( No, -L*By> )
Q2.Translate( No,  h*By> )
%---------------------------------------------------------------
%       Relevant forces and torques
System.AddForceGravity( -g*Ny> )
B.AddTorque( k*theta*Bz> )
%---------------------------------------------------------------
%       System rotational equations of motion about No
Zero = Dot( System.GetDynamics(No), -Bz> )
Solve( Zero, theta'' )
Save MetronomeEquationOfMotion.all
Quit
```

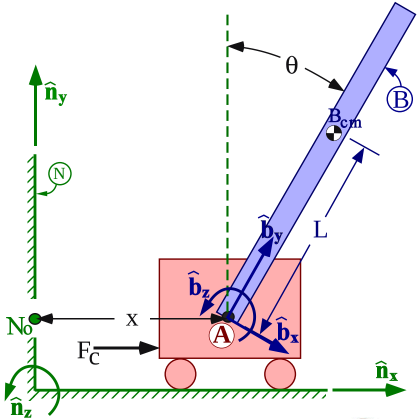


9.11 Equations of motion for a bridge crane



```
% File: BridgeCraneXTheta.al
%------------------------------------------
NewtonianFrame N
Particle       A
RigidFrame     B
Particle       Q
%------------------------------------------
A.SetMass( mA )
Q.SetMass( mQ )
%------------------------------------------
Constant   g
Constant   L
Specified  x’’
Variable   Fx
Variable   q’’
%------------------------------------------
B.RotateZ( N, q )
A.Translate( No, x*Nx> )
Q.Translate( A,  -L*By> )
%------------------------------------------
A.AddForce( Fx*Nx> )
System.AddForceGravity( -g*Ny> )
%--------------------------------------------------------------
%       Kinetic energy, potential energy, and work done on system.
KE = System.GetKineticEnergy()
PE = System.GetForceGravityPotentialEnergy( -g*Ny>, No )
Variable  WorkDoneBySystem’ = Dot(  Fx*Nx>,  A.GetVelocity(N)  )
EnergyConstant = KE + PE - WorkDoneBySystem
%--------------------------------------------------------------
%       Lagrange’s equations of motion
LhsLagrange =  Dt( D( KE, q’ ) )  -  D( KE, q )
vQNPartial> =  D( Q.GetVelocity(N), q’, N )
RhsLagrange =  Dot( vQNPartial>, Q.GetResultantForce() )
ZeroLagrange =  Explicit( LhsLagrange - RhsLagrange )
%--------------------------------------------------------------
%       Newton/Euler equations of motion
Zero[1] = Dot( Bz>, Q.GetDynamics(A) + B.GetDynamics(A) )
Zero[2] = Dot( Nx>, Q.GetDynamics()  + B.GetDynamics() + A.GetDynamics() )
%--------------------------------------------------------------
%       Solve equations of motion for q’’ and Fx.
Solve( Zero, q’’, Fx )
%--------------------------------------------------------------
%       Record input together with responses
Save BridgeCraneXTheta.all
Quit
```

9.12 Dynamics and control of an inverted pendulum on cart



The figure to the right shows a rigid inverted pendulum B attached by a frictionless revolute joint to a cart A (modeled as a particle). The cart A slides on a straight horizontal frictionless track. The track is fixed in a Newtonian reference frame N.

Right-handed orthogonal unit vectors $\widehat{\mathbf{n}}_x$, $\widehat{\mathbf{n}}_y$, $\widehat{\mathbf{n}}_z$ and $\widehat{\mathbf{b}}_x$, $\widehat{\mathbf{b}}_y$, $\widehat{\mathbf{b}}_z$ are fixed in N and B respectively, with:

- $\widehat{\mathbf{n}}_x$ horizontally right and $\widehat{\mathbf{n}}_y$ vertically upward
- $\widehat{\mathbf{n}}_z = \widehat{\mathbf{b}}_z$ parallel to B's axis of rotation in N
- $\widehat{\mathbf{b}}_y$ directed from A to the distal end of B

Quantity	Symbol	Value
Mass of A	m^A	10.0 kg
Mass of B	m^B	1.0 kg
Distance between A and B_{cm} (B's center of mass)	L	0.5 m
B's moment of inertia about B_{cm} for $\widehat{\mathbf{b}}_z$	I_{zz}	0.08333 kg∗m^2
Earth's gravitational constant	g	9.8 m/s^2
$\widehat{\mathbf{n}}_x$ measure of feedback-control force applied to A	F_c	**Specified**
$\widehat{\mathbf{n}}_x$ measure of A's position vector from N_o (a point fixed in N)	x	Variable
Angle from $\widehat{\mathbf{n}}_y$ to $\widehat{\mathbf{b}}_y$ with -$\widehat{\mathbf{n}}_z$ sense	θ	Variable

- Form equations of motion for the system.
  Result:
$$(m^A + m^B)\,\ddot{x} \;+\; m^B\,L\,\cos(\theta)\,\ddot{\theta} \;-\; m^B\,L\,\sin(\theta)\,\dot{\theta}^2 \;=\; F_c$$
$$m^B\,L\,\cos(\theta)\,\ddot{x} \;+\; (\mathrm{I}_{zz} + m^B\,L^2)\,\ddot{\theta} \;-\; m^B\,g\,L\,\sin(\theta) \;=\; 0$$

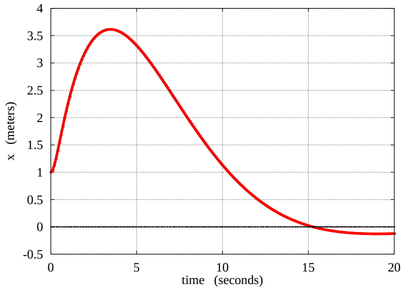
- Consider the nominal solution $x = \dot{x} = \ddot{x} = \theta = \dot{\theta} = \ddot{\theta} = 0$.
  Find $F_{c\text{nom}}$ (the nominal value of F_c) required for this nominal solution to satisfy the equations of motion.
  Result: The **M**otion**G**enesis output shows `Check = [-FcNominal; 0]`
  which means for this solution, the nominal value of F_c is $F_{c\text{nom}} = 0$.

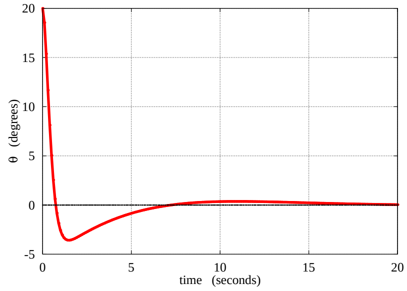
- After introducing the variables `dx`, `dx'`, `dx''`, and `dtheta`, `dtheta'`, `dtheta''` as perturbations of x and θ and their time-derivatives, linearize the equations of motion in perturbations about the aforementioned nominal solution.
  Result:
$$\begin{bmatrix} m^A + m^B & m^B\,L \\ m^B\,L & \mathrm{I}_{zz} + m^B\,L^2 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \text{-}m^B\,g\,L \end{bmatrix} \begin{bmatrix} x \\ \theta \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} F_c \end{bmatrix}$$

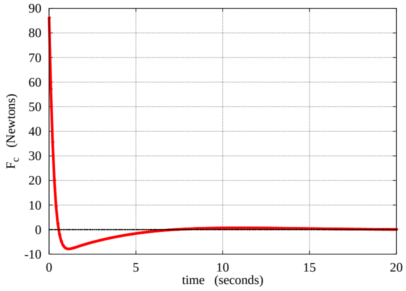
- Put the linearized equations in the form $\dot{X} = A\,X + B\,F_c$ where A and B are 4×4 matrices and X is the 4×1 state matrix `[dx; dtheta; dx'; dtheta']`. A state-space controller specifies F_c as
$$F_c \;=\; k_1\,x \;+\; k_2\,\theta \;+\; k_3\,\dot{x} \;+\; k_4\,\dot{\theta} \qquad \text{where } k_i \ (i{=}1, 2, 3, 4) \text{ are constants (feedback-control gains).}$$

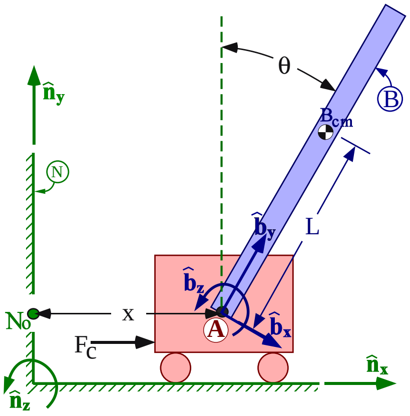
  Show that with $k_1 = k_2 = k_3 = k_4 = 0$, the solution is **unstable**.
  Show that with $k_1 = 1$, $k_2 = 244$, $k_3 = 5.4$, $k_4 = 59$, the solution is **stable**.
  Result: The **M**otion**G**enesis output shows the eigenvalues associated with the A matrix are
$$\text{RootsNoControl} \approx \begin{bmatrix} \text{-}4 & 0 & 0 & +4 \end{bmatrix} \qquad \text{Since an eigenvalue is positive, the solution is unstable.}$$
$$\text{RootsWithControl} \approx \begin{bmatrix} \text{-}3.8 - 1.3\,i & \text{-}3.8 + 1.3\,i & \text{-}0.22 - 0.2\,i & \text{-}0.22 + 0.2\,i \end{bmatrix}$$
  Since all eigenvalues for RootsWithControl have negative real parts, the solution is **stable** for "small" disturbances.

- Simulate the linearized and nonlinear controlled equations of motion for 20 sec. Use initial values `x = dx = 1 m`, `theta = dtheta =` 20°, `x' = dx' = 0`, `theta' = dtheta' = 0`.
  Plot x, θ, and F_c versus time.





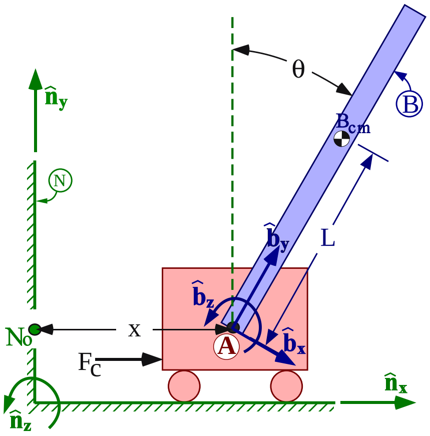




```
%  File: InvertedPendulumOnCartWithControl.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%-----------------------------------------------------------------
NewtonianFrame  N
Particle        A            % Cart
RigidBody       B            % Inverted pendulum
%-----------------------------------------------------------------
Variable   x’’               % Distance between No to A
Variable   theta’’           % Angle from local vertical to B’s long axis
Constant   g+ = 9.81 m/s^2   % Gravitational constant
Constant   L+ = 0.5  m       % Distance between A and Bcm
Specified  Fc
A.SetMass( mA = 10 kg )
B.SetMassInertia( mB = 1 kg,  Izz = 1/12*mB*(2*L)^2,  0,  Izz )
%-----------------------------------------------------------------
%       Rotational and translational kinematics
B.RotateNegativeZ( N, theta )
A.Translate( No, x*Nx> )
Bcm.Translate( A, L*By> )
%-----------------------------------------------------------------
%       Relevant contact and distance forces
System.AddForceGravity( -g*Ny> )
A.AddForce( Fc*Nx> )
%-----------------------------------------------------------------
%       Form Kane’s equations of motion
SetGeneralizedSpeed( x’, theta’ )
Zero = System.GetDynamicsKane()

%*********************************************************************
%       CONTROL SYSTEM / STABILITY ANALYSIS
%*********************************************************************
%       Linearization: Perturbation variables
Variable   dx’’             % Perturbations of x, x’, x’’
Variable   dtheta’’         % Perturbations of theta, theta’, theta’’
Specified  dFc              % Perturbation of Fc.
Variable   FcNominal        % Nominal solution for Fc.
SetImaginaryNumber( i )
%---------------------------------------------------------------------
%       Check conditions for solution: x = x’ = x’’ = 0  and  theta = theta’ = theta’’ = 0.
Check = Evaluate( Zero,  x=0,  x’=0,  x’’=0,  theta=0,  theta’=0,  theta’’=0, Fc=FcNominal )
%---------------------------------------------------------------------
%       Linearize equations of motion about nominal solution
Perturb = Linearize1( Zero, x = 0:dx, x’ = 0:dx’, x’’ = 0:dx’’,         &
          theta = 0:dtheta,  theta’ = 0:dtheta’,  theta’’ = 0:dtheta’’, Fc=0:dFc )
Solve( Perturb,  dx’’, dtheta’’ )
%---------------------------------------------------------------------
%       Form matrix of peturbations and its time-derivative
Xm = [ dx ;  dtheta ;  dx’ ;  dtheta’ ]    % Matrix of perturbations
Xp = dt( Xm )
%---------------------------------------------------------------------
%       Form/simplify matrices A and B  so  Xm’ = A * Xm  + B * Fc.
A = Expand( GetCoefficientMatrix( Xp,  Xm ) )
B = Expand( GetCoefficientMatrix( Xp,  dFc ) )
%---------------------------------------------------------------------
%       Stability when uncontrolled (Fc = 0) and with control.
RootsNoControl = GetEigen( EvaluateAtInput( A ) )
```

```
%--------------------------------------------------------------------
%       Stability with feedback control (k1, k2, k3, k4 are gains).
Constant  k1 = 1.0 N/m,  k2 = 244 N/rad,  k3 = 5.4 N*s/m,  k4 = 59  N*s/rad
K = [k1, k2, k3, k4]
RootsControlled = GetEigen(  EvaluateAtInput( A + B*K )  )
%--------------------------------------------------------------------
Fc = k1*x + k2*theta + k3*x' + k4*theta'
dFc = k1*dx + k2*dtheta + k3*dx' + k4*dtheta'
%--------------------------------------------------------------------
%       Integration parameters and initial values.
Input  tFinal = 20,  tStep = 0.1,  absError = 1.0E-08
Input   x = 1 m,   theta = 20 deg,   x' = 0 m/s,   theta' = 0 rad/s
Input  dx = 1 m,  dtheta = 20 deg,  dx' = 0 m/s,  dtheta' = 0 rad/s
%--------------------------------------------------------------------
%       Quantities to be output by ODE command.
Output  t sec,  x m,  dx m,  theta deg,  dtheta deg,  Fc Newtons
ODE( Zero,  x'',  theta''  ) InvertedPendulumOnCartWithControl
%--------------------------------------------------------------------
Save InvertedPendulumOnCartWithControl.all
Quit
```

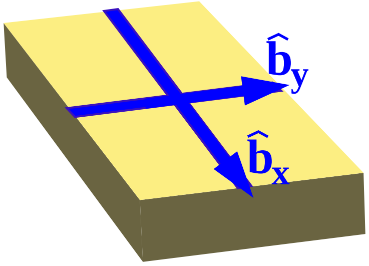
Alternately, form equations of motion via Newton/Euler

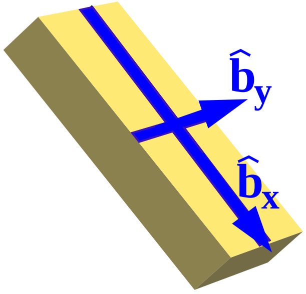


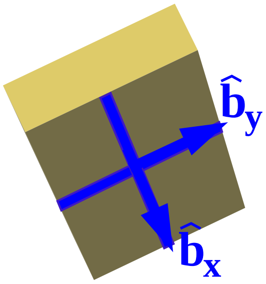
```
%  File: InvertedPendulumOnCartDynamics.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%----------------------------------------------------------------
NewtonianFrame  N
Particle        A             % Cart
RigidBody       B             % Inverted pendulum
%----------------------------------------------------------------
Variable   x''                % Distance between No to A
Variable   theta''            % Angle from local vertical to B's long axis
Specified  Fc                 % Control force on cart
Constant   g+ = 9.81 m/s^2    % Gravitational constant
Constant   L+ = 0.5  m        % Distance between A and Bcm
A.SetMass( mA = 10 kg )
B.SetMassInertia( mB = 1 kg,  Izz = 1/12*mB*(2*L)^2,  0,  Izz )
%----------------------------------------------------------------
%       Rotational and translational kinematics
B.RotateNegativeZ( N, theta )
A.Translate( No, x*Nx> )
Bcm.Translate( A, L*By> )
%----------------------------------------------------------------
%       Relevant contact and distance forces
System.AddForceGravity( -g*Ny> )
A.AddForce( Fc*Nx> )
%----------------------------------------------------------------
%       Form and simplify equations of motion (via Newton/Euler)
EquationsOfMotion[1] = Dot(  Nx>,  A.GetDynamics() + B.GetDynamics() )
EquationsOfMotion[2] = Dot( -Bz>,  B.GetDynamics(A) )
FactorLinear( EquationsOfMotion,  theta'', x'', Fc, g )
%----------------------------------------------------------------
Save InvertedPendulumOnCartDynamics.all
```

9.13 3D spin stability (application to Top-gun and Explorer I)

```
%    File: SpinStability3DRigidBodyEuler.txt
% Purpose: Spin stability of rigid body (books, footballs, aircraft)
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%--------------------------------------------------------------------
%       Physical objects
NewtonianFrame N
RigidBody      B
%--------------------------------------------------------------------
%       Mathematical declarations
Variable   wx', wy', wz'          % Angular velocity measures
Constant   b = 0.0 N*m*sec/rad    % Viscous damping constant
B.SetMassInertia( m = 0.4 kg,  Ixx = 1 kg*m^2,  Iyy = 2 kg*m^2,  Izz = 3 kg*m^2 )
%--------------------------------------------------------------------
%       Rotational kinematics
B.SetAngularVelocityAcceleration( N,  wx*Bx> + wy*By> + wz*Bz> )
%--------------------------------------------------------------------
%       Add relevant torques
B.AddTorque( -b * B.GetAngularVelocity(N) )
%--------------------------------------------------------------------
%       Form equations of motion (angular momentum principle)
RotationalDynamics> = B.GetDynamics( Bcm )
RotationalEqn[1] = Dot( RotationalDynamics>, Bx> )
RotationalEqn[2] = Dot( RotationalDynamics>, By> )
RotationalEqn[3] = Dot( RotationalDynamics>, Bz> )
Solve( RotationalEqn,  wx',  wy',  wz'  )
%--------------------------------------------------------------------
%       Numerical integration parameters and initial values
Input  tFinal = 4 sec,  tStep = 0.02 sec,  absError = 1.0E-08
Input  wx  = 0.2 rad/sec,   wy = 7.0 rad/sec,  wz = 0.2 rad/sec
%--------------------------------------------------------------------
%       Form output quantities and solve ODEs.
H> = B.GetAngularMomentum( Bcm )
Hmag = GetMagnitude( H> )
Wmag = GetMagnitude( B.GetAngularVelocity(N) )
theta = acos( Dot( H>, B.GetAngularVelocity(N) ) / (Hmag * Wmag) )
OutputPlot  t sec,  wx rad/sec,  wy rad/sec,  wz rad/sec
OutputPlot  t sec,  theta deg,  Hmag kg*m^2/sec^2
ODE()  SpinStability3DRigidBodyEuler
%--------------------------------------------------------------------
Save  SpinStability3DRigidBodyEuler.all
Quit
```

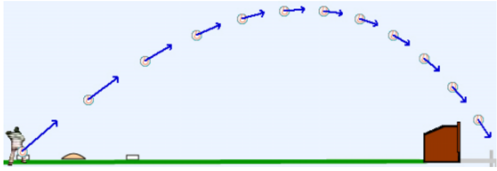






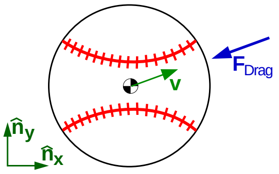
Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ 3D spin stability.

9.14 Kane's equations for projectile motion of a baseball



The figure to the right shows a baseball, modeled as a particle Q, in projectile motion over Earth N (a Newtonian reference frame). The aerodynamic forces on the baseball are modeled as $-b\vec{\mathbf{v}}$ where $\vec{\mathbf{v}}$ is Q's velocity in N.

Description	Symbol	Type	Value
Mass of baseball (5.1 ozm)	m	Constant	145 gram
Earth's gravitational acceleration (32.2 $\frac{\text{ft}}{\text{s}^2}$)	g	Constant	$9.8\ \frac{\text{m}}{\text{s}^2}$
Coefficient in drag force $-b\vec{\mathbf{v}}$ on baseball	b	Constant	$0.05\ \frac{\text{N s}}{\text{m}}$
$\widehat{\mathbf{n}}_\text{x}$ measure of Q's position from N_o	x	Variable	
$\widehat{\mathbf{n}}_\text{y}$ measure of Q's position from N_o	y	Variable	



$\widehat{\mathbf{n}}_\text{x}$ is horizontally-right, $\widehat{\mathbf{n}}_\text{y}$ is vertically-upward, and N_o is home-plate (point fixed in N).

1. Form ***Kane's equations*** for the ***generalized speeds*** $\dot{x}$ and $\dot{y}$.
   Result: (Also solve for $\ddot{x}$ and $\ddot{y}$).

   Kane's equation for generalized speed $\dot{x}$: $m\,\ddot{x} = \boxed{-b\,\dot{x}} \Rightarrow \ddot{x} = \boxed{\frac{-b}{m}\,\dot{x}}$

   Kane's equation for generalized speed $\dot{y}$: $m\,\ddot{y} = \boxed{-b\,\dot{y} - m\,g} \Rightarrow \ddot{y} = \boxed{-g - \frac{b}{m}\,\dot{y}}$

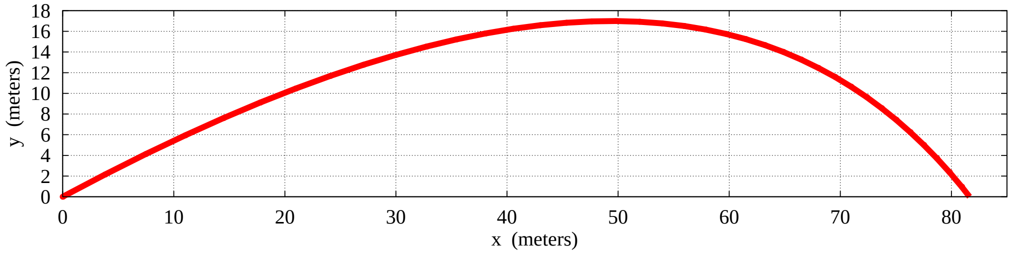
2. Form ***Lagrange's equations*** for the ***generalized coordinates*** x and y.
   Lagrange's and Kane's equations produce the same results (for this problem). **[✓ True]/False**.

3. This system has a kinetic energy **[✓ True]/False**.
   This system has a potential energy **True/[✓ False]** (all forces are conservative).

4. Express the work done by aerodynamic drag in terms of $\dot{x}$ and $\dot{y}$. Form an expression having units of energy that stays constant during the baseball's flight in N.
   Result:

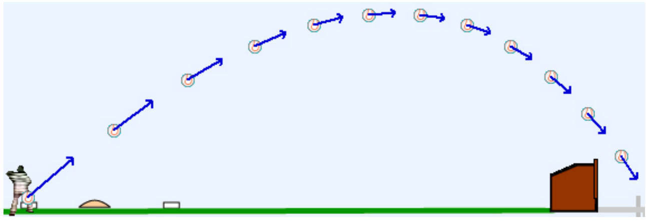
   $$\text{W} = -b \int (\boxed{\dot{x}^2} + \boxed{\dot{y}^2})\, dt \qquad \text{EnergyConstant} = K - \text{W} + \boxed{m\,g\,y}$$

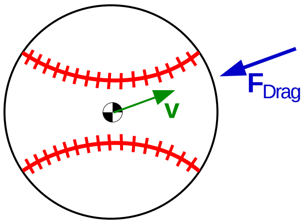
5. **Optional**:** Create an "x-y" plot of the baseball's trajectory when hit from home plate ($x{=}0,\ y{=}0$) with initial speed 44.7 $\frac{\text{m}}{\text{sec}}$ (100 mph) and initial angle of 30°. Ensure the expression for EnergyConstant is approximately constant (≈ to numerical integrator accuracy).
   Result:



Note: Problem solution at **<u>www.MotionGenesis.com</u>** ⇒ **<u>Get Started</u>** ⇒ **Projectile motion.**

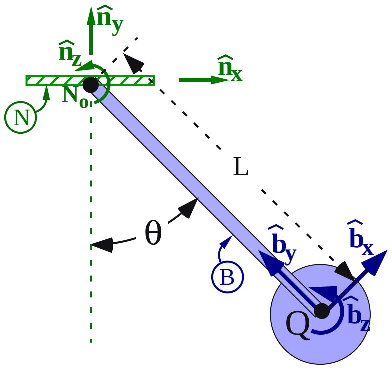
```
% File: MGProjectileMotionKane.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%---------------------------------------------------
%       Physical objects
NewtonianFrame N
Particle       Q
%---------------------------------------------------
%       Mathematical declarations
Constant   g = 9.8 m/s^2,  b = 0.05 N*s/m
Variable   x’’,  y’’
Q.SetMass( m = 145 grams )
%---------------------------------------------------
%       Translational kinematics
Q.Translate( No, x*Nx> + y*Ny> )
%---------------------------------------------------
%       Add relevant forces
Q.AddForce( -m * g * Ny> )
Q.AddForce( -b * Q.GetVelocity(N) )
%---------------------------------------------------
%       Form Kane’s equations of motion
SetGeneralizedSpeed( x’, y’ )
DynamicEquations = System.GetDynamicsKane()
Solve( DynamicEquations,  x’’,  y’’ )
%---------------------------------------------------
%       Solve nonlinear ODEs (numerically integrate)
Input  tFinal = 3.8 sec,  tStep = 0.1 sec,  absError = 1.0E-7
Input  x = 0 m,  x’ = 44.7*cosDegrees(30) m/s
Input  y = 0 m,  y’ = 44.7*sinDegrees(30) m/s
OutputPlot  x m, y m
ODE() MGProjectileMotionKane
%---------------------------------------------------
Save MGProjectileMotionKane.all
Quit
```





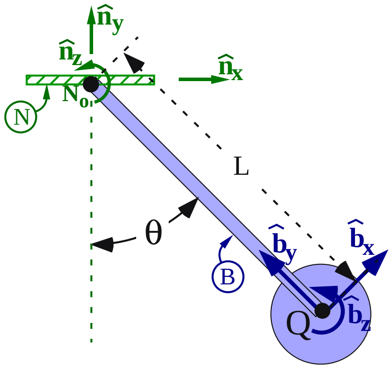
Note: Problem solution at <u>www.MotionGenesis.com</u> ⇒ <u>Get Started</u> ⇒ Projectile motion.

9.15 Kane's equations for the classic particle pendulum



```
%     File:  ClassicParticlePendulumKane.txt
%  Problem:  Equation of motion for pendulum
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%---------------------------------------------------------------
NewtonianFrame  N         % Newtonian reference frame (ground)
RigidFrame      B         % Massless inextensible (rigid) string
Particle        Q         % Particle at end of string
%---------------------------------------------------------------
Variable  theta''         % Pendulum angle
Constant  L = 50 cm       % Length of string
Constant  g = 9.8 m/s^2   % Earth's gravitational acceleration
Q.SetMass( m = 2 kg )
SetGeneralizedSpeed( theta' )
%---------------------------------------------------------------
%       Rotational and translational kinematics
B.RotateZ( N, theta )
Q.Translate( No, -L*By> )
%---------------------------------------------------------------
%       Relevant forces
Q.AddForceGravity( -g*Ny> )
%---------------------------------------------------------------
%       Equations of motion
Dynamics = System.GetDynamicsKane()
Solve( Dynamics, theta'' )
%---------------------------------------------------------------
%       Kinetic and potential energy
KE = System.GetKineticEnergy()
PE = Q.GetForceGravityPotentialEnergy( -g*Ny>, No )
MechanicalEnergy = KE + PE
%---------------------------------------------------------------
%   Integration parameters and initial values of variables.
Input  tFinal = 10 sec,  tStep = 0.02 sec,  absError = 1.0E-08
Input  theta = 30 deg,  theta' = 0 deg/sec
%---------------------------------------------------------------
%       List output quantities and solve ODEs.
Output  t sec,  theta deg,  theta' deg/sec,  KE Joules,  PE Joules,  MechanicalEnergy Joules
ODE()  ClassicParticlePendulumKane
Save   ClassicParticlePendulumKane.all
Quit
```

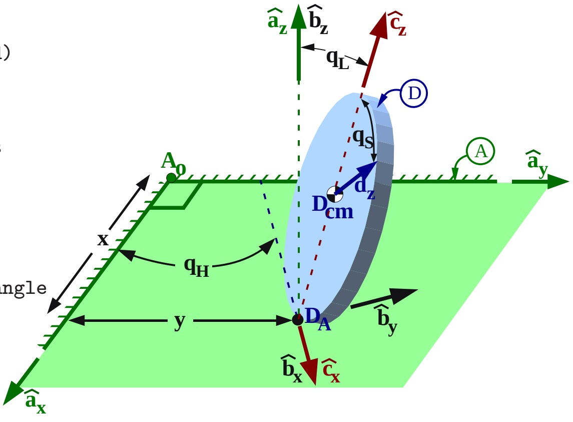
Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Pendulum.



```
   (1) %     File:  ClassicParticlePendulumKane.txt
   (2) %  Problem:  Equation of motion for pendulum
   (3) % Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
   (4) %--------------------------------------------------------------
   (5) NewtonianFrame  N         % Newtonian reference frame (ground)
   (6) RigidFrame      B         % Massless inextensible (rigid) string
   (7) Particle        Q         % Particle at end of string
   (8) %--------------------------------------------------------------
   (9) Variable  theta''         % Pendulum angle
   (10) Constant  L = 50 cm       % Length of string
   (11) Constant  g = 9.8 m/s^2   % Earth's gravitational acceleration
   (12) Q.SetMass( m = 2 kg )
   (13) SetGeneralizedSpeed( theta' )
   (14) %--------------------------------------------------------------
   (15) %       Rotational and translational kinematics
   (16) B.RotateZ( N, theta )
-> (17) B_N = [cos(theta), sin(theta), 0; -sin(theta), cos(theta), 0; 0, 0, 1]
-> (18) w_B_N> = theta'*Bz>
-> (19) alf_B_N> = theta''*Bz>

   (20) Q.Translate( No, -L*By> )
-> (21) p_No_Q> = -L*By>
-> (22) v_Q_N> = L*theta'*Bx>
-> (23) a_Q_N> = L*theta''*Bx> + L*theta'^2*By>

   (24) %--------------------------------------------------------------
   (25) %       Relevant forces
   (26) Q.AddForceGravity( -g*Ny> )
-> (27) Force_Q> = -g*m*Ny>

   (28) %--------------------------------------------------------------
   (29) %       Equations of motion
   (30) Dynamics = System.GetDynamicsKane()
-> (31) Dynamics = [L*m*(g*sin(theta)+L*theta'')]

   (32) Solve( Dynamics, theta'' )
-> (33) theta'' = -g*sin(theta)/L

   (34) %--------------------------------------------------------------
   (35) %       Kinetic and potential energy
   (36) KE = System.GetKineticEnergy()
-> (37) KE = 0.5*m*L^2*theta'^2

   (38) PE = Q.GetForceGravityPotentialEnergy( -g*Ny>, No )
-> (39) PE = -g*L*m*cos(theta)

   (40) MechanicalEnergy = KE + PE
-> (41) MechanicalEnergy = PE + KE

   (42) %--------------------------------------------------------------
   (43) %   Integration parameters and initial values of variables.
   (44) Input  tFinal = 10 sec,  tStep = 0.02 sec,  absError = 1.0E-08
   (45) Input  theta = 30 deg,  theta' = 0 deg/sec
   (46) %--------------------------------------------------------------
   (47) %       List output quantities and solve ODEs.
   (48) Output  t sec,  theta deg,  theta' deg/sec,  KE Joules,  PE Joules,  MechanicalEnergy Joules
   (49) ODE()  ClassicParticlePendulumKane
```

Note: Problem solution at **www.MotionGenesis.com** ⇒ **Get Started** ⇒ **Pendulum**.

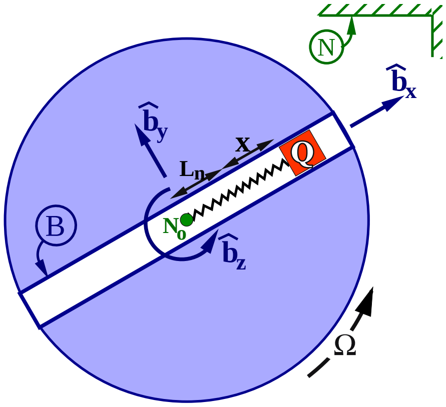
9.16 Dynamics of a rolling disk



```
%    File:  MGRollingDiskKane.txt
% Problem:  Dynamics and simulation of a rolling disk. (Kane’s method)
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%-------------------------------------------------------------------
NewtonianFrame  A                 % Horizontal plane
RigidFrame      B, C              % Heading and tilt reference frames
RigidBody       D                 % Rolling disk
Point           DA( D )           % Point of D in contact with A
%-------------------------------------------------------------------
Variable  wx’,  wy’,  wz’         % Angular velocity of D in A
Variable  x’’,  y’’               % Locates contact point DA from No
Variable  qH’,  qL’,  qS’         % Heading angle, lean angle, spin angle
Constant  r = 0.343 meters        % Radius of disk (13.5 inches)
Constant  g = 9.8   m/s^2         % Earth’s local gravity
Constant  b = 0 N*m*s/rad         % Aerodynamic damping on disk
D.SetMass( m = 2 kg )
D.SetInertia( DCm,  C,  J = 0.25*m*r^2,  I = 2*J,  J )
SetGeneralizedSpeed( wx, wy, wz )
%--------------------------------------------------------------
%       Rotation matrices and angular velocity
B.RotatePositiveZ( A, qH )
C.RotateNegativeX( B, qL )
D.RotatePositiveY( C, qS )
%--------------------------------------------------------------
%       For efficient dynamics, make a change of variable.
wDA> = D.GetAngularVelocity( A )
ChangeVariables[1] = wx - Dot( wDA>, Cx> )
ChangeVariables[2] = wy - Dot( wDA>, Cy> )
ChangeVariables[3] = wz - Dot( wDA>, Cz> )
Solve( ChangeVariables, qH’, qL’, qS’ )
%--------------------------------------------------------------
%       Use the simpler version of angular velocity from here on.
D.SetAngularVelocity( A, wx*Cx> + wy*Cy> + wz*Cz> )
%-------------------------------------------------------------------
%       Form angular acceleration (differentiating in D is more efficient)
alf_D_A> = Dt( D.GetAngularVelocity(A), D )
%--------------------------------------------------------------
%       Position vectors, velocity, and acceleration
Dcm.Translate( Ao,  x*Ax> + y*Ay> + r*Cz> )
DA.SetPositionVelocity( Dcm, -r*Cz> )
%--------------------------------------------------------------
%       Motion constraints (D rolls on A at point DA).
RollingConstraint[1] = Dot( DA.GetVelocity(A), Ax> )
RollingConstraint[2] = Dot( DA.GetVelocity(A), Ay> )
SolveDt( RollingConstraint, x’, y’ )
%--------------------------------------------------------------
%       Add relevant forces.
Dcm.AddForce( -m*g*Az> )
%--------------------------------------------------------------
%       Kane’s equations of motion with simplification
Dynamics = System.GetDynamicsKane()
FactorQuadratic( Dynamics, wx, wy, wz )
%--------------------------------------------------------------
%       Solve for wx’, wy’, wz’ (independent variables)
Solve( Dynamics, wx’, wy’, wz’ )
%--------------------------------------------------------------
%       Integration parameters and initial values for variables.
Input  x = 0 m,  y = 0 m,  qH = 0 deg,  qL = 10 deg,  qS = 0 deg
Input  wx = 0 rad/sec,  wy = 5 rad/sec,  wz = 0 rad/sec
Input  tFinal = 12 sec, tStep = 0.05 sec,  absError = 1.0E-08
%--------------------------------------------------------------
%       List output quantities and solve ODEs (or write MATLAB, C, ... code).
OutputPlot  t sec,  x meters,  y meters,  qL degrees,  qH degrees
ODE() MGRollingDiskKane
%--------------------------------------------------------------
%       Record input together with responses
Save MGRollingDiskKane.all
Quit
```

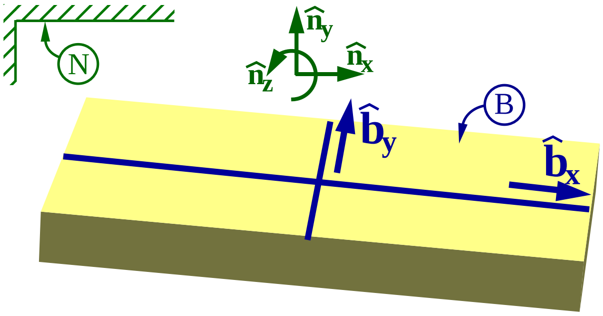
Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Rolling disk.

9.17 Kane's equations for a particle in a horizontal slot



```
% File: ParticleOnSpinningSlotKane.al
%-----------------------------------------------------------
%       Physical objects
NewtonianFrame  N
RigidBody       B
Particle        Q
%-----------------------------------------------------------
%       Mathematical declarations
Variable   x''     % Bx> measure of Q's position from No
Constant   b       % Linear damping constant
Constant   k       % Linear spring constant
Constant   Ln      % Natural length of spring
Specified  Omega'  % Bz> measure of B's angular velocity in N
Q.SetMass( m )
%-----------------------------------------------------------
%       Rotational kinematics
B.SetAngularVelocityAcceleration( N, Omega*Bz> )
%-----------------------------------------------------------
%       Translational kinematics
Bcm.SetVelocity( N, 0> )
Q.Translate( No, (Ln+x)*Bx> )
%-----------------------------------------------------------
%       Add relevant forces
Q.AddForce( -(k*x + b*x')*Bx> )
%-----------------------------------------------------------
%       Form equation of motion
SetGeneralizedSpeed( x' )
Zero = System.GetDynamicsKane()
Solve( Zero, x'' )
%-----------------------------------------------------------
%       Record input together with responses
Save ParticleOnSpinningSlotKane.all
Quit
```

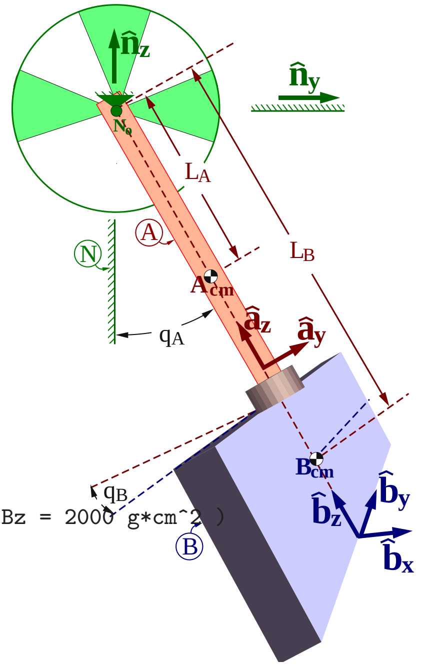
9.18 Lagrange's equation for rigid body rotation



```
%    File: SpinningBookLagrange.al
% Purpose: Compare with other methods for forming EOM
%-----------------------------------------------------
%       Physical objects
NewtonianFrame  N            % Deep space
RigidBody       B            % Rotating rigid body
%-----------------------------------------------------
%       Mathematical declarations
Variable  qx'', qy'', qz''   % Euler angles, derivativ
Constant  hx, hy, hz         % Dimensions of book
Constant  b                  % Damping constant
B.SetMass( m )
B.SetInertia( Bcm, Ixx, Iyy, Izz )
Ixx = m*(hy^2+hz^2)/12
Iyy = m*(hx^2+hz^2)/12
Izz = m*(hx^2+hy^2)/12
%-----------------------------------------------------
%       Rotational kinematics
B.Rotate( N, BodyXYZ, qx, qy, qz )
%-----------------------------------------------------
%       Translational kinematics
Bcm.SetVelocity( N, 0> )     % Bcm is motionless in N
%-----------------------------------------------------
%       Add torque on B due to viscous damping
B.AddTorque( -b * B.GetAngularVelocity(N) )
%-----------------------------------------------------
%       Form and solve equations of motion
SetGeneralizedSpeed( qx', qy', qz' )
Zero = System.GetDynamicsKane()
Solve( Zero, qx'', qy'', qz'' )
%-----------------------------------------------------
%       Save input together with responses
Save SpinningBookLagrangeShort.all
Quit
```

Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ 3D spin stability.

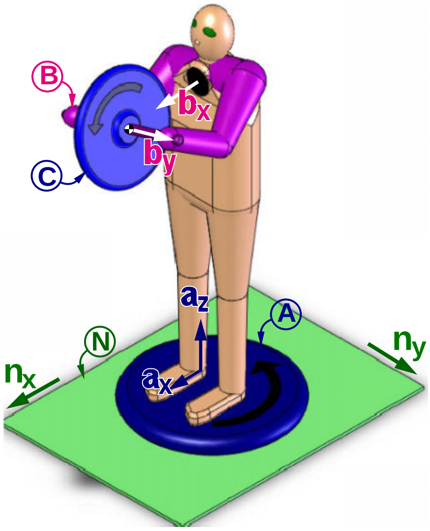
9.19 Kane’s equations for a chaotic 3D pendulum



```
%    File: BabybootWithKanesMethod.txt
% Problem: Analysis of 3D chaotic double pendulum
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%--------------------------------------------------------------------
SetDigits( 5 )     % Number of digits displayed for numbers
%--------------------------------------------------------------------
%       Physical objects
NewtonianFrame  N
RigidBody  A       % Upper rod
RigidBody  B       % Lower plate
%---------------------------------------------------------------
%       Mathematical declarations
Variable   qA’’         % Pendulum angle and its time-derivatives
Variable   qB’’         % Plate angle and its time-derivative
Constant   LA = 7.5 cm  % Distance from pivot to A’s mass center
Constant   LB = 20 cm   % Distance from pivot to B’s mass center
A.SetMassInertia( mA =  10 grams,  IAx = 50 g*cm^2,  IAy,  IAz )
B.SetMassInertia( mB = 100 grams,  IBx = 2500 g*cm^2, IBy = 500 g*cm^2, IBz = 2000 g*cm^2 )
%---------------------------------------------------------------
%       Rotational kinematics
A.RotateX( N, qA )
B.RotateZ( A, qB )
%---------------------------------------------------------------
%       Translational kinematics
Acm.Translate( No, -LA*Az> )
Bcm.Translate( No, -LB*Az> )
%---------------------------------------------------------------
%       Add relevant forces
g> = -9.81*Nz>
System.AddForceGravity( g> )
%---------------------------------------------------------------
%       Kane’s equations of motion (uses generalized speeds)
SetGeneralizedSpeed( qA’, qB’ )
Dynamics = System.GetDynamicsKane()
Solve( Dynamics, qA’’, qB’’ )
%---------------------------------------------------------------
%       Kinetic and potential energy
KE = System.GetKineticEnergy()
PE = System.GetForceGravityPotentialEnergy( g>, No )
Energy = KE + PE
%---------------------------------------------------------------
%       Numerical integration parameters and initial values.
Input  tFinal=10,  tStep=0.02,  absError=1.0E-07,  relError=1.0E-07
Input  qA = 90 deg,  qA’ = 0.0 rad/sec,  qB = 1.0 deg,  qB’ = 0.0 rad/sec
%---------------------------------------------------------------
%       List output quantities and solve ODEs.
OutputPlot  t sec,  qA deg,  qB deg,  Energy N*m
ODE()  Babyboot
%----------------------------------------------------------------
%       Record input together with responses
Save BabybootWithKanesMethod.all
Quit
```

Note: Problem solution at **www.MotionGenesis.com** ⇒ **Get Started** ⇒ **Chaotic pendulum**.

9.20 Kane's equations for a gyro on a turntable

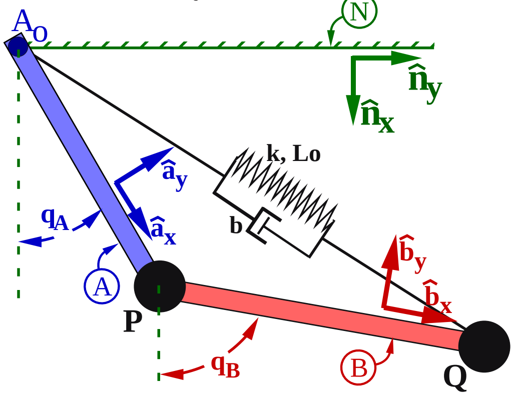


```
% File: HumanOnTurntableWithGyroKaneLagrange.al
%------------------------------------------------------------
%       Physical objects
NewtonianFrame N
RigidBody      A   % Turntable, human legs, torso, and head.
RigidFrame     B   % Human shoulder, arms, and wheel's axle.
RigidBody      C   % Bicycle wheel (rotor).
%------------------------------------------------------------
%       Mathematical declarations
Variable    qA''   % Az> measure of angle from Nx> to Ax>
Specified   qB''   % Ax> measure of angle from Ay> to By>
Variable    wC'    % By> measure of C's angular velocity in B
Constant    Lz     % Az> measure of Bo from No
Constant    Lx     % Ax> measure of Ccm from Bo
C.SetMass( mC )
A.SetInertia( Acm, IAxx, IAyy, IAzz )
C.SetInertia( Ccm, B, IC, JC, IC )
SetGeneralizedSpeed( qA', wC )
%------------------------------------------------------------
%       Rotational kinematics
A.RotateZ( N, qA )
B.RotateX( A, qB )
C.SetAngularVelocityAcceleration( B, wC*By> )
%------------------------------------------------------------
%       Translational kinematics
Acm.SetVelocity( N, 0> )
Bo.Translate(  No, Lz*Az> )
Ccm.Translate( Bo, Lx*Ax> )
%------------------------------------------------------------
%       Form Kane's equation of motion.
ZeroKane = System.GetDynamicsKane()
FactorQuadratic( ZeroKane, qA', qB', wC )
Solve( ZeroKane, qA'', wC' )
%------------------------------------------------------------
%       Record input together with responses
Save HumanOnTurntableWithGyroKaneLagrange.all
Quit
```

9.21 Spring-damper double pendulum: Forces and motion

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 A_o and P, respectively. Right-handed sets of orthogonal unit vectors $\widehat{\mathbf{n}}_\mathrm{i}$, $\widehat{\mathbf{a}}_\mathrm{i}$, $\widehat{\mathbf{b}}_\mathrm{i}$ $(i = \mathrm{x,y,z})$ are fixed in N, A, and B, with $\widehat{\mathbf{n}}_\mathrm{x}$ vertically downward, $\widehat{\mathbf{a}}_\mathrm{x}$ directed from A_o to P, $\widehat{\mathbf{b}}_\mathrm{x}$ directed from P to Q, and $\widehat{\mathbf{n}}_\mathrm{z} = \widehat{\mathbf{a}}_\mathrm{z} = \widehat{\mathbf{b}}_\mathrm{z}$ parallel to the axes of the revolute joints.

Quantity	Symbol	Value
Mass of P	m^P	10 kg
Mass of Q	m^Q	20 kg
Earth's gravitational constant	g	$9.8\ \frac{\mathrm{m}}{\mathrm{s}^2}$
Distance from A_o to P	L_A	1 m
Distance from P to Q	L_B	2 m
Spring's natural length	L_o	1 m
Linear spring constant	k	$200\ \frac{\mathrm{N}}{\mathrm{m}}$
Linear damping constant (force)	b	$100\ \frac{\mathrm{N*s}}{\mathrm{m}}$
Linear damping constant (torques)	c	$100\ \frac{\mathrm{N\,m\,s}}{\mathrm{rad}}$
Angle from $\widehat{\mathbf{n}}_\mathrm{x}$ to $\widehat{\mathbf{a}}_\mathrm{x}$ with $+\widehat{\mathbf{n}}_\mathrm{z}$ sense	q_A	Variable
Angle from $\widehat{\mathbf{n}}_\mathrm{x}$ to $\widehat{\mathbf{b}}_\mathrm{x}$ with $+\widehat{\mathbf{n}}_\mathrm{z}$ sense	q_B	Variable



- Using ***potential energy*** for gravity and the spring, form expressions for the system's generalized forces $\mathcal{F}_{\dot q_A}$ and $\mathcal{F}_{\dot q_B}$ (when damping has stopped) and form equations governing static equilibrium.

Optional:** Determine four static solutions for q_A and q_B between -180° and 180°.

Result: (Using intuition/guessing, circle the **stable** solutions).

Static solutions

#	q_A	q_B
1	0°	0°
$\boxed{2}$	-50.2°	35.2°
$\boxed{3}$	50.2°	-35.2°
4	180°	180°

$$L_\text{Spring} = \sqrt{L_A^2 + L_B^2 + 2\,L_A\,L_B\,\cos(q_A - q_B)} \qquad \text{Stretch} = L_\text{Spring} - L_o$$

$$L_A \left[L_B\,k\,\text{Stretch}\,\frac{\sin(q_A - q_B)}{L_\text{Spring}} - g\,(m^P + m^Q)\,\sin(q_A) \right] = 0$$

$$L_B \left[\text{-}L_A\,k\,\text{Stretch}\,\frac{\sin(q_A - q_B)}{L_\text{Spring}} - g\,(m^Q\,\sin(q_B) \right] = 0$$

- **Optional**:** Form the system's generalized forces $\mathcal{F}_{\dot q_A}$ and $\mathcal{F}_{\dot q_B}$. Form ***Kane's equations*** for ***generalized speeds*** $\dot q_A$, $\dot q_B$ or ***Lagrange's equations*** for ***generalized coordinates*** q_A, q_B.

Note: Damping torque from air of $\text{-}c\,\dot q_A$ and $\text{-}c\,\dot q_B$ act on A and B, respectively.

Result:

$$\text{Stretch'} = \text{-}L_A\,L_B \sin(q_A - q_B)\,(\dot q_A - \dot q_B) \,/\, L_\text{Spring}$$

$$\mathcal{F}_{\dot q_A} = L_A \left[L_B\,(k\,\text{Stretch} + b\,\text{Stretch'})\,\frac{\sin(q_A - q_B)}{L_\text{Spring}} - g\,(m^P + m^Q)\,\sin(q_A) \right] - c\,\dot q_A$$

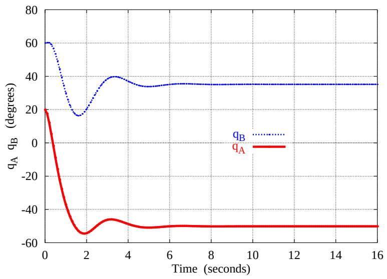
$$\mathcal{F}_{\dot q_B} = L_B \left[\text{-}L_A\,(k\,\text{Stretch} + b\,\text{Stretch'})\,\frac{\sin(q_A - q_B)}{L_\text{Spring}} - g\,(m^Q\,\sin(q_B) \right] - c\,\dot q_B$$

$$0 = L_A^2\,(m^P + m^Q)\,\ddot q_A + L_A\,L_B\,m^Q\,\cos(q_A - q_B)\,\ddot q_B + L_A\,L_B\,m^Q\,\sin(q_A - q_B)\,\dot q_B{}^2 - \mathcal{F}_{\dot q_A}$$

$$0 = L_A\,L_B\,m^Q\,\cos(q_A - q_B)\,\ddot q_A + m^Q\,L_B^2\,\ddot q_B - L_A\,L_B\,m^Q\,\sin(q_A - q_B)\,\dot q_A{}^2 - \mathcal{F}_{\dot q_B}$$

- **Optional**:** Plot q_A and q_B for $0 \le t \le 16$ sec when the system is initially released from ***rest*** with $q_A = 20°$ and $q_B = 60°$. Verify $q_A(t{=}16)$ and $q_B(t{=}16)$ approximate the static solutions.

Result: $q_A(t{=}16) \approx \text{-}50.2°$ $\qquad$ $q_B(t{=}16) \approx 35.2°$

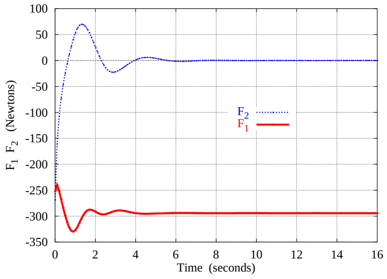


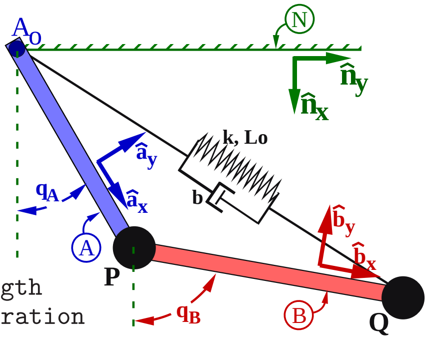
- **Optional**:** Form a numerical integration energy checking function and verify it remains approximately constant.

Note: Problem solution at **www.MotionGenesis.com** ⇒ **Get Started** ⇒ **Pendulums**.

The set of contact forces exerted by N on A across the revolute joint at $A_{\rm o}$ is equivalent to a couple of torque $T_x\,\widehat{\mathbf{n}}_{\rm x} + T_y\,\widehat{\mathbf{n}}_{\rm y}$ together with a force $F_x\,\widehat{\mathbf{n}}_{\rm x} + F_y\,\widehat{\mathbf{n}}_{\rm y} + F_z\,\widehat{\mathbf{n}}_{\rm z}$ applied at $A_{\rm o}$. To plot F_x and F_y for $0 \le t \le 16$ sec, modify the file `MGSpringRestrainedDoublePendulumDynamics.txt` as follows:

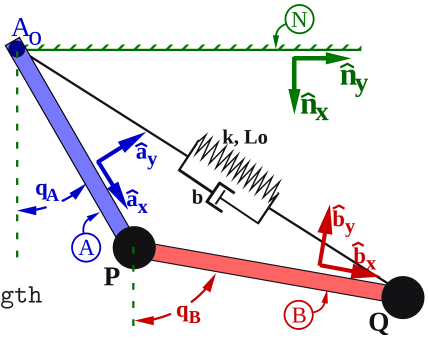
- Augment generalized speeds with `vx` and `vy` as
  `SetGeneralizedSpeed( qA', qB', vx, vy )`
- Set the actual values of the generalized speeds with:
  `SetDt( vx = 0 );      SetDt( vy = 0 )`
- Change the velocity of point $A_{\rm o}$ in N to
  `vx*Nx> + vy*Ny>`
- Augment the **ODE** and **Output** commands for `Fx` and `Fy`.





```
% File: MGSpringRestrainedDoublePendulumStatics.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%----------------------------------------------------------
NewtonianFrame  N
RigidFrame      A, B                 % Rods
Particle        P, Q                 % Particles at end of A, B
%----------------------------------------------------------
Variable   qA',  qB'                 % Angles for A and B
Constant   LA = 1 m,   LB = 2 m      % Lengths of A, B
Constant   k = 200 N/m,    Ln = 1 m  % Spring constant, natural length
Constant   g = 9.8 m/s^2             % Earth's gravitational acceleration
P.SetMass( mP = 10 kg )
Q.SetMass( mQ = 20 kg )
%-----------------------------------------------------------
%       Rotational and translational kinematics
A.RotateZ( N, qA )
B.RotateZ( N, qB )
P.SetPositionVelocity( No, LA*Ax> )
Q.SetPositionVelocity(  P, LB*Bx> )
%-----------------------------------------------------------
%       Relevant forces
System.AddForceGravity( g*Nx> )
LSpring = Q.GetDistance( No )
SpringStretch = LSpring - Ln
UnitVectorFromNoToQ> = Q.GetPosition( No ) / LSpring
Q.AddForce( No,  -k * SpringStretch * UnitVectorFromNoToQ> )
%-----------------------------------------------------------
%       Kane's equations for static equilibrium.
SetGeneralizedSpeed( qA', qB' )
StaticsKane = System.GetStaticsKane()
%-----------------------------------------------------------
%       Alternately, use potential energy U.
U = 1/2*k*SpringStretch^2 + System.GetForceGravityPotentialEnergy( g*Nx>, No )
Statics[1] = -D( U, qA )
Statics[2] = -D( U, qB )
%-----------------------------------------------------------
%       Solve nonlinear equations (requires a guess).
Solve( Statics,   qA = -30 deg,   qB = 30 deg )
%-----------------------------------------------------------
Save MGSpringRestrainedDoublePendulumStatics.all
Quit
```

Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Pendulums.



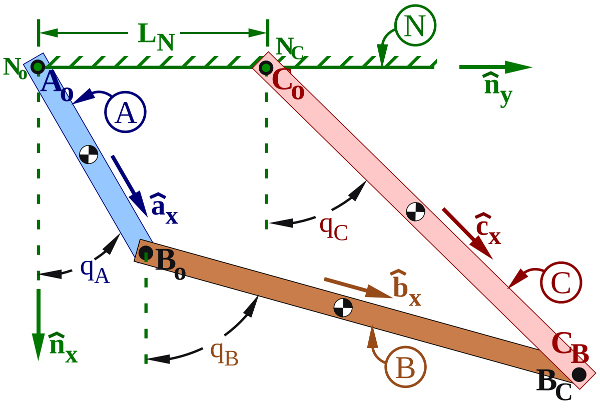
```
% File: MGSpringRestrainedDoublePendulumDynamics.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%---------------------------------------------------------
NewtonianFrame  N
RigidFrame      A, B                % Rods
Particle        P, Q                % Particles at end of A, B
%---------------------------------------------------------
Variable   qA'',  qB''              % Angles for A and B
Constant   LA = 1 m,  LB = 2 m      % Lengths of A, B
Constant   k = 200 N/m,   Ln = 1 m  % Spring constant, natural length
Constant   b = 100 N*s/m            % Damping constant
Constant   c = 100 N*m*s/rad        % Damping constant
Constant   g = 9.8 m/s^2            % Earth's gravitational acceleration
P.SetMass( mP = 10 kg )
Q.SetMass( mQ = 20 kg )
Variable   Fx, Fy                   % Contact forces on Ao from No
Specified  Stretch'                 % Elongation of spring between O and Q
%--------------------------------------------------------------------
Variable   vx', vy'                 % Auxiliary generalized speeds to calculate Fx and Fy.
SetGeneralizedSpeed( qA', qB', vx, vy )
SetDt( vx = 0 );    SetDt( vy = 0 )
%--------------------------------------------------------------------
%       Rotational and translational kinematics
A.RotateZ( N, qA )
B.RotateZ( N, qB )
Ao.SetVelocityAcceleration( N,  vx*Nx> + vy*Ny> )
P.Translate( Ao, LA*Ax> )
Q.Translate(  P, LB*Bx> )
%--------------------------------------------------------------------
%       Relevant forces for statics (gravity, spring, contact forces).
System.AddForceGravity( g*Nx> )
LSpring = Q.GetDistance( Ao )                    % Distance between Q and Ao
SetNoDt( Stretch = LSpring - Ln )                % Calculate spring stretch
UnitVectorFromAoToQ> = Q.GetPosition( Ao ) / LSpring
Q.AddForce( Ao,  -k * Stretch * UnitVectorFromAoToQ> )
Ao.AddForce( No,  Fx*Nx> + Fy*Ny> )              % Contact force on A from N
%--------------------------------------------------------------------
%       Damping force and torques.
Stretch' = Dot( Q.GetVelocity(N), UnitVectorFromAoToQ> )
Q.AddForce( Ao,  -b * Stretch' * UnitVectorFromAoToQ> )
A.AddTorque( -c * qA' * Az> )
B.AddTorque( -c * qB' * Bz> )
%--------------------------------------------------------------------
%       Equations of motion
Zero = System.GetDynamicsKane()
%--------------------------------------------------------------------
%       Integration parameters and initial values.
Input  tFinal = 16,  tStep = 0.1,  absError = 1.0E-07
Input  qA = 20 deg,  qB = 60 deg,  qA' = 0 rad/sec,  qB' =0 rad/sec
%--------------------------------------------------------------------
%       List output quantities and solve ODEs.
EnergyCheck = System.GetEnergyCheckKane()
OutputPlot  t sec,  qA deg,  qB deg,  Fx Newtons, Fy Newtons,  EnergyCheck Joules
ODE( Zero,  qA'', qB'', Fx, Fy )  MGSpringRestrainedDoublePendulumDynamics
%--------------------------------------------------------------------
Save MGSpringRestrainedDoublePendulumDynamics.all
Quit
```

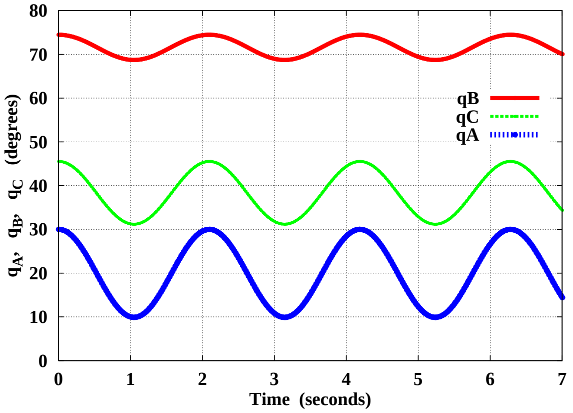
Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Pendulums.

9.22 Four-bar linkage: Motion and contact forces (see statics in Section 8.3)

Form dynamic equations for the following planar four-bar linkage using either ***Kane's equation*** for the ***generalized speed*** $\dot{q}_A$ and/or ***Lagrange's equations*** for the ***generalized coordinate*** q_A.[1]

Using initial values $q_A = 30°$ and $\dot{q}_A = 0$, plot q_A, q_B, q_C for $0 \le t \le 7$ sec. Note: $q_B = 74.47751219°$, $q_C = 45.52248781°$ satisfy the "***loop equation***" when $q_A = 30°$.

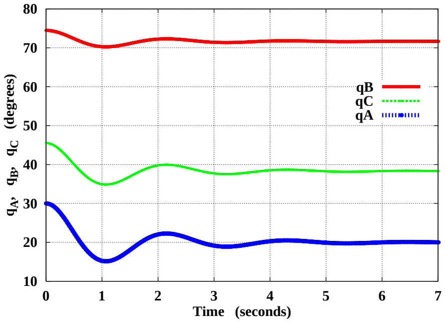


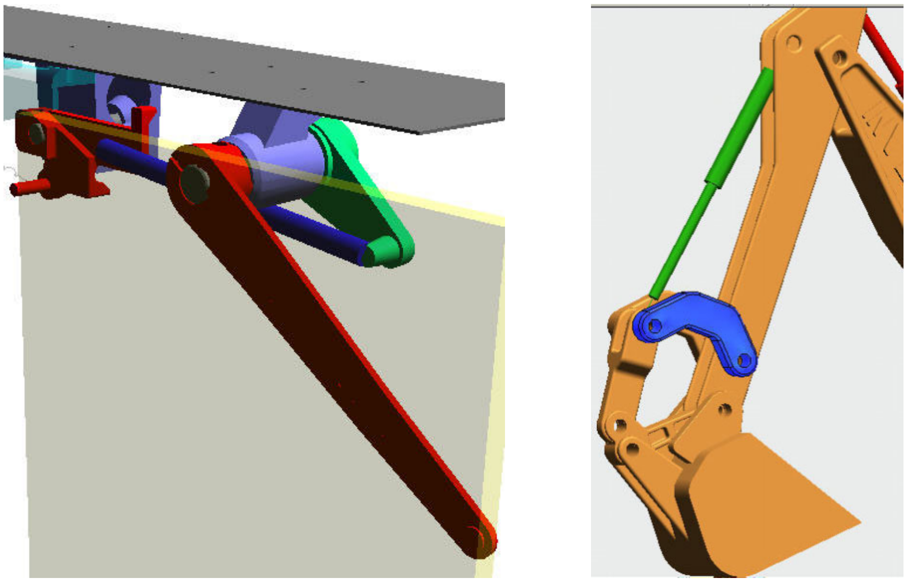


Statics by dynamics with damping.

One way to find a <u>**stable static**</u> equilibrium solution is to simulate the dynamic system with damping (e.g., $H = 200 - 80\,\dot{q}_C$) until the system settles (stops moving). Determine q_A, q_B, q_C when the system stops moving.

Result: $q_A = 20.0°$ $q_B = 71.7°$ $q_C = 38.3°$





Examples of 4-bar linkages: Courtesy Design Simulation Technology (SimWise)

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

Result: After running the file `MGFourBarForcesAndMotion.txt` in MotionGenesis, the file `MGFourBarForcesAndMotion.1` shows time-histories for `Loop[1]`, `Loop[2]`, and `ECheck`.

---

[1] **Note: Problem solution at <u>www.MotionGenesis.com</u> ⇒ <u>Get Started</u> ⇒ Four-bar linkage.**

The set of contact forces exerted by N on A across the revolute joint at $A_{\rm o}$ is equivalent to a couple of torque $\mathrm{T}_x^A\,\widehat{\mathbf{n}}_{\rm x} + \mathrm{T}_y^A\,\widehat{\mathbf{n}}_{\rm y}$ together with a force $F_x^A\,\widehat{\mathbf{n}}_{\rm x} + F_y^A\,\widehat{\mathbf{n}}_{\rm y} + F_z^A\,\widehat{\mathbf{n}}_{\rm z}$ applied at $A_{\rm o}$.
The set of contact forces exerted by B on C across the revolute joint at C_B is equivalent to a couple of torque $\mathrm{T}_x^C\,\widehat{\mathbf{n}}_{\rm x} + \mathrm{T}_y^C\,\widehat{\mathbf{n}}_{\rm y}$ together with a force $F_x^C\,\widehat{\mathbf{n}}_{\rm x} + F_y^C\,\widehat{\mathbf{n}}_{\rm y} + F_z^C\,\widehat{\mathbf{n}}_{\rm z}$ applied at C_B.

- Plot the time histories of F_x^C and F_y^C for $0 \le t \le 10$ sec.

To find F_x^C and F_y^C, the file `MGFourBarForcesAndMotion.txt` was modified as follows:

Change these lines	**To these lines**

`SetGeneralizedSpeed( qA' )`
`ODE( Zero,  qA'' )`
	`SetGeneralizedSpeed( qA', qB', qC' )`
	`ODE( Zero,  qA'', FCx, FCy )`

Uncomment/add these lines	`CB.AddForce( BC,  FCx*Nx> + FCy*Ny> )`
	`Output  t sec,  FCx N,  FCy N`

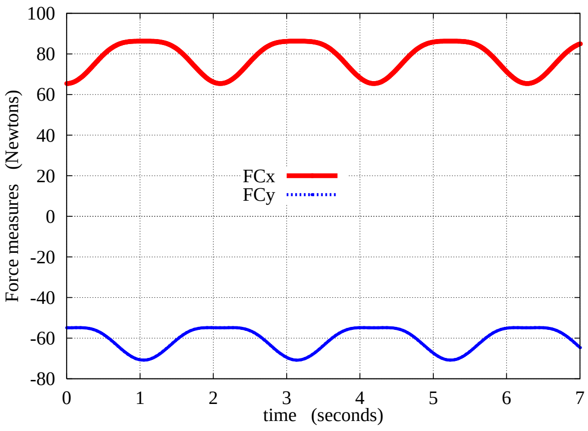
- Plot the time-history of F_x^A for $0 \le t \le 10$ sec.

To find F_x^A, again modify the file `MGFourBarForcesAndMotion.txt`.

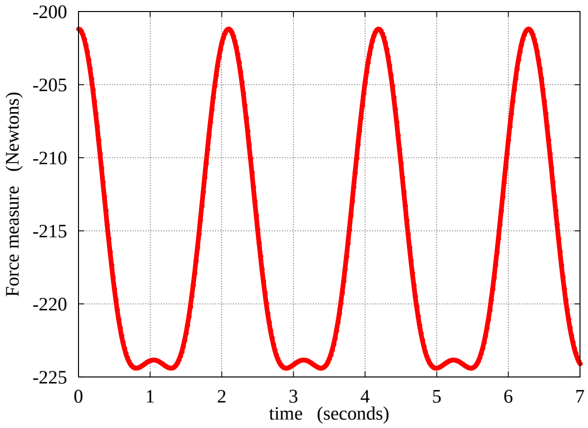
Change these lines	**To these lines**
`Variable   qA'',  qB'',  qC''`	`Variable   qA'',  qB'',  qC'', vx'`

`SetGeneralizedSpeed( qA', qB', qC' )`
`Ao.SetVelocityAcceleration( N, 0> )`
`Solve( Zero,  qA'', FCx, FCy )`
`Output  t sec,  FCx N,  FCy N`
	`SetGeneralizedSpeed( qA', qB', qC', vx )`
	`Ao.SetVelocityAcceleration( N, vx*Nx> )`
	`Solve( Zero,  qA'', FCx, FCy, FAx )`
	`Output  t sec,  FCx N,  FCy N,  FAx N`

Uncomment/add these lines	`SetDt( vx = 0 )`

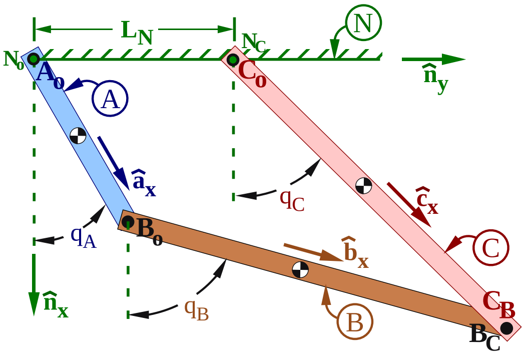


Time-histories of F_x^C and F_y^C



Time-history of F_x^A

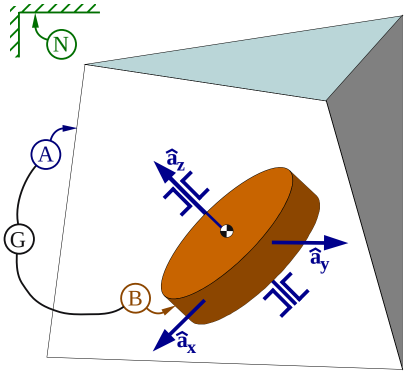
Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Four-bar linkage.



```
% File: MGFourBarForcesAndMotion.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%---------------------------------------------------------------
%       Physical objects
NewtonianFrame  N
RigidBody       A, B, C
Point           BC(B), CB(C)
%---------------------------------------------------------------
%       Mathematical declarations
Constant   LA = 1 m,  LB = 2 m,  LC = 2 m,  LN = 1 m
Constant   g = 9.81 m/s^2              % Gravity
Specified  H = 200                     % Horizontal force
Variable   qA'',  qB'',  qC''          % , vx'
Variable   FAx, FAy,  FCx, FCy         % Contact forces
SetGeneralizedSpeed( qA', qB', qC' )   % , vx
SetAutoZee( ON )                       % Efficient calculations.
SetNoZeeSymbol( FAx, FAy, FCx, FCy )
%---------------------------------------------------------------
A.SetMassInertia( mA = 10 kg,  0,  IA = mA*LA^2/12,  IA  )
B.SetMassInertia( mB = 20 kg,  0,  IB = mB*LB^2/12,  IB  )
C.SetMassInertia( mC = 20 kg,  0,  IC = mC*LC^2/12,  IC  )
%---------------------------------------------------------------
%       Rotational kinematics
A.RotateZ( N, qA )
B.RotateZ( N, qB )
C.RotateZ( N, qC )
%---------------------------------------------------------------
%       Translational kinematics
Ao.SetVelocityAcceleration( N, 0> )    % vx*Nx>
Acm.Translate(  Ao,  0.5*LA*Ax> )
Bo.Translate(   Ao,      LA*Ax> )
Bcm.Translate(  Bo,  0.5*LB*Bx> )
BC.Translate(   Bo,      LB*Bx> )
Co.Translate(   No,      LN*Ny> )
Ccm.Translate(  Co,  0.5*LC*Cx> )
CB.Translate(   Co,      LC*Cx> )
%---------------------------------------------------------------
%       Forces
System.AddForceGravity( g*Nx> )
CB.AddForce( H*Ny> )
Ao.AddForce( FAx*Nx> + FAy*Ny> )
CB.AddForce( BC,  FCx*Nx> + FCy*Ny> )   % "Cut" linkage at CB/BC
%---------------------------------------------------------------
%       Configuration constraints and time-derivatives
Loop> = LA*Ax> + LB*Bx> - LC*Cx> - LN*Ny>
Loop[1] = Dot( Loop>, Nx> )
Loop[2] = Dot( Loop>, Ny> )
%--------------------------------------------------------------------
%       Solve constraints with given constants and initial value of qA.
Input  qA = 30 deg,  qA' = 0 rad/sec
SolveSetInputDt( Loop,  qB = 60 deg,  qC = 20 deg )
%--------------------------------------------------------------------
%       Equations of motion - with Kane's method.
Zero = System.GetDynamicsKane()
%--------------------------------------------------------------------
%       Integration parameters and quantities to be output from ODE.
Input  tFinal = 7 sec,  tStep = 0.02 sec,  absError = 1.0E-07
ECheck = System.GetEnergyCheckKane()
Output  t sec,  qA deg,  qB deg,  qC deg,  Loop[1] m,  Loop[2] m,  ECheck Joules
Output  t sec,  FCx Newtons,  FCy Newtons      %, FAx Newtons
%--------------------------------------------------------------------
%       Numerical solve the ODEs for qA''  (and perhaps FCx, FCy, FAx).
ODE( Zero, qA'', FCx, FCy )  MGFourBarForcesAndMotion
%--------------------------------------------------------------------
Save MGFourBarForcesAndMotion.all
Quit
```

Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Four-bar linkage.

9.23 Gyrostat spin stabilization



The figure to the right shows a gyrostat `G` consisting of a carrier A and a thin uniform cylindrical rotor B moving in a Newtonian frame N. Dextral sets of orthogonal unit vectors $\hat{\mathbf{a}}_x$, $\hat{\mathbf{a}}_y$, $\hat{\mathbf{a}}_z$ are fixed in A and are parallel to G's central principal inertia axes. The rotor B has a central moment of inertia of J about its symmetric axes, which is parallel to $\hat{\mathbf{a}}_z$. G's central principal moments of inertia for $\hat{\mathbf{a}}_x$, $\hat{\mathbf{a}}_y$, $\hat{\mathbf{a}}_z$ are denoted $\mathrm{I_{xx}}$, $\mathrm{I_{yy}}$, $\mathrm{I_{zz}}$, respectively. The generalized speeds ω_x, ω_y, ω_z are the $\hat{\mathbf{a}}_x$, $\hat{\mathbf{a}}_y$, $\hat{\mathbf{a}}_z$ measures of A's angular velocity in N. The constant Ω is the $\hat{\mathbf{a}}_z$ measure of B's angular velocity in A.

When numerical values are required, use $J = 0.07634\ \mathrm{kg\,m^2}$, $\mathrm{I_{xx}} = 1.25\ \mathrm{kg\,m^2}$, $\mathrm{I_{yy}} = 4.25\ \mathrm{kg\,m^2}$, $\mathrm{I_{zz}} = 5\ \mathrm{kg\,m^2}$, $\omega_{z\,\mathrm{nom}} = 1\ \frac{\mathrm{rad}}{\mathrm{sec}}$.

- Form equations of motion which govern angular motions of the system.
  Result:

$$\begin{aligned} \mathrm{I_{xx}}\,\dot{\omega}_x \;+\; \omega_y\,[J\,\Omega \;-\; (\mathrm{I_{yy}} - \mathrm{I_{zz}})\,\omega_z] \;&=\; 0 \\ \mathrm{I_{yy}}\,\dot{\omega}_y \;-\; \omega_x\,[J\,\Omega \;-\; (\mathrm{I_{xx}} - \mathrm{I_{zz}})\,\omega_z] \;&=\; 0 \\ \mathrm{I_{zz}}\,\dot{\omega}_z \;-\; (\mathrm{I_{xx}} - \mathrm{I_{yy}})\,\omega_x\,\omega_y \;&=\; 0 \end{aligned}$$

- Consider the nominal solution `wx = wy = wx' = wy' = wz' = 0, wz = nwz` where `nwz` is a constant. After introducing the variables `dwx`, `dwy`, `dwz` as perturbations of ω_x, ω_y, ω_z, linearize the equations of motion in the perturbations about the nominal solution. Put the linearized equations in the form `X' = A*X` where `A` is a 3×3 coefficient matrix and `X` is the 3×1 state matrix `[dwx; dwy; dwz]`.
  Result:

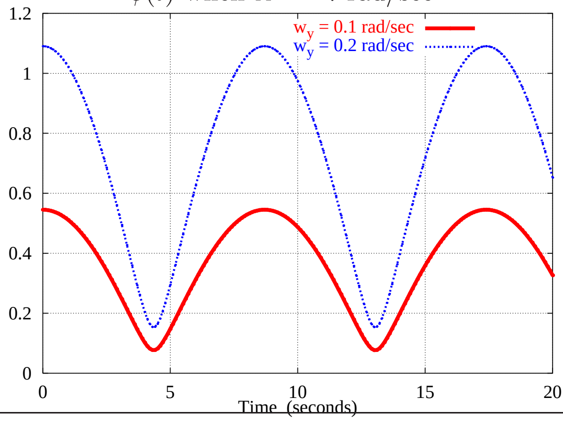
$$A \;=\; \begin{bmatrix} 0 & -\dfrac{J\,\Omega - (\mathrm{I_{yy}} - \mathrm{I_{zz}})\,\omega_{z\,\mathrm{nom}}}{\mathrm{I_{xx}}} & 0 \\ \dfrac{J\,\Omega - (\mathrm{I_{yy}} - \mathrm{I_{zz}})\,\omega_{z\,\mathrm{nom}}}{\mathrm{I_{yy}}} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

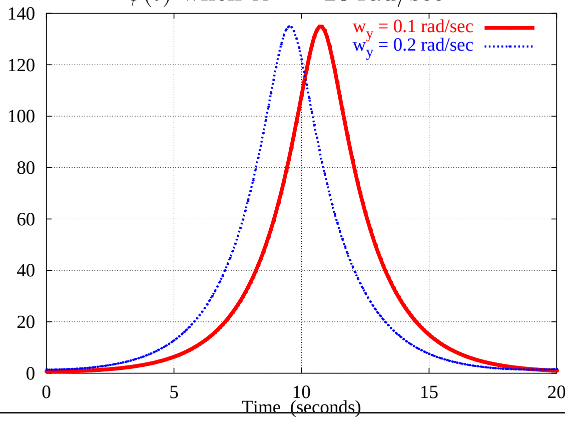
- Determine values of Ω which result in eigenvalues λ of A that are positive.
  Result: The **M**otion**G**enesis response $0.0011\,(9.824 + \Omega)\,(49.12 + \Omega) \;+\; \lambda^2 \;=\; 0$ shows

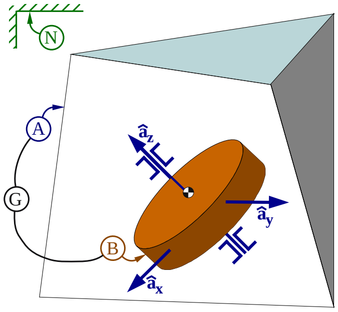
λ^2 is positive when	$-49.12 \;<\; \Omega \;<\; -9.82$	$\mathrm{Real}(\lambda) \;>\; 0$	so solution is "**unstable**"
λ^2 is negative when	$\Omega \;>\; -9.82$ or $\Omega \;<\; -49.12$	$\mathrm{Real}(\lambda) \;=\; 0$	so solution is "**stable**"

- Using the nonlinear equations of motion, run four 20 sec simulations, all with $\omega_x = 0$ and $\omega_z = 1$. Plot the time-history of ϕ, the angle between $\hat{\mathbf{a}}_z$ and the inertial angular momentum of G. For each simulation, check that H (the magnitude of the gyrostats's angular momentum in N) is time-invariant.
  Result: See the plots. By examining (or plotting) the simulation output file `MGGyrostatSpinStability.1`, it is clear that `H` is time-invariant.

#	Ω	ω_y
A	-7	0.02
B	-7	0.01
C	-20	0.02
D	-20	0.01


$\phi(t)$ when $\Omega \;=\; -7$ rad/sec


$\phi(t)$ when $\Omega \;=\; -20$ rad/sec



```
% File: MGGyrostatSpinStability.txt
% Copyright (c) 2009 Motion Genesis LLC.  All rights reserved.
%---------------------------------------------------------------------
NewtonianFrame  N          % Newtonian reference frame
RigidBody       A          % Carrier
RigidFrame      B          % Rotor
%---------------------------------------------------------------------
Constant  Omega = -20 rad/sec  % B’s’ angular speed  in A
Constant  J = 0.07634 kg*m^2   % B’s moment of inertia about spin axis
Variable  wx’,  wy’,  wz’
SetGeneralizedSpeed( wx, wy, wz )
%---------------------------------------------------------------------
%       Mass and inertia   (attribute G’s inertia to A)
A.SetMassInertia( m,  Ix = 1.25 kg*m^2,  Iy = 4.25 kg*m^2,  Iz = 5 kg*m^2 )
%---------------------------------------------------------------------
%       Angular velocities
A.SetAngularVelocityAcceleration( N, wx*Ax> + wy*Ay> + wz*Az> )
B.SetAngularVelocityAcceleration( A, Omega*Az> )
%---------------------------------------------------------------------
%       Velocity and acceleration of G’s mass center is 0>
Acm.SetVelocityAcceleration(  N,  0>  )
%---------------------------------------------------------------------
%       Equations of motion
Zero = System.GetGeneralizedForce() + Frstar() + Gyrostat(FrStar,CYLINDER,A,B,J)
%---------------------------------------------------------------------
%       Calculate gyrostat’s angular momentum
H> = System.GetAngularMomentum(Acm) + Gyrostat(Angmom,CYLINDER,A,B,J)
H = GetMagnitude( H> )
%---------------------------------------------------------------------
%       Angle between angular momentum vector H> and Az>
phi = AngleBetweenUnitVectors(  GetUnitVec( H> ),  Az> )
%---------------------------------------------------------------------
%       Integration parameters and values for constants and variables
Input  tFinal = 20,  tStep = 0.1,  absError = 1.0E-07
Input  wx = 0 rad/sec,  wy = 0.02 rad/sec,  wz = 1 rad/sec
%---------------------------------------------------------------------
%       Quantities to be output by ODE command.
Output  t sec,  phi degs,  H kg*m^2/s
ODE( Zero,  wx’, wy’, wz’ )  MGGyrostatSpinStability
%*********************************************************************
%       STABILITY ANALYSIS
%*********************************************************************
%       Linearization: Perturbation vars + nominal solution parameters.
Variable  dwx’,  dwy’,  dwz’      % Perturbations of wx, wy, wz.
Constant  nwz = 1 rad/sec         % Nominal solution for wz.
%---------------------------------------------------------------------
%       Check nominal solution satisifies the equations of motion.
Check = Evaluate( Zero,  wx=0, wx’=0,  wy=0, wy’=0,  wz=nwz, wz’=0 )
%---------------------------------------------------------------------
%       Linearize equations of motion about nominal solution.
Perturb = Linearize1( Zero, wx = 0 : dwx,  wx’ = 0 : dwx’,  wy = 0 : dwy, &
                            wy’ =0 : dwy’, wz = nwz: dwz,  wz’ = 0 : dwz’ )
Solve( Perturb,  dwx’, dwy’, dwz’ )
%---------------------------------------------------------------------
%       Form, X, X’, and A matrices in the matrix equation X’ = A * x
Xm = [ dwx;  dwy;  dwz ]
Xp = Dt( Xm )
Am = D(  Xp,  Transpose(Xm)  )
%---------------------------------------------------------------------
%       To find eigenvalues of Am symbolically, find the roots of the
%       equation found by setting determinant( Lambda * I - A ) = 0.
Variable Lambda
det = Determinant( Lambda * GetIdentityMatrix(3) - Am )
det /= Lambda                    % Inspection of det shows Lambda = 0 is a root
%---------------------------------------------------------------------
%       Find values of Omega which result in Lambda > 0.
det := EvaluateAtInput( det,  Omega = Omega )
%---------------------------------------------------------------------
Save MGGyrostatSpinStability.all
Quit
```

Note: Problem solution at www.MotionGenesis.com ⇒ Get Started ⇒ Gyros.

10 Other information

10.1 Functions and commands

Type **HELP** at a **M**otion**G**enesis line prompt to see an on-screen list of ≈ 100 commands.
Type **HELP commandName** for detailed help (e.g., type **HELP Solve** for help with the **Solve** command).
Most commands may be nested. For example, the `Dot` command may appear as an argument of the `acos` command, e.g., `theta = acos( Dot(Ax>, By>) )`.

10.2 Stand-Alone commands

Many commands such as `Renee = cos(x)` use an equals sign to make an assignment. Alternately, "**stand-along commands**" make assignments without an explicit equals sign. For example:

```
3*x + 4*y = 37
4*x - 2*y = -2
```

can be solved by executing the following **M**otion**G**enesis input file

```
   (1) Variable  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. Other stand-alone commands include `Rotate`, `Translate`, `SetVelocity`, ...

10.3 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 **M**otion**G**enesis input line in the form

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

where `X` is the *name* of an expression (not the expression itself) and `list_of_arguments` stands for arguments of the dual function, then **M**otion**G**enesis interprets this line as

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

Dual functions differ from other commands in that they avoid changing the functional character of an expression, but alter its appearance.

10.4 Creating your own commands: .A and .R files

You can create new commands by creating an ASCII file that resides in the **M**otion**G**enesis `MGToolbox` 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 **M**otion**G**enesis 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 **M**otion**G**enesis line prompt. Typing `Wow = SUM(3,5)` results in `Wow = 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 **M**otion**G**enesis 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 special symbols to assist in creating .a and .r files. The symbol `#NUM_ARGS#` evaluates to the number of arguments passed to the .a or .r command; `#NewtonianFrame#` evaluates to the name declared in the NewtonianFrame declaration. 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# }
```

10.5 Default settings

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

```
     (1) SetAutoFactor( OFF )           (1) SetAutoFactor( 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
```

`SetAutoRhs` is another default setting.

- When `SetAutoRhs` is `ALL`, the right-hand side of all quantities are automatically substituted for the quantity.
- When `SetAutoRhs` is `ON`, substitutions are made when the right-hand side of quantities are "simple".
- When `SetAutoRhs` is `OFF`, no substitutions are made.

One way to see the effect of `SetAutoRhs` is to compare the files below.

```
   (1) SetAutoRhs( OFF )    (1) SetAutoRhs( ON )     (1) SetAutoRhs( ALL )
   (2) Constant b           (2) Constant b           (2) Constant 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)
```

To view the values assigned to all settings, type `DEFAULTS` at a line prompt.

10.6 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

10.7 Editing keys for online editing in PC/Windows

`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`