Lab 2: Getting started with Simbody

Simbody is a numerical software library that efficiently simulates systems governed by Newton's laws ($\mathbf{F} = m\mathbf{a}$). From a falling apple to a walking human to a protein with thousands of molecules, Simbody simulates a wide range of biological structures and physical systems with user-specified forces, mass and inertia properties, and joints/constraints. Simbody is built on efficient algorithms that optimize linear algebra calculations, numerical integration, and forming/solving equations of motion. Simbody also provides on-screen visualization of simulation results through VTK (**V**isualization **T**ool**K**it).

Developing physics-based simulations of biological structures with SimTK Simbody requires a rudimentary understanding of biology and physics and a background in C or C^{++}. Before starting Lab 2, you should have a C^{++} compiler installed on your computer, should have read and understood the C^{++} coding standards, and should have completed Lab 1. The point of this laboratory is to:

- Download and install the VTK and SimTK libraries
- Compile and build a simulation of a falling apple (1D motion)
- Specify initial values, mass and inertia properties, and integration parameters in Simbody
- Plot and animate your simulation results
- Modify the falling apple C^{++} code to simulate and animate 2D projectile motion
- Create and compile C^{++} code to simulate and animate a 3D rigid body

2.1 Downloading and installing VTK and SimTK libraries

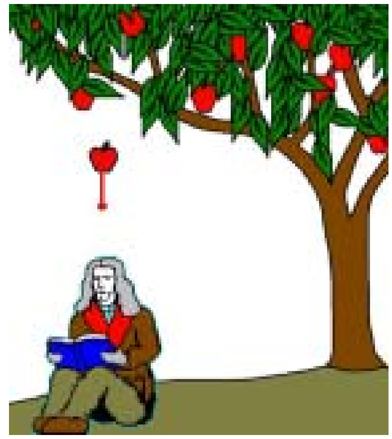
To download and install the VTK and SimTK libraries, proceed as follows
(These libraries are described in Section 2.8.)

- Create a directory (folder) in the `C:\Program Files` directory named **SimTK**
  In other words, ensure there is a directory whose path is `C:\Program Files\SimTK`
- Go to www.simtk.org/home/training
- Click on the Documents link on the left-hand side
- Download the file `LibrariesAndHeadersForVTKWin.zip` to `C:\Program Files\SimTK`
- Download the file `LibrariesAndIncludeFilesForSimTKWinIntel.zip` to `C:\Program Files\SimTK`
- Unzip the VTK libraries to `C:\Program Files\SimTK`.
  For example, right-mouse click on the file `LibrariesAndHeadersForVTKWin.zip`, hover the mouse on **WinZip** and select **Extract to . . . .** Browse to `C:\Program Files\SimTK` and click **Extract**.
  This extracts 1000^{+} files ($\approx$ 100 MB) to the folder `C:\Program Files\SimTK\VTK`
- Unzip the SimTK libraries to `C:\Program Files\SimTK`.
  For example, right-mouse click on the file `LibrariesAndIncludeFilesForSimTKWinIntel.zip`, hover the mouse on **WinZip** and select **Extract to . . . .** Browse to `C:\Program Files\SimTK` and click **Extract**.
  This extracts 100^{+} files ($\approx$ 30 MB) to the folder `C:\Program Files\SimTK\core`

Note: These instructions and associated files (e.g., .h, .lib, and .dll files) have been tested on Windows XP. If the VTK and SimTK libraries for your operating system are not available on www.simtk.org, try to emulate the instructions for creating and packaging these files shown in Section 2.9.

[0]Last updated April 13, 2007 by Paul Mitiguy.

2.2 Simulating Newton's apple



The figure to the right shows an apple falling from a tree. The point of this lab is to simulate the falling apple "by-hand" and with Simbody- and to loosely tie together **biology** (i.e., an apple and its biological parent - the tree), **Newtonian mechanics** (i.e., $\mathbf{F} = m\mathbf{a}$), and **simulation** (i.e., C^{++}, Simbody, and your computer).

To simulate the falling apple, one must make a ***model***.[a] Creating an ***accurate*** model requires ***judgement*** to differentiate between what can be simplified and what cannot. Shown below are simplifying assumptions made when modeling this system.

[a] A ***model*** is a simplified representation of a complex system.

Modeling

1. **The apple is a *rigid body*.**
   The rigid body assumption allows one to replace all the forces by a simpler equivalent set. For example, the actual gravitational forces on the apple are distributed, but with the rigid body assumption and uniform gravitational field assumption, the set of gravitational forces on the apple can be replaced with a single resultant force at the apple's mass center.

2. **The apple can be modeled as a small uniform sphere.**
   This assumption is reasonable due to the fact that air-resistance and other forces (except gravity) are neglected, and the body's orientation is not of interest.

3. **The only motion of interest is vertical translation of the apple relative to the Earth.**
   This assumption is useful if the motion of interest occurs over a short period of time (e.g., less than an hour). Over longer periods of time, the rotation of the Earth causes the particle to move horizontally.

4. **The Earth is a *Newtonian reference frame*.**
   A Newtonian reference frame is a reference frame in which Newton's equation $\mathbf{F} = m\mathbf{a}$ accurately predicts forces and motion.[1] Newton's law $\mathbf{F} = m\mathbf{a}$ requires a non-accelerating and non-rotating reference frame. Although Earth is rotating (daily around its axis and yearly about the sun), the acceleration associated with these motions is ***assumed*** to be insignificantly small.

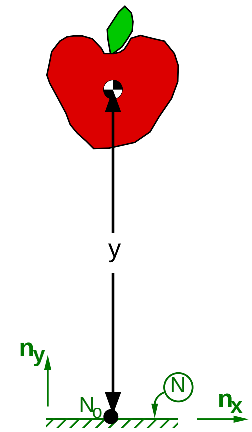
5. **The gravitational attraction of Earth can be approximated as a uniform field.**
   In reality, the gravitational forces on the system vary as objects move further or closer to Earth. This assumption may be unreasonable if the apple falls for many kilometers.

6. **Other than Earth, gravitational forces are negligible.**
   This assumes that the system's motion is not affected by the gravitational attraction of other massive objects, e.g., heavy objects in the same proximity, or the moon, sun, black holes, etc. An example helps validate this assumption. The magnitude of the gravitational force between two identical uniform lead spheres of radius one foot and weight 3035 lbf, whose centers are located only 3 feet apart (their closest point is 1 foot apart) is 3.4×10^{-5} lbf. This is nearly ninety million times smaller than the force exerted by Earth on the sphere.

7. **Many forces are negligible.**
   This assumes that other forces (aerodynamic, friction, magnetic, electrostatic) do not substantially affect the system. This assumption is flawed for certain analyses as it is clear that other forces, e.g., air resistance, can play a major role in the behavior of the system.

[1] A ***Newtonian reference frame*** is sometimes called an ***inertial reference frame*** or ***fixed reference frame*** or ***ground***.

Analytical solution to Newton's falling apple with $\mathbf{F} = m\,\mathbf{a}$



To explore the concepts of numerical simulation in Simbody, it is helpful to do a "by-hand" problem. To analyze the falling apple, introduce a Newtonian reference frame N; right-handed orthogonal unit vectors $\mathbf{n}_\mathrm{x}$, $\mathbf{n}_\mathrm{y}$, $\mathbf{n}_\mathrm{z}$ fixed in N with $\mathbf{n}_\mathrm{x}$ pointing horizontally right and $\mathbf{n}_\mathrm{y}$ pointing vertically upward; a point N_o fixed in N; and the following identifiers:

Quantity	Symbol	Type	Value
Earth's gravity	g	constant	9.8 m/sec
Mass of apple	m	constant	0.142 kg (5 ozm)
Apple's vertical position from N_o	y	variable	varies

- Determine the resultant of all ***forces*** on the apple.
  Note: The study of forces is called ***kinetics*** and varies significantly in fields such as molecular dynamics (electrostatics and van der Waals forces), biomechanics (muscles), aerospace (lift/drag), robotics (actuators), etc. Simbody has special ***subsystems*** for efficiently calculating forces e.g., intermolecular forces that can dominate the simulation speed.
  Result:
  $$\mathbf{F} \;=\; \boxed{\quad\quad}\,\mathbf{n}_\mathrm{y}$$

- Form the position vector $\mathbf{r}$ from point N_o to the apple's ***center of mass*** (shown as ◕).
  Next, twice-differentiate this vector to form the apple's velocity and acceleration.
  Express your results in terms of $\dot{y}$ and $\ddot{y}$, the 1^{st} and 2^{nd} time-derivatives of y.
  Note: Simbody has efficient algorithms for calculating ***kinematics***, e.g., a point's position, velocity, and acceleration, a bases' rotation matrix, and a rigid body's angular velocity and angular acceleration.
  Result:
  $$\mathbf{r} \;=\; y\,\mathbf{n}_\mathrm{y} \qquad\qquad \mathbf{v} \;=\; \boxed{\;}\,\mathbf{n}_\mathrm{y} \qquad\qquad \mathbf{a} \;=\; \boxed{\;}\,\mathbf{n}_\mathrm{y}$$

- Form an equation of motion with Newton's law $\mathbf{F} = m\,\mathbf{a}$.
  Note: There are many ways to form equations of motion including Free-Body-Diagrams, D'Alembert, Lagrange, Hamilton, Gibbs, Kane, Constraint-method, recursive Newton/Euler. Simbody uses a "***Jain and Rodriguez Spatial Operator Generalized Coordinate Method***" that has efficient $O(n)$ (Order-N) properties similar to Featherstone's method.
  Result:
  $$\text{-}m\,g \;=\; m\,\ddot{y}$$

- Solve for the apple's vertical acceleration $\ddot{y}$.
  Note: There are sophisticated linear algebra methods for solving the sets of coupled algebraic equations that arise in solving for accelerations in multi-body systems. For example, methods for solving linear algebraic equations include Choleksy, LU, QR, and SVD decomposition, PCBG (pre-conditioned bi-conjugate gradient), simplex solvers (inequality constraints), sparse matrix solvers, and updating/downdating. The Simbody method for forming equations of motion has $O(n)$ (Order-N) properties that significantly reduce the computational cost of solving for accelerations.
  Result:
  $$\ddot{y} \;=\; \text{-}g$$

- Integrate $\ddot{y}$ twice to find $\dot{y}$ and y in terms of $\dot{y}(0)$ and $y(0)$, the initial values of $\dot{y}$ and y.
  Note: Frequently, it is **impossible** to find closed-form "by-hand" solutions to the differential equations that govern motion. This unusually simple problem allows us to analytically integrate $\ddot{y}$ to find $\dot{y}$ and y. Simbody uses efficient, general-purpose ***numerical integrators*** to solve differential equations. In addition, Simbody allows users to employ ***generalized coordinates*** (sometimes called ***internal coordinates***) which, when chosen properly, provide a near-minimal set of smooth variables which integrate quickly.
  Result:
  $$\dot{y} \;=\; \dot{y}(0) \;+\; \text{-}g\,t \qquad\qquad y \;=\; y(0) \;+\; \dot{y}(0)\,t \;+\; \frac{\text{-}1}{2}\,g\,t^2$$

- The apple is dropped from **rest** from a height of 10 m. Plot (**by hand**) the time-history of $\dot{y}$ and y for $0 \leq t \leq 4$ sec. Note: Plotting functions and data is usually done with programs such as Excel or Matlab.

Compiling, linking, and simulating Newton's apple

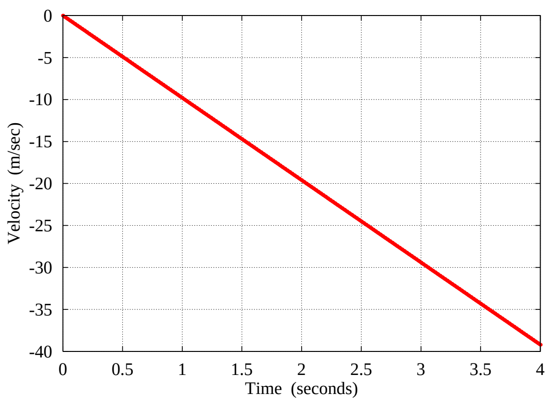
The point of the last section was to simulate a falling apple "by-hand". The point of this section is to use Simbody to simulate the falling apple.

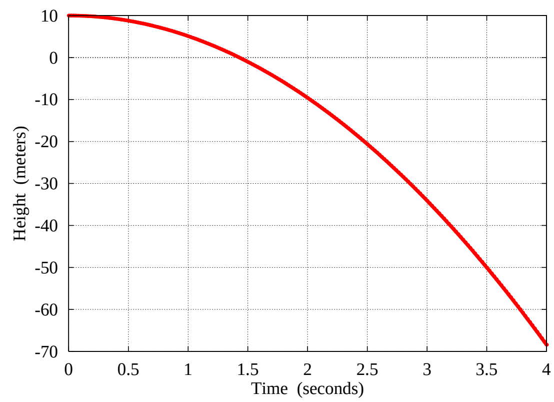
- Go to www.simtk.org/home/training
- Click on the 

 link on the left-hand side
- Download the file `LabGettingStartedWithSimbody.zip`
- Unzip the file `NewtonsApple.cpp`
- Compile this file, link it with the SimTK libraries, and run it (see Section 2.6 for more information)
- Numbers should scroll on your screen. These numbers are also written to the file `NewtonsAppleResults.txt`. This file is useful for plotting (see the next section).

Plotting numerical results for Newton's apple

The apple is dropped from **rest** from a height of 10 m. Using the data in `NewtonsAppleResults.txt` and Excel, Octave, Matlab, or other plotting software, plot the time-history of $\dot{y}$ and y for $0 \le t \le 4$ sec.





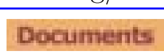
Overlay and visually compare the graph of the results for y produced by Simbody with the graph of the exact analytical function $y(t) = 10 - 4.9\,t^2$. Knowing the exact analytical results at $t = 4$ sec are $y(4) = $ -68.4 m, determine the number of accurate digits Simbody produces for y at $t = 4$ sec. Assess how accurately Simbody simulates the falling apple[2] in light of the fact that double precision calculations carry ≈ 16 significant digits.[3]

Result:

Number of accurate digits $=$ Accuracy: **High/Medium/Low**

Visualizing Newton's apple with VTK in Simbody

Plots show relationships between various quantities and are useful for inclusion in printed matter (e.g., journals, textbooks, and .pdf documents). On-screen visualization of a simulation provides significantly more information and enjoyment. To visualize Newton's falling apple with VTK in Simbody

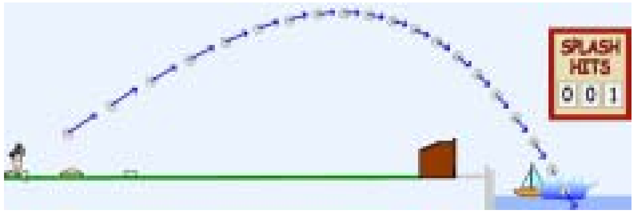
- Go to www.simtk.org/home/training
- Click on the 

 link on the left-hand side
- Download the file `LabGettingStartedWithSimbody.zip`
- Unzip the file `NewtonsAppleForVisualization.cpp`
- Compile this file, link it with the SimTK and VTK libraries, and run it (see Section 2.6)
- Enjoy the animation

[2]Due to numerical integration error, Simbody is unable to simulate every system with the same accuracy.

[3]The largest positive and negative IEEE double precision numbers are $\approx \pm 1.7976931348623157 \times 10^{308}$. The double precision numbers with the smallest exponents are $\approx \pm 2.22507385072020 \times 10^{-308}$.

2.3 Simulating projectile motion



The figure to the right shows a baseball being hit out of AT&T park. The point of this lab is to modify the 1D falling apple simulation from Section 2.2 to investigate 2D projectile motion of a baseball.

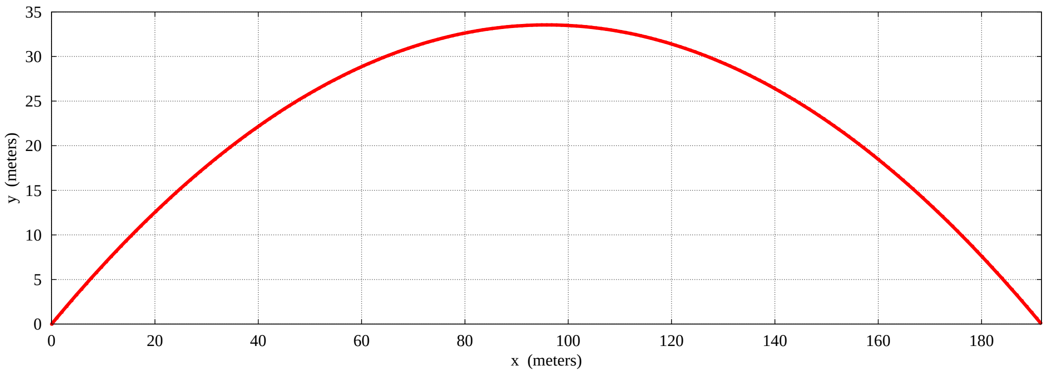
For the following questions: use the same modeling assumptions as were used for the falling apple in Section 2.2; assume the baseball is launched at 35° from the horizontal with an initial speed of 44.7 m/sec (100 mph); and use the following symbols and values.

Quantity	Symbol	Type	Value
Earth's gravity	g	constant	9.8 m/sec
Mass of baseball	m	constant	0.142 kg (5 ozm)
Radius of gyration of baseball (sphere)	r	constant	2.3114 cm (0.91 in)
Baseball's horizontal position from N_o	x	variable	Initially $x=0$
Baseball's vertical position from N_o	y	variable	Initially $y=0$

- Verify the analytical solutions for $x(t)$, $y(t)$, and $y(x)$ are

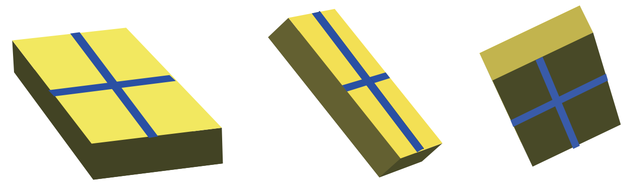
$$x(t) = 44.7\,\cos(35°)\,t \qquad y(t) = 44.7\,\sin(35°)\,t - 4.9\,t^2 \qquad y(x) = \tan(35°)\,x - \frac{4.9}{[44.7\,\cos(35°)]^2}\,x^2$$

- Knowing $y=0$ is ground, verify that the ball hits the ground at $t \approx 5.232422$ sec and $x \approx 191.5909$ m.
- Plot the exact analytical results for $y(x)$ for $0 \le x \le 192$ m.



- Modify the Simbody C^{++} program `NewtonsAppleForVisualization.cpp` to simulate the baseball and visualize its motion. Submit your modified and **fully commented** C^{++} program `ProjectileMotion.cpp` to your SimTK project at www.simtk.org.
- Plot the Simbody results for $y(x)$ for $0 \le x \le 192$ m (simulate for 5.23 sec with an integration step of 0.01 sec).
  Result:
  See previous plot.
- Assess how accurately Simbody simulates the baseball by comparing its solution for $y(t)$ with the exact analytical solution for $y(t)$ at $t=5.2$ sec.
  Result:
  Number of accurate digits = ___ Accuracy: **High/Medium/Low**
- Optional**: Use the Simbody numerical integrator root finder to find the value of x at which the ball hits the ground.

2.4 Simulating a 3D rigid body



The figure to the right shows a rotating rigid book. The point of this lab is to create C^{++} code that links with Simbody and to investigate the affect of ***moments of inertia*** on 3D rotation.

To facilitate this simulation, right-handed orthogonal unit vectors $\mathbf{b}_\mathrm{x}$, $\mathbf{b}_\mathrm{y}$, $\mathbf{b}_\mathrm{z}$ are fixed in the book and parallel to central principal axes. The following table describes other relevant quantities.
(It is interesting that the equations governing the book's rotation (shown below and to the right for reference only) are significantly more difficult to form and solve than the falling apple in Section 2.2 or the projectile motion in Section 2.3.)

Quantity	Symbol	Type
Central moment of inertia for $\mathbf{b}_\mathrm{x}$	I_{xx}	constant
Central moment of inertia for $\mathbf{b}_\mathrm{y}$	I_{yy}	constant
Central moment of inertia for $\mathbf{b}_\mathrm{z}$	I_{zz}	constant
$\mathbf{b}_\mathrm{x}$ measure of angular velocity	ω_x	dependent variable
$\mathbf{b}_\mathrm{y}$ measure of angular velocity	ω_y	dependent variable
$\mathbf{b}_\mathrm{z}$ measure of angular velocity	ω_z	dependent variable

$$\begin{aligned}
\dot{\omega}_x &= \left[(I_{yy} - I_{zz})\,\omega_z\,\omega_y\right] \,/\, I_{xx} \\
\dot{\omega}_y &= \left[(I_{zz} - I_{xx})\,\omega_x\,\omega_z\right] \,/\, I_{yy} \\
\dot{\omega}_z &= \left[(I_{xx} - I_{yy})\,\omega_y\,\omega_x\right] \,/\, I_{zz}
\end{aligned}$$

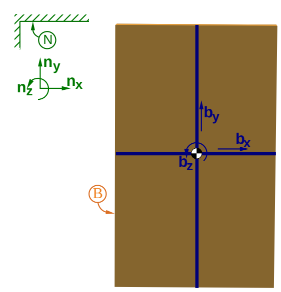
- Assuming the book is not translating, its kinetic energy and central angular momentum are

$$K = \frac{1}{2}\left(I_{xx}\,\omega_x^2 + I_{yy}\,\omega_y^2 + I_{zz}\,\omega_z^2\right)$$

$$\begin{aligned}
\mathbf{H} &= I_{xx}\,\omega_x\,\mathbf{b}_\mathrm{x} + I_{yy}\,\omega_y\,\mathbf{b}_\mathrm{y} + I_{zz}\,\omega_z\,\mathbf{b}_\mathrm{z} \\
&= H_x\,\mathbf{b}_\mathrm{x} + H_y\,\mathbf{b}_\mathrm{y} + H_z\,\mathbf{b}_\mathrm{z}
\end{aligned}$$

  Using your intuition, circle the quantities that you **guess** remain constant (are "conserved").

  $\omega_x \qquad \omega_y \qquad \omega_z \qquad K \qquad H_x \qquad H_y \qquad H_z \qquad |\mathbf{H}|$



- Spin about the ***minimum axis***.
  Use Simbody to simulate this system for $0 \le t \le 4$ with initial values (in rad/sec) of ω_x=0.2, ω_y=7.0, and ω_z=0.2. Output t, ω_x, ω_y, ω_z, H_x, H_y, H_z, $|\mathbf{H}|$, and K. Use a uniform-density book of mass 0.4 kg and dimensions L_x=20 cm high, L_y=30 cm wide, and L_z=5 cm thick.

- Checking your numerical results, circle the quantities that remain constant (are "conserved").

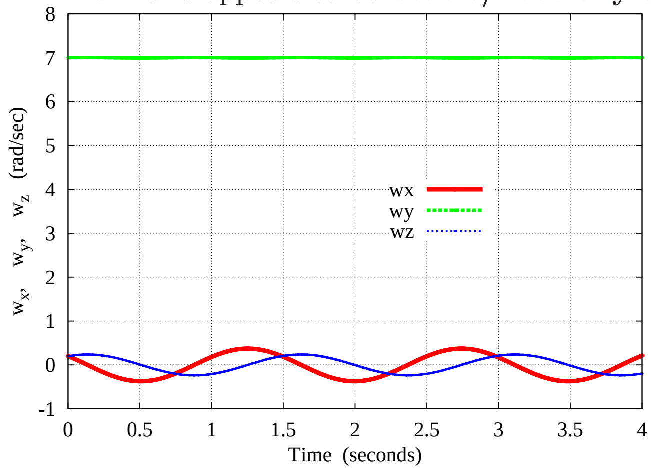
  $\omega_x \qquad \omega_y \qquad \omega_z \qquad K \qquad H_x \qquad H_y \qquad H_z \qquad |\mathbf{H}|$

- Spin about the ***intermediate axis***.
  Simulate this system again except use initial values ω_x=7.0, ω_y=0.2, and ω_z=0.2.

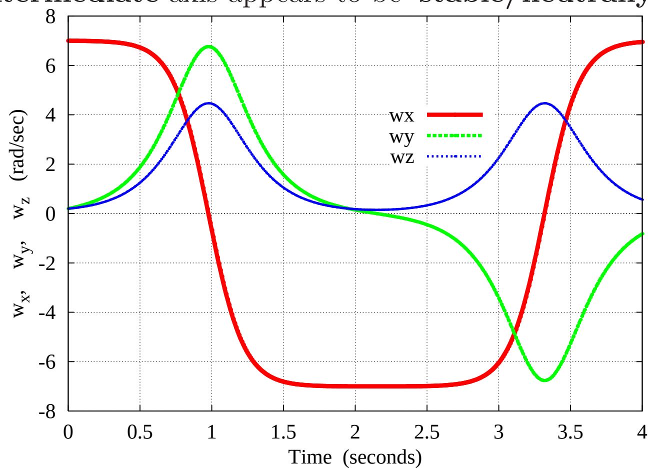
- Spin about the ***maximum axis***.
  Simulate this system again except use initial values ω_x=0.2, ω_y=0.2, and ω_z=7.0.

- Create plots corresponding to initial spin about the ***minimum axis***, ***intermediate axis***, and ***maximum axis***. In view of your plots and by experimentally spinning a book about each of the three axes, circle the phrase that best describes the stability of spin about each axis.

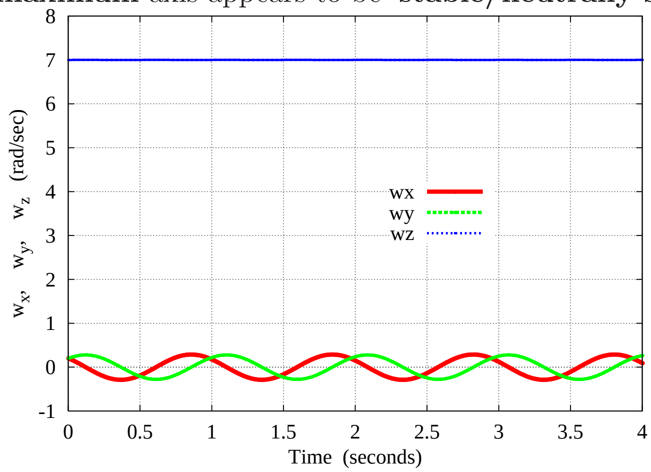
Spin about the **minimum** axis appears to be **stable/neutrally stable/unstable**.



Spin about the **intermediate** axis appears to be **stable/neutrally stable/unstable**.



Spin about the **maximum** axis appears to be **stable/neutrally stable/unstable**.

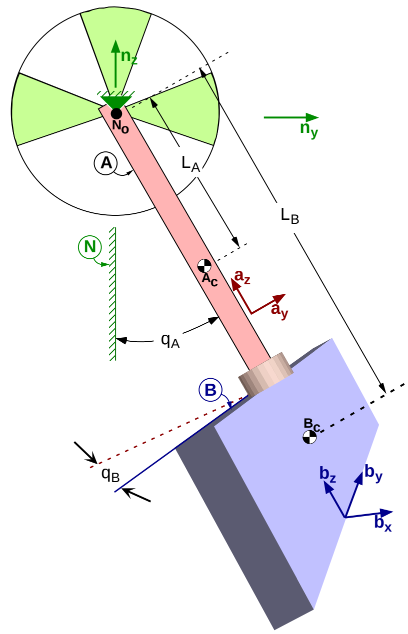


2.5 Biosimulation project description

Submit a description of your project and include the following:

- One or more schematics showing the points, particles, bodies, constraints, etc.
- A short paragraph describing the physical system
- A short list of modeling considerations
- An interesting question that you would like to answer
- A list of simple numerical experiments that help validate your simulation results
  (e.g., with no gravity, no other applied forces, and no initial motion, the system remains at rest.)

To help understand what to do, consider the following simple system.
(Note: Your project should be more biologically significant.)



The figure to the right is a schematic representation of a swinging babyboot attached by a shoelace to a rigid support. The mechanical model of the babyboot consists of a thin uniform rod A attached to a fixed support N by a revolute joint, and a uniform plate B connected to A with a second revolute joint so that B can rotate freely about A's axis. (Note: The revolute joints' axes are perpendicular, *not* parallel.)

Modeling considerations

- The bodies are rigid.
- The revolute joints are frictionless.
- There is no slop or flexibility in the revolute joints.
- The Earth is a Newtonian reference frame.
- Air resistance is negligible.
- The force due to Earth's gravitation is uniform and constant.
- Other distance forces, e.g., electromagnetic and gravitational forces, are negligible.

2.6 Creating Simbody programs with Microsoft Visual C++

To compile or link your own C++ program with Simbody, you need to inform the Microsoft Visual Studio compiler and linker of the existence and location of the library header files (.h files) as well as the library (.lib and .dll) files.

2.6.1 Creating an empty Simbody Win32 Console Application in the SimTK folder

- Ensure Microsoft Visual C++ Express Edition is installed on your computer
- Click on the Windows Start menu, select **Programs**, then slide-the-mouse to the **Visual C++ 2005 Express Edition** folder, then slide-right-and-down to the **Microsoft Visual C++ 2005 Express Edition** executable
- From within the Visual C++ 2005 Express Edition, click on the **File** menu, click on the **New** menu item, and the slide-right to click on the **Project. . .** menu item.
  - Under **Project types:**, select **Win32**
  - Give the project a sensible **name**, e.g., `NewtonsApple`
  - Make the project's **location** `C:\Program Files\SimTK` (If necessary, create the folder SimTK)
  - Click the OK button, then click the Next > button **(not the Finish button)**
  - Ensure the **Application Settings** are set to create a **Console Application**, an **Empty Project**, with **NO precompiled header**.
  - Click the Finish button
- This process creates a file `NewtonsApple.sln` in the folder `C:\Program Files\SimTk\NewtonsApple`
- From within Visual C++, click on the **File** menu and then click on the **Exit** menu item.

2.6.2 Adding files to the "NewtonsApple" Win32 Console Application

- Double-click on the file `NewtonsApple.sln` in the folder `C:\Program Files\SimTk\NewtonsApple` (This should invoke Microsoft Visual C++ and open the "NewtonsApple" project.)
- Click on the **Project** menu and slide-down and click on **Add Existing Item. . .** .
- Browse to the directory containing the files to add, select the files, and click Add.
  (For example, to add the file `C:\Program Files\SimTK\NewtonsApple\NewtonsApple.cpp`, browse to the directory `C:\Program Files\SimTK\NewtonsApple`, select the file `NewtonsApple.cpp`, then click Add.
  Note: A commented sample `NewtonsApple.cpp` file is available by visiting www.simtk.org/home/training, clicking on the Documents link on the left-hand side, downloading **LabGettingStartedWithSimbody.zip**, and extracting the file **NewtonsApple.cpp**. Alternately, type the file in Section 2.7.
- From within Visual C++, click on the **File** menu and then click on the **Save all** menu item.
- Click on the **File** menu and then click on the **Exit** menu item.

2.6.3 Compiling your project with the SimTK libraries

Certain information must be present in your **program** to **compile** a program that uses Simbody. The following three lines should appear near the top of a C++ file that calls Simbody methods:

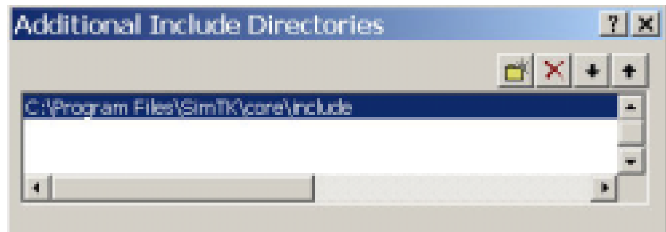
```
#include "SimTKsimbody.h"
using namespace SimTK;
using namespace std;
```

Additionally, certain information must be available for your **compiler**.

- If the `NewtonsApple` project is not already open, double-click on the file `NewtonsApple.sln`

- From within your Microsoft Visual C++ compiler, click on the **Project** menu, slide-down and click on **NewtonsApple Properties . . .** .
- If necessary, expand **Configuration Properties** by clicking on its + sign.
- To the right of the drop-down menu titled **Configuration:**, select **All configurations**
- If necessary, expand **C/C++** by clicking on its + sign.
- Click on **General**
- In the right-hand panel, click on **Additional Include Directories** and click on the **. . .** to the far-right (i.e., the button). This opens the **Additional Include Directories** panel.

- Click in this panel and type:
  `C:\Program Files\SimTK\core\include`
  Click the button to exit this panel.
  Click the to exit the Property panel.



- From within Visual C++, click on the **File** menu and then click on the **Save all** menu item.
- Click on the **File** menu and then click on the **Exit** menu item.

2.6.4 Linking your project with the SimTK libraries

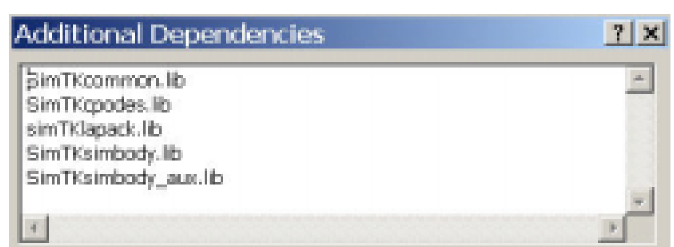
In addition to information that must be present in your **program** and available to your **compiler**, certain information must be available to your Microsoft Visual **linker**.

- If the `NewtonsApple` project is not already open, double-click on the file `NewtonsApple.sln`
- From within your Microsoft Visual C++ compiler, click on the **Project** menu, slide-down and click on **NewtonsApple Properties . . .** .
- If necessary, expand **Configuration Properties** by clicking on its + sign.
- To the right of the drop-down menu titled **Configuration:**, select **All configurations**
- If necessary, expand **Linker** by clicking on its + sign.
- Click on **General**
- In the right-hand panel, click on **Additional Library Directories** and click on the **. . .** to the far-right (i.e., the button). This opens the **Additional Library Directories** panel.

- Click in this panel and type:
  `C:\Program Files\SimTK\core\lib`
  Click the button to exit this panel.
  Go to the next step.
  Do not exit the Property panel.

- Under **Linker**, click **Input**. In the right-hand panel, click **Additional Dependencies** and click the **. . .** to the far-right (i.e., the button). This opens the **Additional Dependencies** panel.

- Click in this panel and type:
  ```
  SimTKcommon.lib
  SimTKcpodes.lib
  simTKlapack.lib
  SimTKsimbody.lib
  SimTKsimbody_aux.lib
  ```
  Click the button to exit this panel.
  Click the to exit the Property panel.



- From within Visual C++, click on the **File** menu and then click on the **Save all** menu item.
- Click on the **File** menu and then click on the **Exit** menu item.

2.6.5 Running your project with the SimTK dynamic-linked libraries

In addition to information that must be present in your **program** and available to your **compiler** and **linker**, certain information must be available when your program is **executed** so it can find the SimTK ***dynamic-linked library*** (***.dll***) files. To set the computer's ***PATH*** environment variable:

- Click on the Windows 

 menu, select **Settings**, and then **Control Panel**.
  Next, choose the **System** icon which opens up a dialogue box.
  (Note: You may access this same dialogue box by right-mouse-clicking on the  icon and selecting 

.)
- Click on the **Advanced** tab and click on the **Environment variables** button.
- Under **System Variables**, click **Path** and then click the **Edit** button.
- Ensure the **Variable value** starts with: `C:\Program Files\SimTK\core\lib;`
- Click **OK** to exit each open dialogue box.

2.6.6 Running your project with Microsoft's dynamic link libraries

Certain Microsoft ***dynamic-linked libraries*** (***.dll***) must be available when your program is **executed**. Many (or all) of these may already be installed. If you find a .dll error when your program launches, you may want to try the following:

- Go to www.simtk.org/home/training
- Click on the 

 link on the left-hand side
- Download the file `ZZMicrosoftVisualStudioDllsRequiredBySimbody.zip`
- Unzip the files to `C:\Program Files\SimTK\core\lib`.

2.6.7 Building and running the "NewtonsApple" Win32 Console Application

- If the `NewtonsApple` project is not already open, double-click on the file `NewtonsApple.sln`
- Click on the **Build** menu and then slide-down and click on **Build Solution**
- The compiler will attempt to ***compile*** the relevant C++ ***source files***.
  For example, the source file `NewtonsApple.cpp` will compile to the ***object file*** named `NewtonsApple.obj`
- You must fix all compiler errors (bugs in your program) before you can ***link***.
  It is highly advisable to also fix all your warnings as many of them will show up later as run-time errors.
- If compiling all the source files is successful, Visual C++ will attempt to link your object files to each other and to the standard C++ libraries.[4]
  You may see compile and/or link errors if you are trying to use the Simbody libraries and have not informed the compiler and/or linker of the existence and location of certain .h, .lib, and .dll files. See Section 2.6.4 for information.
- If all goes well, you may see a message such as

```
------ Build started: Project: NewtonsApple, Configuration: Debug Win32 ------
Compiling...
NewtonsApple.cpp
Compiling manifest to resources...
Linking...
Embedding manifest...
Build log was saved at "file://c:\Program Files\SimTK\NewtonsApple\Debug\BuildLog.htm"
NewtonsApple - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

- To run the program from within the Visual C++ environment, click on the **Build** menu and select **Start Debugging**
- If you encounter a run-time debug error (the program runs but it produces incorrect results), try the Visual C++ ***debugger***. To start the debug process, do the following:
  - Get help (preferably human) to understand the debugging process
  - Ensure that the build is set to **Win32 Debug** (not **Win32 Release**)
  - Insert a ***breakpoint*** in your code by right-mouse clicking on a suspect line in your code.
    If in doubt, set the breakpoint at the first line of code in **main** (the line that appears after the first { in **main**).
  - Click on the **Debug** menu and then slide-down and click on **Start Debugging**.
  - Press function key **F10** to step over a function/method
  - Press function key **F11** to step into a function/method
  - Press function key **F5** to go to the next breakpoint

2.6.8 Deploying your executable programs to end-users (who are not developers)

- Create a "release" version of the program.
  In the top-middle of Microsoft Visual Studio, select **Release**
- Ensure the end-user has the executable program (e.g., NewtonsApple.exe)
- Ensure the end-user has the SimTK libraries (.dlls and .libs) in the same directory as NewtonsApple.exe. Also ensure the end-user has the appropriate Microsoft .dlls. (see Section 2.6.6).
  Alternately, place the SimTK libraries in a directory named in the computer's PATH environment variable
  Note: The end-user does not need header files (.h files) or .lib files - these are only useful for developers.
- Alternately, link with the SimTK static libraries and deploy as a large, single executable.

[4] The standard C++ libraries include the standard math library that calculates cos(0.2), the standard input/output library that prints characters to the screen, and the standard time library that gets time and date information from your computer.

2.7 Sample NewtonsApple.cpp

```
//------------------------------------------------------------------------------
// File:     NewtonsApple.cpp
// Class:    None
// Parent:   None
// Children: None
// Purpose:  Simulates Newton's falling apple
//------------------------------------------------------------------------------
// The following are standard C/C++ header files.
// If a filename is enclosed inside < >  it means the header file is in the Include directory.
// If a filename is enclosed inside " "  it means the header file is in the current directory.
#include <ctype.h>      // Character Types
#include <math.h>       // Mathematical Constants
#include <stdarg.h>     // Variable Argument Lists
#include <stdio.h>      // Standard Input/Output Functions
#include <stdlib.h>     // Utility Functions
#include <string.h>     // String Operations
#include <signal.h>     // Signals (Contol-C + Unix System Calls)
#include <setjmp.h>     // Nonlocal Goto (For Control-C)
#include <time.h>       // Time and Date information
#include <assert.h>     // Verify Program Assertion
#include <errno.h>      // Error Codes (Used in Unix system())
#include <float.h>      // Floating Point Constants
#include <limits.h>     // Implementation Constants
#include <stddef.h>     // Standard Definitions
#include <exception>    // Exception handling (e.g., try, catch throw)
//------------------------------------------------------------------------------
#include "SimTKsimbody.h"
using namespace SimTK;
using namespace std;
//------------------------------------------------------------------------------


//------------------------------------------------------------------------------
// Prototypes for local functions (functions not called by code in other files)
//------------------------------------------------------------------------------
bool  SimulateNewtonsApple( void );
bool  WriteStringToFile(   const char outputString[], FILE *fptr )  { return fputs( outputString, fptr ) != 0; }
bool  WriteStringToScreen( const char outputString[] )              { return WriteStringToFile( outputString, stdout
bool  WriteDoubleToFile( double x, int precision, FILE *fptr );
FILE*  FileOpenWithMessageIfCannotOpen( const char *filename, const char *attribute );


//------------------------------------------------------------------------------
// The executable program starts here
//------------------------------------------------------------------------------
int  main( int numberOfCommandLineArguments, char *arrayOfCommandLineArguments[] )
{
   // Simulate the multibody system
   bool simulationSucceeded = SimulateNewtonsApple();

   // Keep the screen displayed until the user presses the Enter key
   WriteStringToScreen( "\n\n Press  Enter  to terminate the program: " );
   getchar();

   // The value returned by the main function is the exit status of the program.
   // A normal program exit returns 0 (other return values usually signal an error).
   return simulationSucceeded == true ? 0 : 1;
}



//------------------------------------------------------------------------------
bool  SimulateNewtonsApple( void )
{
   // Declare a multibody system (contains one or more force and matter sub-systems)
   MultibodySystem  mbs;

   // 0. The ground's right-handed, orthogonal x,y,z unit vectors are directed with x horizontally right and y verti
   // 1. Create a gravity vector that is straight down (in the ground's frame)
   // 2. Create a uniform gravity sub-system
   // 3. Add the gravity sub-system to the multibody system
   Vec3 gravityVector( 0, -9.8, 0 );
   UniformGravitySubsystem gravity( gravityVector );
   mbs.addForceSubsystem( gravity );

   // Create a matter sub-system (the apple)
   SimbodyMatterSubsystem  apple;
```

```
// Create the mass, center of mass, and inertia properties for the apple
const Real  massOfApple = 0.142;

// The location of the apple's center of mass is a vector from the apple's
// origin expressed in the "x, y, z"' unit vectors fixed in the apple's frame.
// Example: The vector(0,0,0) locates the apple's center of mass at the apple's origin.
// Example: The vector(1,0,0) locates the apple's center of mass 1 unit in the "x" direction from the apple's origin.
const Vec3  appleCenterOfMassLocation( 0, 0, 0 );

// Create the apple's inertia matrix about its origin for the "x, y, z" unit vectors fixed in the apple's frame.
// Note: The 3x3 inertia matrix is symmetric - so only 6 elements need to be defined.
// Ixx, Iyy, Izz  are moments  of inertia (    diagonal terms in the matrix)
// Ixy, Ixz, Iyz  are products of inertia (off-diagonal terms in the matrix)
// The following assumes an apple of radius 1.44 inches (3.6576 cm) with a radius of gyration of 0.91 inches (2.3114 c
// which approximates a perfect sphere of radius 1.44 inches (3.6576 cm)
const Real I = massOfApple * pow( 0.023114, 2 );  // Inertia = mass * radiusOfGyration^2
const Real  Ixx = I,  Iyy = I,   Izz = I;
const Real  Ixy = 0,  Ixz = 0,   Iyz = 0;
const Inertia  appleInertiaMatrix( Ixx, Iyy, Izz, Ixy, Ixz, Iyz );

// The MassProperties class holds the mass, center of mass, and inertia properties of a rigid body.
// Although the next line creates an instance of the MassProperties class for the apple,
// it does not get associated with the apple until the  addRigidBody  method.
MassProperties  appleMassProperties( massOfApple, appleCenterOfMassLocation, appleInertiaMatrix );

// The apple's motion is related to ground via "mobilizers".
// The "mobilizers" specify the allowable motion of the apple to the ground.
// More specifically, the motion of an "outboard body" (e.g., the apple)
// to its inboard body (e.g., the ground) is specified by first constructing:
// 1. An  "outboard frame"  on the ground (which hooks to the "outboard body" - the apple )
// 2. An  "inboard  frame"  on the apple  (which hooks to the "inboard  body" - the ground)

// The orientation and position of the  outboard frame  from the ground's frame is specified below.
// The outboard frame's axes are aligned with the ground's axes and its origin is coincident with the ground's origin.
// In other words, for this simple problem the outboard frame and the ground frame are identical.
const Transform outboardFrameTransformFromGround;  // The default constructor is the identity transform

// The orientation and position of the  inboard  frame  from the apple's  frame is specified below.
// The inboard frame's  axes are aligned with the apple's  axes and its origin is coincident with the apple's  origin
// In other words, for this simple problem the inboard  frame and the apple frame are identical.
// Although the inboard frame can be constructed in a simple manner analogous to the outboardFrameTransformFromGround
// it is worthwhile to look at the details of the rotation matrix and position vector in the transform:
// a. The rotation matrix relating the InboardFrame's x,y,z axes to the AppleFrame's x,y,z axes is specified InboardFr
// b. The position of the InboardFrame's origin from the AppleFrame origin, expressed in terms of the AppleFrame's "x,
const Rotation   inboardFrameOrientationInApple;   // ( 1,0,0, 0,1,0, 0,0,1 );
const Vec3       inboardFrameOriginLocationFromAppleOrigin( 0, 0, 0 );
const Transform  inboardFrameTransformFromApple( inboardFrameOrientationInApple, inboardFrameOriginLocationFromAppleOr

// There are many ways that the apple can move relative to the ground.
// The following allows the apple to move in "x", "y", and "z" directions.
// Another option producing the same result is Mobilizer::Free
Mobilizer appleToGroundMobilizer = Mobilizer::Cartesian;
const BodyId appleBodyId = apple.addRigidBody( appleMassProperties, inboardFrameTransformFromApple, GroundId, outboard

// Add the matter (apple) sub-system to the system.
mbs.setMatterSubsystem( apple );

// Create a state for this system.
// Define appropriate states for this multi-body system.
// Set the initial time to 0.0
State s;
mbs.realize( s );
s.setTime( 0.0 );

// Set the initial values for the configuration variables (x,y,z)
apple.setMobilizerQ( s, appleBodyId, 0,  0.0 );
apple.setMobilizerQ( s, appleBodyId, 1, 10.0 );
apple.setMobilizerQ( s, appleBodyId, 2,  0.0 );

// Set the initial values for the motion variables
apple.setMobilizerU( s, appleBodyId, 0,  0.0 );
apple.setMobilizerU( s, appleBodyId, 1,  0.0 );
apple.setMobilizerU( s, appleBodyId, 2,  0.0 );

// Create a study using the Runge Kutta Merson integrator (alternately use the CPodesIntegrator)
RungeKuttaMerson myStudy( mbs, s );

// Set the numerical accuracy for the integrator
myStudy.setAccuracy( 1.0E-7 );
```

```
   // The next statement does lots of accounting
   myStudy.initialize();

   // Open a file to record the simulation results (they are also displayed on screen)
   FILE *outputFile = FileOpenWithMessageIfCannotOpen( "NewtonsAppleResults.txt", "w" );
   WriteStringToFile(   "time      yLocation    yVelocity  mechanicalEnergy  yDifferenceFromExact\n",  outputFile );
   WriteStringToScreen( "time      yLocation    yVelocity  mechanicalEnergy  yDifferenceFromExact\n" );

   // Set the numerical integration step and the time for the simulation to run
   const Real integrationStepDt = 0.01;
   const Real finalTime = 4.0;
   const Real finalTimeCompare = finalTime - 0.01*integrationStepDt;

   // Run the simulation and print the results
   while( 1 )
   {
      // Query for results to be printed
      Real time = s.getTime();
      Real kineticEnergy = mbs.getKineticEnergy(s);
      Real uniformGravitationalPotentialEnergy = mbs.getPotentialEnergy(s);
      Real mechanicalEnergy = kineticEnergy + uniformGravitationalPotentialEnergy;

      // Locate the apple origin's from the ground's origin, expressed in terms of the ground's "x,y,z" unit vectors
      // Extract the apple's y-location from this vector.
      const Vec3 appleLocation = apple.calcBodyOriginLocationInBody( s, appleBodyId, GroundId );
      Real yLocation = appleLocation[1];

      // Get the apple origin's velocity in ground, expressed in terms of the ground's "x,y,z" unit vectors.
      // Extract the apple's y-velocity from this vector.
      const Vec3 appleVelocity = apple.calcBodyOriginVelocityInBody( s, appleBodyId, GroundId );
      Real yVelocity = appleVelocity[1];

      // Get the apple origin's acceleration in ground, expressed in terms of the ground's "x,y,z" unit vectors.
      // Extract the apple's y-acceleration from this vector.
      // const Vec3 appleAcceleration = apple.calcBodyOriginAccelerationInBody( s, appleBodyId, GroundId );
      // Real yAcceleration = appleAcceleration[1];

      // Exact analytical results and difference of numerical integration results
      double yExact = 10 - 4.9*time*time;
      double yDifferenceFromExact = yExact - yLocation;

      // Print results to screen
      WriteDoubleToFile( time,                 2, stdout );
      WriteDoubleToFile( yLocation,            4, stdout );
      WriteDoubleToFile( yVelocity,            4, stdout );
      WriteDoubleToFile( mechanicalEnergy,     7, stdout );
      WriteDoubleToFile( yDifferenceFromExact, 7, stdout );
      WriteStringToScreen( "\n" );
      // Print results to file
      WriteDoubleToFile( time,                 2, outputFile );
      WriteDoubleToFile( yLocation,            4, outputFile );
      WriteDoubleToFile( yVelocity,            4, outputFile );
      WriteDoubleToFile( mechanicalEnergy,     7, outputFile );
      WriteDoubleToFile( yDifferenceFromExact, 7, outputFile );
      WriteStringToFile( "\n", outputFile );

      // Check if integration has completed
      if( time >= finalTimeCompare ) break;

      // Increment time step
      myStudy.step( time + integrationStepDt );
   }

   // Simulation completed properly
   return true;
}


//------------------------------------------------------------------------------
FILE*  FileOpenWithMessageIfCannotOpen( const char *filename, const char *attribute )
{
   // Try to open the file
   FILE *Fptr1 = fopen( filename, attribute );

   // If unable to open the file, issue a message
   if( !Fptr1 )
   {
      WriteStringToScreen( "\n\n Unable to open the file: " );
      WriteStringToScreen( filename );
      WriteStringToScreen( "\n\n" );
```

```
   }

   return Fptr1;
}


//-----------------------------------------------------------------------------
bool  WriteDoubleToFile( double x, int precision, FILE *fptr )
{
   // Ensure the precision (number of digits in the mantissa after the decimal point) makes sense.
   // Next, calculate the field width so it includes one extra space to the right of the number.
   if( precision < 0 || precision > 17 ) precision = 5;
   int fieldWidth = precision + 8;

   // Create the format specifier and print the number
   char format[20];
   sprintf( format, " %%- %d.%dE", fieldWidth, precision );
   return fprintf( fptr, format, x ) >= 0;
}
```

2.8 Optional**: Description of SimTK and VTK libraries

Library	Description
VTK	***VTK*** is a **free**, "easy-to-use" ***V***isualization ***T***ool***K***it for 3D computer graphics and image processing. VTK supports many visualization algorithms, including scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as implicit modeling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangularition. Users can mix 2D imaging with 3D graphics and data. VTK runs on most computers (Windows 98/ME/NT/2000/XP, Mac OSX, and Unix platforms). More information about VTK is available at www.vtk.org.
SimTKlapack	***Lapack*** is a highly efficient ***L***inear ***a***lgebra ***pack***age. Lapack runs on most computers and is optimized for a computer's processor (optimizing for cache, instruction set, number of processors, etc.). More information about Lapack is available at www.simtk.org/home/lapack and at www.netlib.org/lapack.
SimTKsimmath	***SimTKsimmath*** is a library of classes, methods, and functions for nonlinear optimization, numerical differentiation, numerical integration, polynomial interpolation, random number generation, etc. More information about SimTKsimmath is available at www.simtk.org/home/simmath.
SimTKcommon	***SimTKcommon*** is a library of SimTK classes, methods, functions, and data that are common to many of the SimTK libraries. This library holds physical constants, exception and error checking, macros for handling binary capabilities, and compiler dependencies. It also contains an easy-to-use C^{++} interface to SimTK Lapack that efficiently and naturally handles arrays and matrices. More information about SimTKcommon is available at www.simtk.org/home/simtkcommon.
SimTKcpodes	***SimTKcpodes*** is a highly-efficient, high-order, multi-step, numerical integrator that is capable of integrating stiff systems. ***CPODES*** is a branch of the better-known ***CVODES*** integrator (http://acts.nersc.gov/sundials) that provides an accuracy-enhancing method for enforcing algebraic equations associated with differential algebraic equations (DAEs). (Algebraic equations arise in multibody systems due to closed loops, rolling, sum-squared of Euler parameters, conservation of energy/momentum, etc.) More information about SimTKcpodes is available at www.simtk.org/home/cpodes.
SimTKsimbody	***SimTKsimbody*** is an efficient multibody simulator that manages systems (e.g., force and mass subsystems), states (e.g., variables being integrated), and studies (e.g., statics, dynamics, optimization, etc.). More information about Simbody is available at www.simtk.org/home/simbody

2.9 Optional**: Developer instructions for packaging SimTK libraries

The following steps were used in April 2007 to package the SimTK libraries for Windows XP. At that time, all the libraries depended on the ***Operating System*** (e.g., Windows, Macintosh or Linux), some depended on the ***CPU*** (***central processing unit***) (e.g., Intel or AMD), and some depended on other installed libraries and header files.

Library	Dependencies	Website
VTK		www.vtk.org
SimTKlapack	CPU	www.simtk.org/home/lapack
SimTKcommon	SimTKlapack	www.simtk.org/home/simtkcommon
SimTKsimmath	SimTKcommon, SimTKlapack	www.simtk.org/home/simmath
SimTKcpodes	SimTKcommon, SimTKlapack	www.simtk.org/home/simtkcpodes
SimTKsimbody	SimTKcommon, SimTKlapack, SimTKsimmath, SimTKcpodes	www.simtk.org/home/simbody

2.9.1 Installing CMake

CMake reads a generic ASCII text file which contains instructions on how to compile and link source code, libraries, and object files and generates compiler-specific files (e.g., .dsw files for older Microsoft Visual Studio C++ compilers, .sln files for newer Microsoft Visual Studio C++ compilers, or makefiles for gcc compilers, etc.) **CMake** is helpful for building various SimTK libraries, including SimTKSimmatrix, SimTKcommon, and SimTKSimbody.

- Go to www.cmake.org and click on the **Download** link on the left-hand side
- **Download** the latest **CMake** installer for your computer, e.g., `cmake-2.4.6-win32-x86.exe`
- Double-click on the downloaded installer .exe file and follow the on-screen instructions.
  (Accept the default installation options.)
- Optional**: Delete the downloaded **CMake** installer .exe file to free up some disk space

2.9.2 Installing VTK

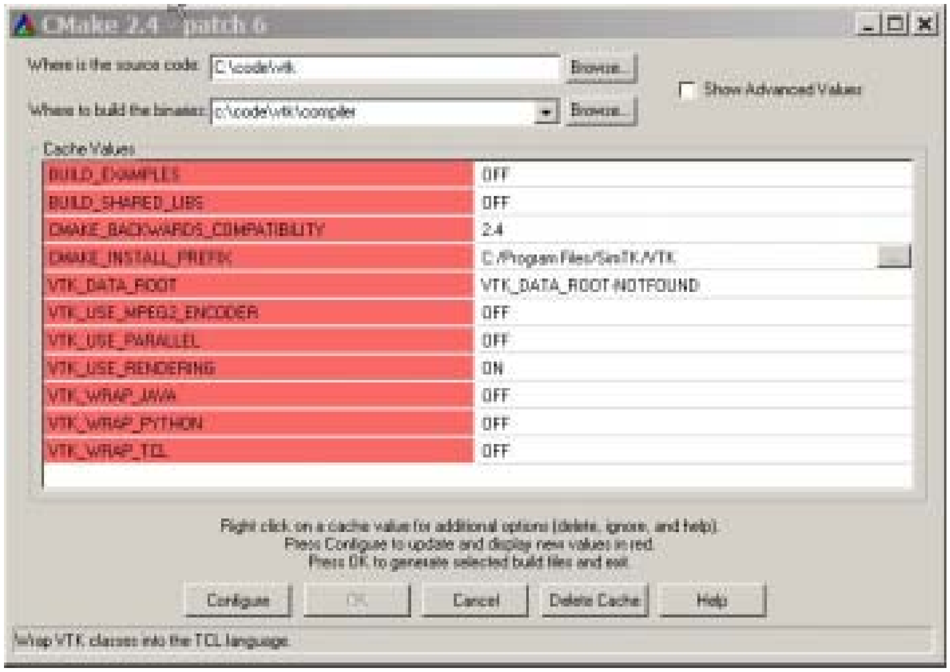
Downloading and installing VTK to your computer

- Go to www.vtk.org and click on the **Download** link on the left-hand side
- Click on **Download the latest release**
- Click on (and download) the appropriate **source** .zip file, e.g., `vtk-5.0.3.zip`
- Right-mouse-click on the downloaded .zip file and select **Extract all**
- Click **Next**, select the destination `C:\Program Files\SimTK`, and click **Next**
- Follow the on-screen and web-based instructions to finish the installation

Using CMake to create instructions for your C++ compiler

- Click on Windows 

, select **Programs**, then **CMake** folder, then **CMake** executable
- Change the information in the **CMake** fields to:
  Where is the source code: `c:\code\vtk`
  Where to build the binaries: `c:\code\vtk\compiler`
  Enter the rest of the information as follows:



- Click **Configure**
- From the drop-down menu, select **Visual Studio 8 2005** (or your installed compiler) and click **OK**
  If necessary, allow **CMake** to create the directory `c:\code\vtk\compiler` by clicking **OK**.
- **Wait** until a new panel appears
- Click **Configure** again, then click **OK** to exit **CMake**.
  This creates files containing instructions for the Microsoft Visual Studio 8 2005 compiler.

Building source code and installing library files

- If you selected Microsoft Visual Studio 8 2005, double click on the file:
  `c:\code\vtk\compiler\vtk.sln`
- In the top-middle of Microsoft Visual Studio, select **Release**.
- Right-mouse-click on **ALL_BUILD** and select **Build**
  This compiles, links, and builds the VTK library files.
- Right-mouse-click on **INSTALL** and select **Build**
  This copies the library and header files to the proper directories.
- Exit from your compiler

2.9.3 Installing SimTKlapack

Before installing Lapack, you need information about your computer's CPU which is available by clicking on the Windows Start menu, selecting **Settings**, and then **Control Panel**. Next, choose the **System** icon which opens up a dialogue box. Information about your computer's operating system and CPU are displayed under the General tab. Click Cancel when you are finished.
(Note: CPU information may be available by right-mouse-clicking on the  icon and selecting Properties.)

Downloading SimTK Lapack to your computer

- Go to www.simtk.org/home/lapack and click on the Downloads link on the left-hand side
- For a 32-bit Windows computer with an Intel processor, **download** the file
  `Simtklapack_windows_32bit_Intel_generic.exe`
- For a 32-bit Windows computer with an AMD processor, **download** the file
  `Simtklapack_windows_32bit_AMD_generic.exe`

Installing SimTK Lapack to your computer

- Double-click on the downloaded file and then unzip to (possible by clicking the **Browse** button) `C:\Program Files\SimTK`
- This installs the following files to the following folders:

File	Folder	Comment
`SimTKlapack.dll`	`C:\Program Files\SimTK\core\lib`	
`SimTKlapack.lib`	`C:\Program Files\SimTK\core\lib`	
`SimTKlapack.h`	`C:\Program Files\SimTK\core\include`	
`Examples`	`C:\Program Files\SimTK\examples`	Not complete or necessary
`Documentation`	`C:\Program Files\SimTK\doc`	Not complete or necessary

2.9.4 Installing SVN

SVN is a command-line program which helps "get" and "check-in" source code from web-based repositories. SVN is required to get source-code from www.simtk.org to your computer.

- Go to www.svn.org and **download** and install the latest version of SVN for your computer
- Note: Windows users typically prefer ***Tortoise SVN*** because it has an easy-to-use graphical user interface. To installer Tortoise SVN (instead of plain SVN), do the following:
  - Go to www.tortoisesvn.net
  - Click on the **Download** link on the left-hand side
  - Download the installer for your computer (e.g., 32-bit or 64-bit version)
  - Double-click on the downloaded installer .msi file and follow the on-screen instructions. (Accept the default installation options.)
  - You may have to restart your computer to finish the installation
  - Optional**: Delete the downloaded Tortoise SVN installer .msi file to free up disk space

2.9.5 Installing SimTKcommon

Downloading source code to your computer

- Create a directory `c:\code\simtkcommon`
- Right-mouse-click in that empty directory and select **SVN checkout**
- Enter the URL of repository as: `https://simtk.org/svn/simtkcommon/trunk` and click **OK**
- After the files download to your computer, click **OK** to finish

Using CMake to create instructions for your C^{++} compiler

- Click on the Windows  menu, select **Programs**, then **CMake** folder, then **CMake** executable
- Change the information in the **CMake** fields to:
  Where is the source code: `c:\code\simtkcommon`
  Where to build the binaries: `c:\code\simtkcommon\compiler`
- Click **Configure**
- From the drop-down menu, select **Visual Studio 8 2005** (or your installed compiler) and click **OK**
  If necessary, allow **CMake** to create the directory `c:\code\simtkcommon\compiler` by clicking **OK**.
- **Wait** until a new panel appears
- Click **Configure** again, then click **OK** to exit **CMake**.
  This creates files containing instructions for the Microsoft Visual Studio 8 2005 compiler.

Building source code and installing library files

- If you selected Microsoft Visual Studio 8 2005, double click on the file: `c:\code\simtkcommon\compiler\simtkcommon.sln`
- In the top-middle of Microsoft Visual Studio, select **Release**.
- Right-mouse-click on **ALL_BUILD** and select **Build**
  This compiles, links, and builds the simtkcommon library files.
- Right-mouse-click on **INSTALL** and select **Build**
  This copies the library and header files to the proper directories.
- Exit from your compiler

This installs the following files to the following folders:

File	Folder
`SimTKcommon.dll`	`C:\Program Files\SimTK\core\lib`
`SimTKcommon.lib`	`C:\Program Files\SimTK\core\lib`
`SimTKcommon.h`	`C:\Program Files\SimTK\core\include`

2.9.6 Installing SimTKsimmath, SimTKcpodes, and SimTKsimbody

The installation instruction for various SimTK libraries are similar to the installation instructions for SimTKcommon in Section 2.9.5. The following table shows differences in installation of the various libraries.

Library name and usage	Source code directory	URL of repository
SimTKsimmath	`c:\code\simtksimmath`	`https://simtk.org/svn/simmath/trunk`
SimTKcpodes	`c:\code\simtkcpodes`	`https://simtk.org/svn/cpodes/trunk`
SimTKsimbody	`c:\code\simtksimbody`	`https://simtk.org/svn/simbody/trunk`

Downloading source code to your computer

- Create a source code directory, e.g., `c:\code\simtksimmath`
- Right-mouse-click in that empty directory and select **SVN checkout**
- Enter the URL of repository as: `https://simtk.org/svn/simmath/trunk` and click **OK**
- After the files download to your computer, click **OK** to finish

Using CMake to create instructions for your C^{++} compiler

- Click on the Windows  menu, select **Programs**, then **CMake** folder, then **CMake** executable
- Change the information in the **CMake** fields, e.g., for SimTKsimmath to:
  Where is the source code: `c:\code\simtksimmath`
  Where to build the binaries: `c:\code\simtksimmath\compiler`
- Click **Configure**
- From the drop-down menu, select **Visual Studio 8 2005** (or your installed compiler) and click **OK**
  If necessary, allow **CMake** to create the directory `c:\code\simtksimmath\compiler` by clicking **OK**.
- **Wait** until a new panel appears
- Click **Configure** again, then click **OK** to exit **CMake**.
  This creates files containing instructions for the Microsoft Visual Studio 8 2005 compiler.

Building source code and installing library files

- Double-click on the appropriate Microsoft Visual Studio file, e.g., for SimTKsimmath: `c:\code\simtksimmath\compiler\simtksimmath.sln`
- In the top-middle of Microsoft Visual Studio, select **Release**.
- Right-mouse-click on **ALL_BUILD** and select **Build**
- Right-mouse-click on **INSTALL** and select **Build**
- Exit from your compiler

The table below shows the installed files and associated folders:

File	Folder
`simmath.dll`	`C:\Program Files\SimTK\core\lib`
`simmath.lib`	`C:\Program Files\SimTK\core\lib`
`simmath.h`	`C:\Program Files\SimTK\core\include`
`SimTKcpodes.dll`	`C:\Program Files\SimTK\core\lib`
`SimTKcpodes.lib`	`C:\Program Files\SimTK\core\lib`
`SimTKcpodes.h`	`C:\Program Files\SimTK\core\include`
`SimTKsimbody.dll`	`C:\Program Files\SimTK\core\lib`
`SimTKsimbody.lib`	`C:\Program Files\SimTK\core\lib`
`SimTKsimbody.h`	`C:\Program Files\SimTK\core\include`