Chapter 28

Computer techniques

Available in .pdf format at www.MotionGenesis.com ⇒ Textbooks ⇒ Resources

28.1 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 3.141592 . . ., and `imaginary` as $\sqrt{-1}$.

28.2 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
```

28.3 Symbolic differentiation with MotionGenesis

Symbolic manipulators are useful for calculating ***partial derivatives*** and ***ordinary time-derivatives***.
Note: **M**otion**G**enesis 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'
```

28.4 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
```

Solutions of previous *linear* algebraic equations with MATLAB® (numeric)

After assigning $t = 0.2$, MATLAB® **numerically** solves the linear equations in Section 28.1 via:

```
t = 0.2;
%-------------------------------------------------------------
Coef(1,1) = 3;   Rhs(1,1) = -(9*sin(t) - 12);
SolutionToAxEqualsB = Coef \ Rhs;
x = SolutionToAxEqualsB(1)
%-------------------------------------------------------------
Coef(1,1) = 3;   Coef(1,2) = 2;   Rhs(1,1) = -(9*sin(t) - 12);
Coef(2,1) = 2;   Coef(2,2) = 4;   Rhs(2,1) = -(5*cos(t) - 11);
SolutionToAxEqualsB = Coef \ Rhs;
y = SolutionToAxEqualsB(1)
z = SolutionToAxEqualsB(2)
%-------------------------------------------------------------
Coef(1,1) = 3;   Coef(1,2) = 2;   Coef(1,3) = 2;   Coef(1,4) = 3;   Rhs(1,1) = 9*sin(t);
Coef(2,1) = 2;   Coef(2,2) = 4;   Coef(2,3) = 2;   Coef(2,4) = 3;   Rhs(2,1) = 5*cos(t);
Coef(3,1) = 4;   Coef(3,2) = 5;   Coef(3,3) = 6;   Coef(3,4) = 7;   Rhs(3,1) = 11;
Coef(4,1) = 9;   Coef(4,2) = 8;   Coef(4,3) = 7;   Coef(4,4) = 6;   Rhs(4,1) = 15;
SolutionToAxEqualsB = Coef \ Rhs;
x1 = SolutionToAxEqualsB(1)
x2 = SolutionToAxEqualsB(2)
x3 = SolutionToAxEqualsB(3)
x4 = SolutionToAxEqualsB(4)
```

28.5 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]
```

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

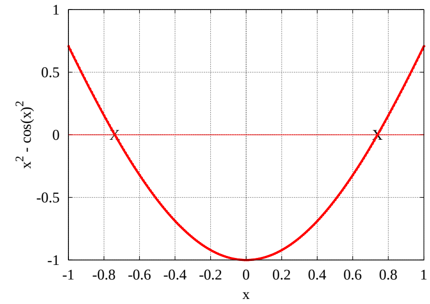
```
>> polynomial = [1, 2, 3, 5, 9, 17];
>> p = roots( polynomial )

p =
   0.8763 + 1.4560i
   0.8763 - 1.4560i
  -1.8576
  -0.9475 + 1.5070i
  -0.9475 - 1.5070i
```

28.6 Solutions of *nonlinear* algebraic equations

One way to find the solution to a ***nonlinear algebraic equation*** is to graph the function and identity the values of x that make the function equal to 0. For example, the graph of the function to the right is **nonlinear** (i.e., it is **not a line**) and has two solutions, namely $x \approx 0.7391$ and $x \approx$ -0.7391.

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



Another way to solve a nonlinear equation is to use a computer program. Most algorithms start with a **guess** and iterate towards a solution (frequently a solution close to the guess). For example, the following MotionGenesis commands produce the solution $x = 0.7391$.

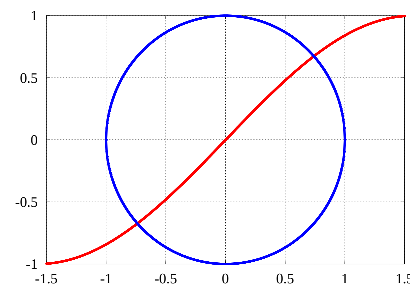
```
Variable  x
Solve( x^2 - cos(x)^2,  x=2 )      % x=2 is a guess to a solution
Quit
```

28.6.1 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 $x = 0.7391$, $y = 0.6736$ and $x =$ -0.7391, $y =$ -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] 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.

$$x^2 + y^2 = 1$$
$$y = \sin(x)$$



For example, the following MotionGenesis 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
```

28.6.2 Solutions of *coupled nonlinear* algebraic equations with MATLAB®

- Use a **text editor** to create the file `NonlinearSolveCircleSine.m` (as shown below).
- Invoke MATLAB® and ensure `NonlinearSolveCircleSine.m` is in the current working directory.
- Type `NonlinearSolveCircleSine` at the MATLAB® prompt.
- Note: The MATLAB® nonlinear solver `fsolve` requires the ***optimization*** toolbox.

```
%--------------------------------------------------------------------
%      File:  NonlinearSolveCircleSine.m
%   Purpose:  Solving a set of nonlinear equations with Matlab
%      Note:  Requires Matlab's optimization toolbox
%--------------------------------------------------------------------
function solutionToNonlinearEquations = NonlinearSolveCircleSine
initialGuess = [ 2, 0 ];
solveOptions = optimset('fsolve');
solutionToNonlinearEquations = fsolve( @CalculateFunctionEvaluatedAtX, initialGuess, solveOptions );

%========================================================================
function fx = CalculateFunctionEvaluatedAtX( X )
x = X(1);    y = X(2);
fx(1) = x^2 + y^2 - 1;        % x^2 + y^2 = 1    (unit circle)
fx(2) = y - sin(x);           % y = sin(x)       (sine wave)
```

28.7 Solution of ordinary differential equations (ODEs)

Computers languages such as C and Fortran 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 are **much 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).

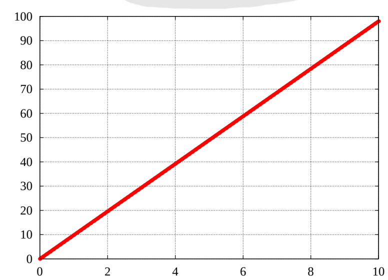
28.7.1 Solution of 1^{st}-order ODE (numerical integration)

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

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

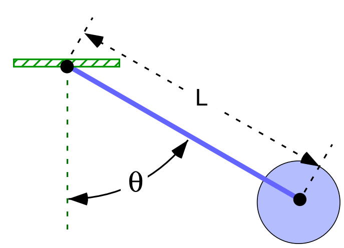
Although this ODE is easily solvable by ***separation of variables*** and integration as $v(t) = v(0) + 9.8\,t$, it can also be solved by computer numerical integration as shown in the following **M**otion**G**enesis file.

```
% 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`

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

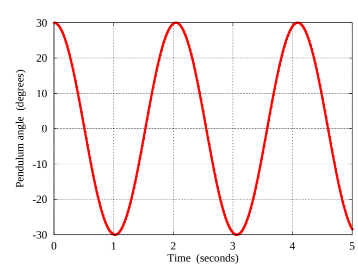


The figure to the right shows a 1 m pendulum swinging on Earth's surface. The pendulum's motion is governed by the **nonlinear** 2^{nd}-order ODE

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

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

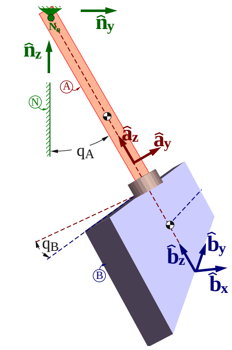
```
% File: ClassicParticlePendulumShort.al
%-------------------------------------------------------
Variable theta'' = -9.8*sin(theta)
Input  theta=30 deg, theta'=0,  tFinal=5, integStp=0.02
Output t sec, theta deg
ODE() ClassicParticlePendulum
Quit
```



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`

28.7.3 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).

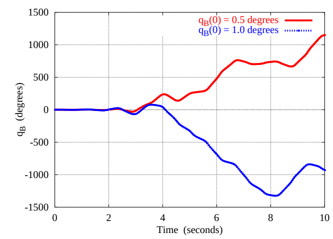


$$\ddot{q}_A = \frac{2\,[508.89\,\sin(q_A) \;-\; \sin(q_B)\,\cos(q_B)\,\dot{q}_A\,\dot{q}_B]}{-21.556 + \sin^2(q_B)}$$

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

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

```
Variable  qA'', qB''    % Angles and their first and 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, integStp=0.02 sec, absError=1.0E-07,  relError=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`

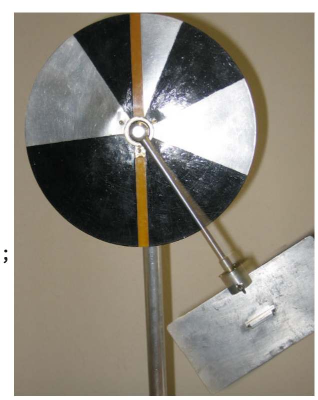
Solution of ordinary differential equations with MATLAB®

The MATLAB® solution for the ODEs in Section 28.7.3 has two functions. The first function is the main routine that drives the numerical integrator. The second function contains the differential equations in first-order form. To use MATLAB®:

- Use a text editor to create the file `BabybootODE.m`
- Invoke MATLAB® and ensure `BabybootODE.m` is in the current working directory
- Type `BabybootODE` at the MATLAB® prompt

```
%--------------------------------------------------------------------
% File:  BabybootODE.m   (solving differential equations with MATLAB)
%--------------------------------------------------------------------
function BabybootODE
degreesToRadians = pi/180;
initialState = [ 90*degreesToRadians  1.0*degreesToRadians  0  0 ];
timeInterval = linspace( 0, 10, 1000 );
odeOptions = odeset( 'RelTol', 1.0e-7, 'Abstol', 1.0E-8 );
[time,stateMatrix] = ode45( @odefunction, timeInterval, initialState, odeOptions );
qB = stateMatrix(:,2);
plot( time, qB/degreesToRadians, 'r-' )
xlabel( ' Time (seconds) ' );
ylabel( ' Plate angle (degrees) ' );

function timeDerivativeOfState = odefunction( t, state )
qA  = state(1);    % Pendulum angle
qB  = state(2);    % Plate angle
qAp = state(3);    % qA', time derivative of the pendulum angle
qBp = state(4);    % qB', time derivative of the plate angle
qApp = 2*( 508.89*sin(qA) - sin(qB)*cos(qB)*qAp*qBp ) / (-21.556 + sin(qB)^2);
qBpp = -sin(qB)*cos(qB)*qAp^2;
timeDerivativeOfState = [qAp; qBp; qApp; qBpp];
```



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

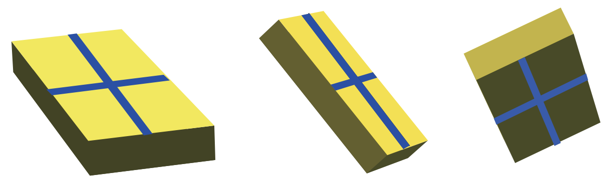
$$\dot{\omega}_x = \frac{(I_{yy} - I_{zz})}{I_{xx}}\,\omega_z\,\omega_y$$

$$\dot{\omega}_y = \frac{(I_{zz} - I_{xx})}{I_{yy}}\,\omega_x\,\omega_z$$

$$\dot{\omega}_z = \frac{(I_{xx} - I_{yy})}{I_{zz}}\,\omega_y\,\omega_x$$

A **M**otion**G**enesis solution[1] 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, ω_x, ω_y, ω_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
```



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

28.8 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 )
```

28.9 Matrix calculations with MATLAB®

Matrices in MATLAB®

```
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 MATLAB®

```
A = zeros( 3 );          % 3x3 matrix of zeros
B = zeros( 2, 3 );       % 2x3 matrix of zeros
F = eye( 3 );            % 3x3 identity matrix
G = eye( 2, 3 );         % 2x3 matrix with 1 along the diagonal and 0 elsewhere
```

Transpose of a matrix with MATLAB®

```
A  =  [ 1, 2, 3;  4, 5, 6 ]
B  =  A'                     % The prime symbol denotes transpose
```

Submatrices with MATLAB®

```
A = [1, 2, 3, 4;  5, 6, 7, 8;  9, 10, 11, 12];
B = A( 2, 1:4 )          % 1x4 matrix with row 2 of A
C = A( 1:3, 2 )          % 3x1 matrix with col 2 of A
D = A( 1:3, 2:4 )        % 3x3 matrix with cols 2 to 4 of A
E = A( 2:3, 2 )          % 1x2 matrix with elements 2,2 and 3,2 of A
```

Determinant and inverse of a matrix with MATLAB®

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

Solving linear algebraic equations with MATLAB® (numerical)

```
B = [1; 2; 3]
A = [2, 3, 4;
     5, 7, 9;
     7, 3, 5]
X = A\B
```

Eigenvalues and eigenvectors with MATLAB®

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