```
Function Velocity(Vox, ax, t)
Velocity = Vox + ax * t
End Function

Function Position(Xo, Vox, ax, t)
Position = Xo + Vox * t + 0.5 * ax * t ^ 2
End Function
```

Notice that the **Position** and **Velocity** function definitions appear to involve the variable "**t**" and not "**deltaT**"as required for incremental iteration. However, remember that function arguments are "passed by value" meaning that the value of the variable in the argument is transferred, which in the "calling" statement in the main **Do…Loop** is the value of **deltaT**. The use of "**t**" in the function definition is only a name, and not necessarily the actual time.

6.4.2 The On-the-Fly Approach

Application # 14 : Interactive Rocket Car with Variable Thrust

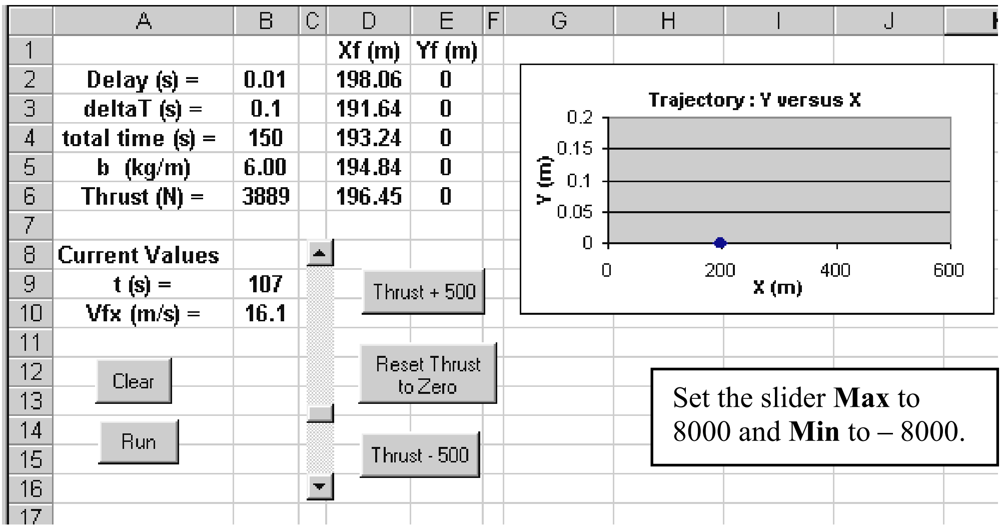
Two methods of changing the rocket engine **Thrust** of **Application # 8** on-the-fly are presented here : using a scroll bar, and using buttons. The scroll bar approach is marginally simpler and requires that the code statement : **Thrust = scbThrust.Value** be included either in the scroll bar subroutine (as shown below), or in the main loop (before **Fnetx** is calculated). Every time the slider is changed the value of **Thrust** will therefore be changed. In fact, if the statement is included in the main loop the entire scroll bar subroutine could be omitted.

While the scroll bar approach allows the **Thrust** to be set to any value in the range between the maximum and minimum values of the scrollbar, the command button approach can only change the **Thrust** in the quantum increments programmed in the button. For example, the code statement : **Thrust = Thrust + 500** will increase the **Thrust** by 500 Newtons each time the button is pressed.

```
Private Sub cmdThrustPlus500_Click()
Thrust = Thrust + 500
Worksheets("sheet1").Range("B6") = Thrust
End Sub
```

The new value of **Thrust** is displayed on the worksheet.

A button to reduce the thrust would also be required (to created negative values, which act to the left).



	A	B	C	D	E
1				Xf (m)	Yf (m)
2	Delay (s) =	0.01		198.06	0
3	deltaT (s) =	0.1		191.64	0
4	total time (s) =	150		193.24	0
5	b (kg/m)	6.00		194.84	0
6	Thrust (N) =	3889		196.45	0
7					
8	Current Values				
9	t (s) =	107			
10	Vfx (m/s) =	16.1			

Set the slider **Max** to 8000 and **Min** to – 8000.

Application # 8 (Rocket Car) was used as the shell for this prog-ram; additions are shown bold.

```
Option Explicit
Dim Delay As Double, Thrust As Double

Private Sub cmdClear_Click()
 Worksheets("sheet1").Range("D2:E6") = 0
 Worksheets("sheet1").Range("B9:B10") = 0
 scbThrust.Value = 0
End Sub

Private Sub cmdRun_Click()
Dim Xo As Double, Vox As Double, Xf As Double, Vfx As Double, mass As Double
Dim ax As Double, t As Double, deltaT As Double, Fnetx As Double
Dim row As Integer, Total_time As Double
Dim b As Double, JJ As Integer

Delay = Worksheets("sheet1").Range("B2")
deltaT = Worksheets("sheet1").Range("B3")
Total_time = Worksheets("sheet1").Range("B4")
b = Worksheets("sheet1").Range("B5")
Thrust = scbThrust.Value
```

This line is only necessary to define the thrust in the event that the user does not push the **Clear** button, and does not adjust the scroll bar,

```
mass = 2000: Xo = 0: Vox = 0
row = 2
t = 0
```

Alternative location for **Thrust = scbThrust.Value** (just after the **Do**) to avoid the need for a scroll bar subroutine.

```
Do
   Fnetx = Thrust – Sgn(Vox) * b * Vox ^ 2
   ax = Fnetx / mass
```

Application # 8, did not allow the reversal of drag direction using : **– Sgn(Vox)**

```
   Xf = Xo + Vox * deltaT + 0.5 * ax * deltaT ^ 2
   Vfx = Vox + ax * deltaT

   Xo = Xf
   Vox = Vfx
   t = t + deltaT

   Call TimeDelay(Delay)
   Worksheets("sheet1").Cells(row, 4) = Xf
   Worksheets("sheet1").Cells(row, 5) = 0
   Worksheets("sheet1").Range("B9") = t
   Worksheets("sheet1").Range("B10") = Vfx
```

Included to provide the user with additional information concerning the motion.

```
   For JJ = 1 To 40
   DoEvents: DoEvents
   Next JJ

   row = row + 1
   If row = 7 Then row = 2

Loop Until t >= Total_time

End Sub
```

```
Private Sub cmdReset_Click()
Thrust = 0
scbThrust.Value = 0
Worksheets("sheet1").Range("B6") = Thrust
End Sub
```

The **Reset** button resets the **Thrust** and the scroll bar to zero, which allows the car to be decelerated by air resistance .

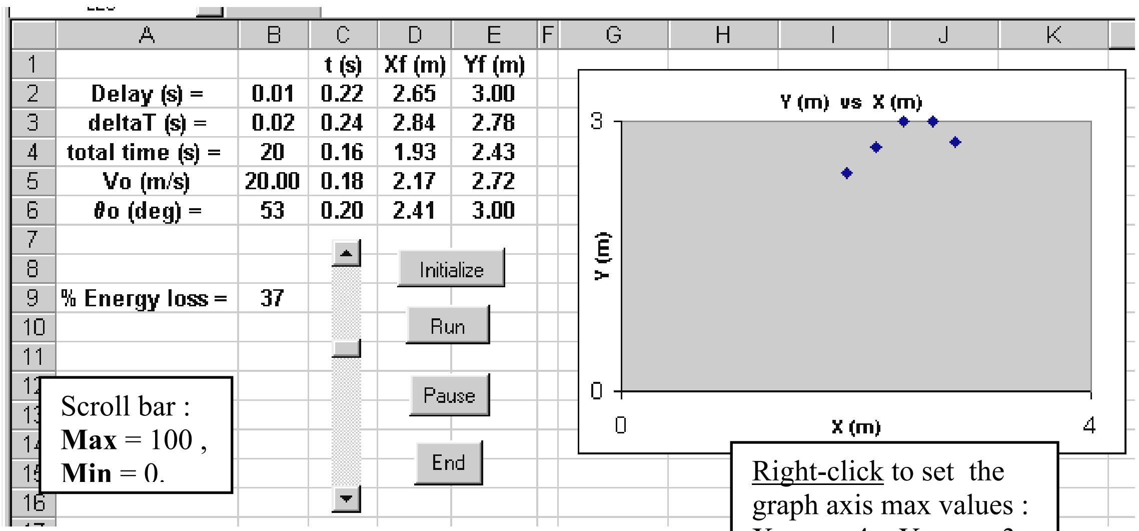
```
Private Sub cmdThrustMinus500_Click()
Thrust = Thrust – 500
Worksheets("sheet1").Range("B6") = Thrust
End Sub

Private Sub cmdThrustPlus500_Click()
Thrust = Thrust + 500
Worksheets("sheet1").Range("B6") = Thrust
End Sub

Private Sub scbThrust_Change()
Thrust = scbThrust.Value
Worksheets("sheet1").Range("B6") = Thrust
End Sub
```

Set the slider **Max** to 8000 and **Min** to – 8000.

During **On-the-Fly** changes the program continues executing around the main **Do** loop. With **Pause** changes, the change of state of **bStop** produced by pressing the **Pause** button is detected, and the main loop is exited, as well as the **Run** subroutine; program execution is complete and ceases, but certain variables retain their values in case the user decides to resume the simulation.

Application # 15 : Ball-In-A-Box (Combination Pause & On-the-Fly Changes)

This application simulates a ball bouncing around in a closed box under the influence of gravity. If no energy is lost the ball will bounce around forever. A **scroll bar** is provided to vary the amount of energy (really velocity) lost with each collision. Collisions are modelled by detecting if the ball is about to move outside the boundaries of the box (using, naturally, an **If** statement) and reversing the relevant velocity component. For example, if the ball is about to cross the wall at X = 4, the X component of velocity has its sign reversed to represent the rebound off the wall; the Y component is unaffected in this case. The energy loss occurs by reducing both Vox and Voy by the same factor, which is not necessarily realistic. The program allows the user the option of making either **On-the-Fly** changes or **Pause** changes.



	A	B	C	D	E
1			t (s)	Xf (m)	Yf (m)
2	Delay (s) =	0.01	0.22	2.65	3.00
3	deltaT (s) =	0.02	0.24	2.84	2.78
4	total time (s) =	20	0.16	1.93	2.43
5	Vo (m/s)	20.00	0.18	2.17	2.72
6	θo (deg) =	53	0.20	2.41	3.00
7					
8					
9	% Energy loss =	37			
10					
11					

Scroll bar :
Max = 100 ,
Min = 0.

Right-click to set the graph axis max values :
Xmax = 4., Ymax = 3.

```
Option Explicit
Dim Delay As Double, bStop As Boolean, row As Integer
Dim Xo As Double, Vox As Double
Dim Yo As Double, Voy As Double
Dim Vo As Double, Theta As Double, t As Double
```

```
Private Sub cmdEnd_Click()
End
End Sub
```

The program is a modification of the 2-D projectile : **Application # 13**

```
Private Sub scbLoss_Change()
Worksheets("sheet1").Range("B9") = scbLoss.Value
End Sub
```

```
Private Sub cmdRun_Click()
Dim Xf As Double, Vfx As Double, ax As Double, Fnetx As Double
Dim Yf As Double, Vfy As Double, ay As Double, Fnety As Double
Dim deltaT As Double, Total_time As Double, mass As Double
Dim W As Double, JJ As Integer, kk As Double

Delay = Worksheets("sheet1").Range("B2")
deltaT = Worksheets("sheet1").Range("B3")
Total_time = Worksheets("sheet1").Range("B4")
```

Compare how **kk** is declared and defined here with **Thrust** of **Application # 14**. **kk** is not declared globally since it is not used in the **scbLoss( )** code, and it's value is determined below when needed.

```
mass = 1
W = mass * 9.81
bStop = False
ax = 0

Do
   Fnety = –W
   ay = Fnety / mass

   Xf = Position(Xo, Vox, ax, deltaT)
   Yf = Position(Yo, Voy, ay, deltaT)
   Vfx = Velocity(Vox, ax, deltaT)
   Vfy = Velocity(Voy, ay, deltaT)

   Worksheets("sheet1").Range("B9") = scbLoss.Value
```

Should be redundant but occasionally the new value is not written on the worksheet by the **scbLoss( )** code

```
    ' Determine retained velocity factor "kk" for collision from scroll bar energy loss %
   kk = 1 – (scbLoss.Value / 100)
   kk = Sqr(kk)    ' The Sqr accounts for KE depending on V^2

   ' Detect wall collision, reverse relevant velocity component, multiply by retained
   ' velocity factor kk,  adjust initial position to prevent being trapped outside the box.

   If Xf >= 4 Then Vfx = –kk * Vfx: Vfy = kk * Vfy: Xf = 4
   If Xf <= 0 Then Vfx = –kk * Vfx: Vfy = kk * Vfy: Xf = 0
   If Yf >= 3 Then Vfy = –kk * Vfy: Vfx = kk * Vfx: Yf = 3
   If Yf <= 0 Then Vfy = –kk * Vfy: Vfx = kk * Vfx: Yf = 0
```

The ball is shifted back to the wall to prevent it from being trapped outside the box.

```
   Xo = Xf
   Yo = Yf
   Vox = Vfx
   Voy = Vfy
   t = t + deltaT

   Call TimeDelay(Delay)
   Worksheets("sheet1").Cells(row, 3) = t
   Worksheets("sheet1").Cells(row, 4) = Xf
   Worksheets("sheet1").Cells(row, 5) = Yf

      For JJ = 1 To 50
      DoEvents: DoEvents
      Next JJ

   row = row + 1
   If row = 7 Then row = 2

Loop Until t >= Total_time Or bStop = True

End Sub
```

```
Private Sub cmdInitialize_Click()
Vo = Worksheets("sheet1").Range("B5")
Theta = Worksheets("sheet1").Range("B6") * 3.14159 / 180
Vox = Vo * Cos(Theta): Voy = Vo * Sin(Theta)

Xo = 0: Yo = 0
row = 2: t = 0
Worksheets("sheet1").Range("C2:C6") = 0
Worksheets("sheet1").Range("D2:D6") = Xo
Worksheets("sheet1").Range("E2:E6") = Yo
End Sub

Private Sub cmdPause_Click()
bStop = True
End Sub

Function Velocity(Vox, ax, t)
Velocity = Vox + ax * t
End Function

Function Position(Xo, Vox, ax, t)
Position = Xo + Vox * t + 0.5 * ax * t ^ 2
End Function
```

6.5 Hidden Loops

Incremental iteration is based on the requirement that the net force (really the acceleration) remain constant over the incremental interval otherwise the position and velocity functions do not accurately model the motion. It follows that the smaller the **Δt** the more likely the acceleration is to remain constant and so the better the approximation. Although error considerations suggest that **Δt** should always be as small as possible the trade-off here is two-fold : the smaller the **Δt** the slower the simulation, and the greater the number of data points produced (which becomes significant if complete data sets are desired).

Application # 17 simulates the simple harmonic motion of a mass attached to a spring moving vertically. A choice of **Δt** $= 10^{-3}$ seconds leads to significant deviations in the energy graphs; however, the better choice of **Δt** $= 10^{-4}$ s (100 μs) generates 10,000 data points every second. As the **Excel** worksheet allows of the order of 66,000 rows the simulation will crash before completing 7 seconds. [The simulation would actually crash just prior to 3.3 seconds as the **row** index exceeds the maximum of 32,767 allowed for integers. For data generation between 32,767 and 66,000 values the row index must be declared as a **Long** integer.] At the other end of the spectrum are space trajectories, such as the rocket moving from the Earth to the Moon in a later problem. While **Δt** may be10 seconds or longer in such situations, a huge volume of data will nevertheless be produced due to the long running times.

The solution is to create a "hidden" loop inside the main loop. A **hidden loop** is an iterative loop that contains no output statements (and so writes no data onto the worksheet); it performs a specified number of iterations and output is only produced once it's exited.

```
DeltaT = 0.0001

Do
   For I = 1 To 100
      Fnetx = whatever
      ax = Fnetx / mass
      Xf = Xo + Vox * deltaT + 0.5 * ax * deltaT ^ 2
      Vfx = Vox + ax * deltaT
      Xo = Xf
      Vox = Vfx
      t = t + deltaT
   Next I

   Worksheets("sheet1").Cells(row, 3) = t
   Worksheets("sheet1").Cells(row, 4) = Xf
   Worksheets("sheet1").Cells(row, 5) = 0
   row = row + 1
Loop Until t >= Total_time Or bStop = True
```

One hundred of the usual iterations occur in the **For…Next** loop, but output is only produced once the **For…Next** loop is exited, or every 0.01 seconds (100 * 0.0001 s). The **For…Next** loops are said to be "**hidden**" since no output is produced and the user is unaware of their existence.

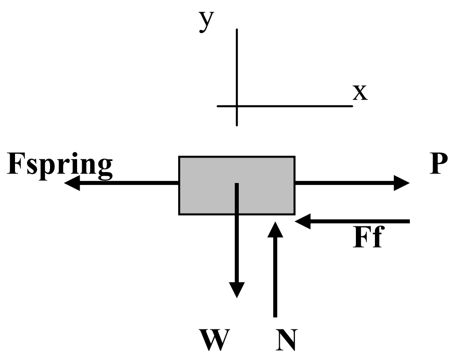
Caution : They are a variety of combinations of **For…Next** and **Do…Loop**s that can be used to program the inner hidden loops and outer main loop. Particular combinations may corrupt the cumulative values of **t** if care is not taken (that is, **t** may not increase in increments of **deltaT**, and certain values might end up being skipped). You are encouraged to follow the structure given above which offers the useful **bStop** exit condition.

Application # 16 : Damped Oscillatory Motion of Mass Attached to Spring (with Friction, and Applied Force)

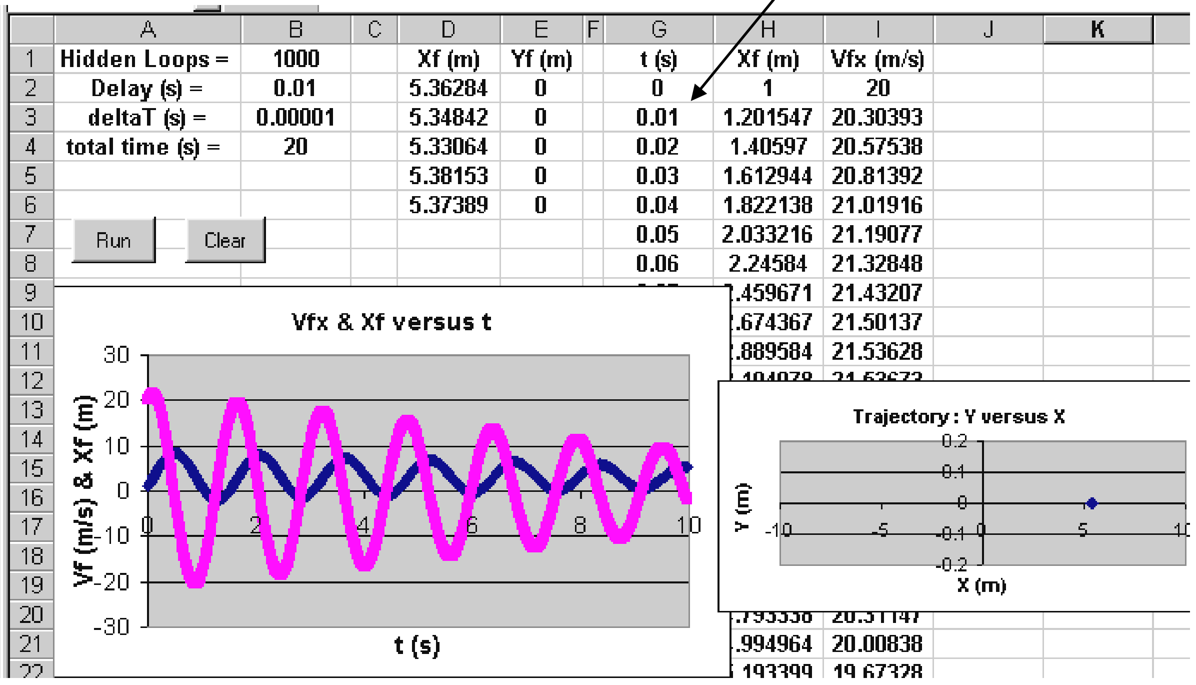
A 2 kg block is attached to a spring (k = 32 N/m) and rests on a surface for which $u_k = 0.2$. An applied force of P = 100 N is pulling horizontally on the block. The block is moving at 20 m/s to the right when the spring is stretched 1 meter . Construct a simulation that displays the velocity and position of the mass as a function of time, also create a comet tail trajectory (not shown below). X = 0 coincides with the un-stretched location of the spring. (Ff = uk N = 0.2*20 = 4 Newtons for g = 10 m/s^2)

Since the spring is attached to the block the spring force term is always present in the net force :

Fnetx = –k * Xo + P – Sgn(Vox) * Ff



With 1000 hidden loops and Δt = 0.00001 the data points are produced every 1000* 0.00001 = 0.01 seconds.



	A	B	C	D	E	F	G	H	I	J	K
1	Hidden Loops =	1000		Xf (m)	Yf (m)		t (s)	Xf (m)	Vfx (m/s)		
2	Delay (s) =	0.01		5.36284	0		0	1	20		
3	deltaT (s) =	0.00001		5.34842	0		0.01	1.201547	20.30393		
4	total time (s) =	20		5.33064	0		0.02	1.40597	20.57538		
5				5.38153	0		0.03	1.612944	20.81392		
6				5.37389	0		0.04	1.822138	21.01916		
7							0.05	2.033216	21.19077		
8							0.06	2.24584	21.32848		
9								.459671	21.43207		
10								.674367	21.50137		
11								.889584	21.53628		
12								[illegible]	[illegible]		
13											
14											
15											
16											
17											
18											
19											
20								[illegible]	[illegible]		
21								.994964	20.00838		
22								[illegible]	[illegible]		

Application # 8 (Rocket car) is used as the shell for this program.

```
Option Explicit
Dim Delay As Double

Private Sub cmdRun_Click()
Dim Xo As Double, Vox As Double, Xf As Double, Vfx As Double, mass As Double
Dim ax As Double, t As Double, deltaT As Double, Fnetx As Double
Dim row As Integer, Total_time As Double, row2 As Integer
Dim b As Double, Ff As Double, k As Double, P As Double
Dim I As Integer, Hidden_Loops As Integer
```

```
Hidden_Loops = Worksheets("sheet1").Range("B1")
Delay = Worksheets("sheet1").Range("B2")
deltaT = Worksheets("sheet1").Range("B3")
Total_time = Worksheets("sheet1").Range("B4")

mass = 2: Xo = 1: Vox = 20: k = 32: P = 100: Ff = 4
t = 0

Worksheets("sheet1").Range("G2") = t
Worksheets("sheet1").Range("H2") = Xo
Worksheets("sheet1").Range("I2") = Vox

row = 2: row2 = 3

Do
   For I = 1 To Hidden_Loops
      Fnetx = -k * Xo + P - Sgn(Vox) * Ff
      ax = Fnetx / mass

      Xf = Xo + Vox * deltaT + 0.5 * ax * deltaT ^ 2
      Vfx = Vox + ax * deltaT

      Xo = Xf
      Vox = Vfx
      t = t + deltaT

   Next I
   Call TimeDelay(Delay)
   Worksheets("sheet1").Cells(row, 4) = Xf
   Worksheets("sheet1").Cells(row, 5) = 0
   Worksheets("sheet1").Cells(row2, 7) = t
   Worksheets("sheet1").Cells(row2, 8) = Xf
   Worksheets("sheet1").Cells(row2, 9) = Vfx
   DoEvents
   row = row + 1
   row2 = row2 + 1
   If row = 7 Then row = 2

Loop Until t >= Total_time

End Sub


Private Sub cmdClear_Click()
 Worksheets("sheet1").Range("G2:I10000").Clear
 Worksheets("sheet1").Range("D2:E6") = 0
End Sub
```

The initial values of the complete data are written out before the loop is entered.

The complete data printout starts in row 3 since row 2 holds the initial values.

Programming to Obtain the Maximum Extension of the Spring

One of the variations of this program might involve identifying the maximum extension of the spring. In a typical theoretical solution involving the Work-Energy method the final velocity is set to zero and a quadratic equation in the spring extension X results. However, searching for a final velocity of zero and then identifying the associated position will not work with an incremental iterative analysis since it is unlikely that any of the iterations will end exactly at the instant that the block momentarily stops moving, regardless of how small Δt is made. That is, there will never be a value of velocity which is exactly zero. Instead, one approach is to have the program search for the smallest value of **Vox** and then store the corresponding value of **Xf** (a minimum value search as described Chapter 3 and extended to vectors later in this chapter, **section 6.7**).

```
If Abs(Vox) < small Then small = Abs(Vox): X_Max = Xf
```

where the value of **small** is initialized outside the **Do…Loop** to a large value (so that the initialized value is not , by misfortune, the smallest value).

Interpreting negative signs in physics. The absolute value was taken above to prevent negative velocities from falsely being recognized as small. ***Remember, a negative position, velocity, acceleration, or force in physics simply means that the vector quantity is directed in the negative direction (to the left in this case).*** A large negative velocity in the current situation simply means that the block is moving to the left with great speed.

Clearly the very first maximum **Xf** will be the greatest extension. Unfortunately, the program will continue to search for a minimum velocity as the block oscillates back and forth in a decaying motion [5]. And, a value of minimum velocity closer to zero may very likely be found at a later time for which the extension (or compression) is not the overall maximum. This can be avoided by picking off the first **Xf** maximum by perhaps using the **Sgn( )** function to detect the first velocity reversal.

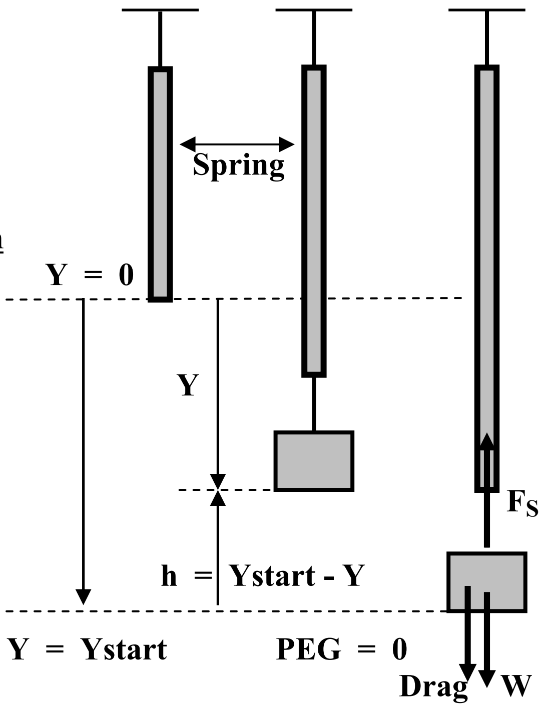
Led Astray by Preconceived Ideas : Interestingly, our approach to finding the maximum extension has been unduly influenced by the prior learning of the standard theoretical analysis which imposes the constraint Vfx = 0 . In fact, the velocity condition can be ignored altogether for the programmed analysis; all that's needed is a search for **X_max** which could be done using : **If Abs(Xf) > X_max Then X_max = Abs(Xf)** (plus an initialization and write of **X_max**).

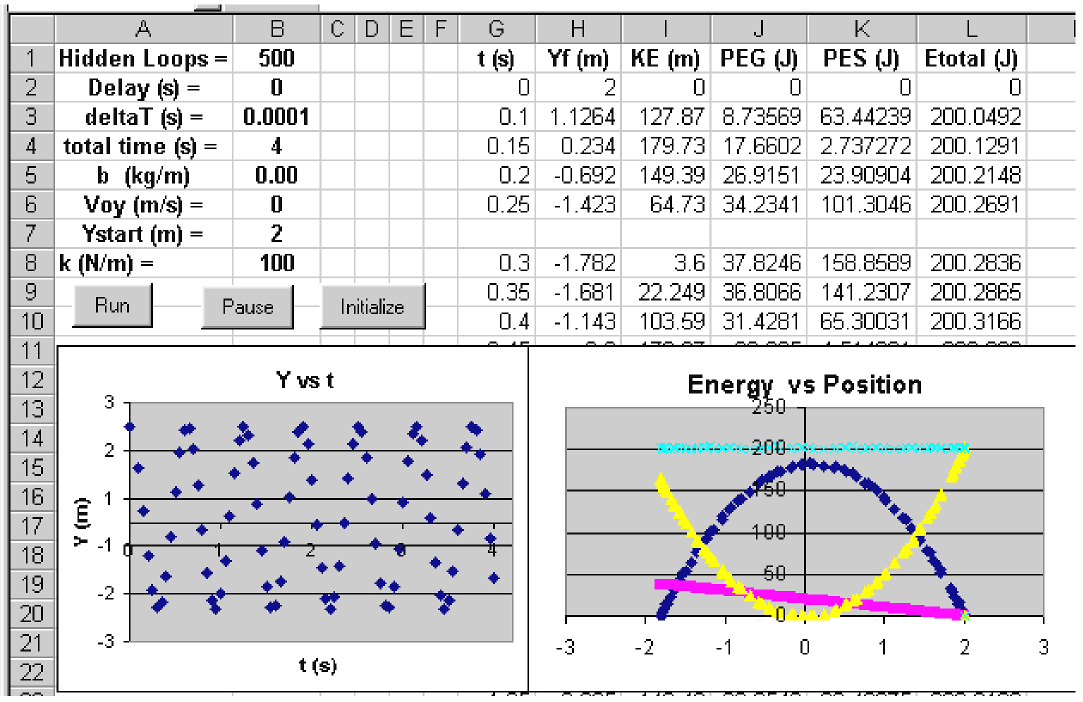
[5] In the absence of force **P** the block will oscillate right and left until friction work consumes all the energy and the spring ends up in some stretched equilibrium position. External force **P** complicates matters by adding energy as the block moves to the right and consuming energy as it moves to the left. As successive rightward displacements will always be greater than successive leftward displacements (including compression) it might be concluded that more energy will be added than removed by **P**, and since **P** is greater in magnitude than **Ff** it might be argued that more energy will be put in than removed (the system is "endothermic" rather than "exothermic", to borrow from chemistry). Does it make sense, then, that the motion is damped out ? It might be useful exercise for you to increase the total time and allow the block to reach its ultimate equilibrium position (if it ever does !).

<u>Application # 17</u> : <u>Simple Harmonic Motion of a Mass Hanging Vertically from a Spring</u>

An object of mass "**M**" is hanging vertically from a spring (spring constant = **k**). The object is initially a distance "**Ystart**" from the unstretched location of the spring (**Y = 0**) and has a velocity **Voy** (assumed upward for the drag force shown in the diagram). This application will plot graphs of the kinetic, potential, and total energy versus position, and the position as a function of time. A hidden loop is used to reduce the amount of data. the spring force may be positive or negative

The net force acting on the mass at any time is **Fnet = Weight – Spring Force ± Drag** where the spring force may be positive or negative. [*Note : the positive direction for "Y" has been chosen to be downward.*] The spring force contribution is <u>opposite</u> to the extension or compression of the spring. For the initial downward <u>extension</u> of the spring, the spring force is directed upward; when the spring is <u>compressed</u>, the spring force is directed <u>downward</u>. The drag term due to air resistance has a sign that varies depending on the direction of motion. When the mass moves down the drag is directed upward and is negative; when the mass moves up the drag acts downward and is positive.





	A	B	C	D	E	F	G	H	I	J	K	L
1	Hidden Loops =	500					t (s)	Yf (m)	KE (m)	PEG (J)	PES (J)	Etotal (J)
2	Delay (s) =	0					0	2	0	0	0	0
3	deltaT (s) =	0.0001					0.1	1.1264	127.87	8.73569	63.44239	200.0492
4	total time (s) =	4					0.15	0.234	179.73	17.6602	2.737272	200.1291
5	b (kg/m)	0.00					0.2	-0.692	149.39	26.9151	23.90904	200.2148
6	Voy (m/s) =	0					0.25	-1.423	64.73	34.2341	101.3046	200.2691
7	Ystart (m) =	2										
8	k (N/m) =	100					0.3	-1.782	3.6	37.8246	158.8589	200.2836
9							0.35	-1.681	22.249	36.8066	141.2307	200.2865
10							0.4	-1.143	103.59	31.4281	65.30031	200.3166

Once again the formal definition of the spring force includes a negative sign **– k*Yo** that automatically adjusts the direction of the force for both extension or compression. [Provided that the positive direction of Y is selected in the direction of the extension !] As with most mechanics problems, the mass is considered to be a point mass; that is, it has no dimensions so that **Y** represents both the location of the mass and the extension/compression of the spring, and is used to determine the gravitational potential energy as **PEG = mass * 10 * (Ystart - Yf)** .

Fnet = M * g – k * Yo – Sgn(Vi) * b * Voy ^ 2

Three different energies are present : kinetic energy : **KE** = **(1/2)mV^2** , gravitational potential energy : **PE_G** = mgh = **mg(Ystart - Y)**, and spring potential energy : **PE_S = (1/2)kY^2** . The total energy is the sum of these three energies.

This program is based on Application # 9 (vertical projectile) with a pause button & hidden loops included.

```
Option Explicit
Dim Delay As Double, bStop As Boolean
Dim Yo As Double, Voy As Double, Ystart As Double
Dim t As Double, row As Integer, mass As Double
Dim KE As Double, PEG As Double, PES As Double, Etotal As Double
Dim k As Double

Private Sub cmdInitialize_Click()
Worksheets("sheet1").Range("G2:L600").Clear

Voy = Worksheets("sheet1").Range("B6")
Ystart = Worksheets("sheet1").Range("B7")
k = Worksheets("sheet1").Range("B8")

row = 2: t = 0: mass = 1

Yo = Ystart
PEG = 0    ' the starting point is the zero location for PEG
PES = 0.5 * k * Voy ^ 2
KE = 0.5 * mass * Voy ^ 2
Etotal = KE + PEG + PES
Cells(row, 7) = t
Cells(row, 8) = Yo
Cells(row, 9) = KE
Cells(row, 10) = PEG
Cells(row, 11) = PES
Cells(row, 12) = Etotal
row = row + 1
End Sub

Private Sub cmdPause_Click()
bStop = True
End Sub
```

Ystart is the location of the PEG = 0 reference and will be used in the calculation of the PEG.

```
Private Sub cmdRun_Click()
Dim Yf As Double, Vfy As Double
Dim ay As Double, deltaT As Double, Fnety As Double
Dim Total_time As Double
Dim b As Double
Dim I As Integer, Hidden_Loops As Integer, JJ As Integer

Hidden_Loops = Worksheets("sheet1").Range("B1")
Delay = Worksheets("sheet1").Range("B2")
deltaT = Worksheets("sheet1").Range("B3")
Total_time = Worksheets("sheet1").Range("B4")
b = Worksheets("sheet1").Range("B5")

bStop = False

Do
      For I = 1 To Hidden_Loops
         Fnety = mass * 10 – k * Yo – Sgn(Voy) * b * Voy ^ 2
         ay = Fnety / mass

         Yf = Yo + Voy * deltaT + 0.5 * ay * deltaT ^ 2
         Vfy = Voy + ay * deltaT

         Yo = Yf
         Voy = Vfy
         t = t + deltaT
      Next I

   PEG = mass * 10 * (Ystart - Yf)
   PES = 0.5 * k * Yf ^ 2
   KE = 0.5 * mass * Vfy ^ 2
   Etotal = KE + PEG + PES
   Cells(row, 7) = t
   Cells(row, 8) = Yo
   Cells(row, 9) = KE
   Cells(row, 10) = PEG
   Cells(row, 11) = PES
   Cells(row, 12) = Etotal
   Call TimeDelay(Delay)
   Worksheets("sheet1").Cells(row, 7) = t
   Worksheets("sheet1").Cells(row, 8) = Yf

      For JJ = 1 To 60
      DoEvents: DoEvents
      Next JJ

   row = row + 1

Loop Until t >= Total_time Or bStop = True

End Sub
```

Set appropriate axis **Max** and **Min** values for the graph axes to avoid annoying auto-scaling shifts.

Try using **deltaT = 0.001** and **Number_Loops = 50**; The **KE** and total energy curves end up being quite scattered.

6.6 Implicitly Defined Boundary Conditions

Because of the nature of the equal sign in computer code, the boundary conditions can be implicitly applied. [As was mentioned in Chapter 1, the equal sign is an *assignment* expression which takes the value on the right side and *assigns it* (enters it) to the storage location given on the left side.] By dropping the " f " and the "o" from the variables, and working only with **X**, **Y**, **Vx**, and **Vy** , it is possible to replace the code statements on the left below with the following :

$$\left.\begin{array}{l}\textbf{Xf = Position(Xo, Vox, ax, deltaT)}\\ \textbf{Yf = Position(Yo, Voy, ay, deltaT)}\\ \\ \textbf{Vfx = Velocity(Vox, ax, deltaT)}\\ \textbf{Vfy = Velocity(Voy, ay, deltaT)}\end{array}\right\} \xrightarrow{\textbf{Replaced by}} \left\{\begin{array}{l}\textbf{X = Position(X, Vx, ax, deltaT)}\\ \textbf{Y = Position(Y, Vy, ay, deltaT)}\\ \\ \textbf{Vx = Velocity(Vx, ax, deltaT)}\\ \textbf{Vy = Velocity(Vy, ay, deltaT)}\end{array}\right.$$

The boundary condition statements can be completely removed :

$$\left.\begin{array}{l}\textbf{Xo = Xf}\\ \textbf{Yo = Yf}\\ \textbf{Vox = Vfx}\\ \textbf{Voy = Vfy}\end{array}\right\} \boxed{\text{Delete !!}}$$

The variable mass rocket can similarly be converted to implicit boundary conditions :

$$\textbf{Mf = Mo – k*deltaT} \longrightarrow \textbf{M = M – k*deltaT}$$

$$\textbf{Mo} = \textbf{Mf} \longleftarrow \boxed{\text{Delete !!}}$$

Future notes may delve further into implicit boundary conditions.

6.7 Maximum, Minimum, and Closest Value Searches Revisited

The maximum, minimum, and closest value searches described in Chapter 3 applied to scalar values only. The modifications presented here for the maximum and minimum searches amount to including the absolute value function **Abs( )**; the closest value search which remains unchanged.

Negative signs have different meanings for vectors : Negative signs attached to **vector** quantities only indicate direction, and not relative size. A "large" negative velocity, net force, acceleration, displacement, or momentum actually denotes that the quantity is directed in the negative direction and, in fact, has a large value in that direction. A ball thrown vertically off a cliff at + 15 **j** m/s which hits the bottom 5 seconds later at about – 35 **j** m/s clearly has a maximum speed of 35 m/s directed downward. [Whereas a scalar temperature of -35^{O} C is certainly much smaller than a temperature of $+15^{O}$ C.] It follows that the smallest velocity of an object is zero when it's at rest. Although negative **scalar** numbers are always smaller than positive scalar values, the same is not true for vectors for which the absolute values (magnitudes) must be compared..

Search for maximum. Of typical interest are the maximum height, maximum displacement, maximum stretch of a spring, maximum force, and maximum velocity.

The maximum displacement of an object, or the maximum deflection of a spring (which may occur when it's in compression), can be found using

If Abs(Xf) > X_Max Then X_Max = Abs(Xf): t_max = t

with the value **X_Max** of initialized to **X_Max = –1E+15** outside the loop. The absolute value of displacement vector **Xf** is taken since a negative signs indicates direction and not size. [**X_Max** could have been initialized to **0** since the smallest value of a vector is zero; however, for consistency with scalar searches I sometimes use the scalar initial values.]

Note : a maximum height search is a special case that is essentially a scalar search : a negative location of – 100 **j** meters, although representing a larger directed position on the trajectory, would never be interpreted as being the maximum height for a situation where the largest positive location was + 50 **j** meters. Maximum heights are found without the absolute value as :

If yf > Y_max Then Y_max = yf: t_max = t

where the value of **Y_max** has been initialized outside the loop to a very *small* value : **Y_max = –1E+15**. Notice that the time at **Y_max** has also been captured.

A maximum velocity can be found using :

If Abs(Vfx) > V_max Then V_max = Abs(Vfx): t_at_max_V = t: X_at_max_V = Xf

where the value of **V_max** is initialized outside the loop to **V_max = 0**. Once again a continual updating occurs so that if the current value of **Abs(Vfx)** is greater than the previous **V_max** , the previous value is replaced. Observe that the time and location of the maximum velocity point are also stored and updated.

Reformulating Searches : A search for maximum height, or maximum deflection of a spring, or maximum distance up an incline can be reformulated in terms of searching for when the velocity is zero. The physics of the situation has imposed the additional physical constraint that the displacement is maximized when the object momentarily comes to rest so that a search on the alternative variable velocity is an option. Conversely, a search for a zero velocity in some cases might be appropriately re-framed as a search for a maximum position.

Search for the minimum value : As with the scalar minimum search, the minimum search of vector quantities is the inverse of the maximum search and is created by reversing the inequality and starting at a large initial value so that the starting value is not inadvertently the smallest. The single difference is that smallest possible minimum value is always zero. A minimum velocity can be found using :

If Abs(Vfx) < V_min Then V_min = Abs(Vfx): t_at_min_V = t: X_at_min_V = Xf

where the value of **V_min** is initialized outside the loop to **V_min = 1E+20**.

Search for the closest value : A closest search is actually a minimum search on the difference between the tested value and the target value in which the smallest theoretical value of comparison parameter **Small** is **0**. The formulation is identical to the scalar approach which already addressed potential misinterpretation of negative values by including an **Abs( )** function. A typical problem might involve finding the velocity of an object when its located at **yf = + 20** meters and would be programmed as :

Y_target = 20: Small = 1E+20: Closest = 22222 (this code is located before the search loop is entered)

[The 22222 value has no particular significance; all that's required is a value that is large relative to the numbers involved.]

If Abs(yf - Y_target) < Small Then Closest = yf: V_at_closest = Vfy: Small = Abs(yf - Y_target)

Notice that the closest **yf** value is not the principal objective of this search; it's the value of **Vfy** for which **yf** is closest to 20. Once again the initial value **Closest = 22222** is selected for "noticeability" so that the user recognizes something is wrong if this value appears at the output.

You may be wondering why the search over a continuous set of variables could not be simpler. In this example the object likely started at **yf = 0** meters and moved continuously upward and past the location **yf = + 20** m. Why not use the statement : **If yf = 20 Then V_at_closest = Vfy** ? The difficulty is that the simulation cannot produce every point on the trajectory (it's "sampled", for those of you in the Electronics class), so that no matter how small the value of **deltaT** it is not likely that the value of **yf** will be exactly 20.000000000. Remember, a lot of decimal places are being carried and the **If** statement is unforgiving in requiring that **yf = 20** exactly.

Problems [Chapter 6]

Construct simulations for the situations described below by plotting comet tail trajectories. Features such as complete data, hidden loops, and interactive controls need only be added where requested (although you're free to add them if desired). Consider drag forces if **b** or **Cd** are mentioned.

1. An object is thrown vertically into the air with a velocity of 25 m/s from a point 2 meters above the ground. Develop a program that finds the time and velocity at a position read off the worksheet. Note that there will be two solutions for each position. Include a theoretical solution for one set of values Display a trajectory graph and graphs of y vs t and Vy vs t. Use incremental iteration, although not really necessary.

2. Two trains initially separated by 2,000 meters are on the same track approaching one another. Construct a simulation that plots the trajectory of each train on the same graph and determines the time and locations where the trains are separated by a distance read off the worksheet. Also write out the closest approach for each set of data. Note : there will generally be 2 solutions for each separation, but there may be no solution for certain separations. Try the following data but feel free to invent your own: Train A : Vox = + 40 m/s, ax = – 1 m/s^2, Xo = 0 m; Train B : Vox = – 30 m/s, ax = + 0.5 m/s^2, Xo = 2,000 meters. Include a theoretical solution for one set of data. Use incremental iteration, although not really necessary.

3. A 2 kg block has a velocity of 2 m/s down a 37^o incline (μk = 0.2). The top end of the block is attached to a spring (k = 20 N/m) which is also connected to a wall at the top of the incline. Ignore drag. Plot the actual trajectory as it moves up and down the incline (use rotated coordinates). Add a second series of two points fitted with a linear function to describe the incline. Use **deltaT = 0.01** and plot the complete data for Xf and Vfx on the same graph.

4. A 1 kg block moves on a horizontal surface (μk = 0.2) due to the action of two springs (k1 = k2 = 20 N/m) that are attached, one on each end. The two springs are each stretched 20 cm in order to be attached to the block between them. (Consider the block to have a zero width). The block is pulled 20 cm to the left such that the left spring is at its unstretched position and released from rest . (The right spring is therefore stretched 40 cm when the block is released.) Plot the trajectory of the block.

5. The combined mass of a cyclist and bicycle is 65 kg. The cyclist is capable of generating a frictional force of 150 Newtons at the rear tire in order to propel the bike forward. (Note that the frictional force is causing the motion. Ignore rolling friction on the front tire. Plot the trajectory of the bicycle assuming that air resistance can be modelled as **b*V^2** ; try a value of **b** = 0.5 kg/m. Note : the values of the frictional force and **b** may not be realistic. Without writing out complete data, add code that determines the speed and distance travelled at a some specific time. (Since that time may not actually occur due to the choice of **deltaT**, the code must find the values at the time closest to the specified time.) Assume that the cyclist can decrease their **b** 10 % by leaning forward and down, compare their speed and distance travelled after 15 seconds with and without this reduction.

6. Despite appearing very awkward recumbent bicycles, on which the rider lies back and pedals with feet forward, apparently hold the bicycle maximum speed record. Although the rider cannot create as much downward force on the pedals of a recumbent bike (since in the extreme they can't stand and use their weight), the area of the rider and bike perpendicular to the motion is significantly smaller so that the drag is reduced. Construct a simulation (or modify problem # 5) that models drag as **0.5 ρ Area Cd V^2** and determine the maximum speed of a normal bike (propulsive friction = 150 N, Area = 0.44m^2, total mass = 70 kg) and a recumbent bike (propulsive friction = 140 N, Area = 0.32m^2, total mass = 75 kg). Assume Cd = 1.8 in each case, although clearly the Cd should be different and should

vary along with the area as the cyclist pedals. (Note: these values may not be realistic.) Once the program is working add code that allows the cyclist to stop pedaling at a certain time and which determines the distance over which the bicycle coasts to a rest.

7. A 100 gram ball (diameter = 25 cm, Cd = 0.15) is dropped from a bridge. Construct a simulation of the motion. Also plot complete data of Yf and Vfy versus time for 5 seconds. Interpret the graphs. Repeat the simulation using a mass of 50 grams and comment on the differences. Try **deltaT = 0.01** seconds.

8. A helium filled birthday balloon (mass of rubber = 1 gram, diameter = 25 cm, Cd = 0.15) is released and moves vertically upward under the influence of the buoyant force. The buoyant force is equal to the weight of the volume of air displaced by the balloon, or **ρair*Volume of balloon*g**. Construct a simulation of the motion. Also plot complete data of Yf and Vfy versus time for 5 seconds. Try **deltaT = 0.01** seconds. Include the weight of the helium (**ρHe** = 0.179 kg/m^3, **ρair** = 1.20 kg/m^3).

9. The helium filled balloon of problem # 8 has an additional 25 gram mass taped to it and is released from a window that is 15 meters high. The 25 gram mass falls off after the balloon travels 12 meters (when it is 3 meters above the ground). Construct a simulation of the motion. Also plot complete data of Yf and Vfy versus time for about 15 seconds. Try **deltaT = 0.01** seconds. Interpret the graphs.

10. A 100 gram ball (diameter = 25 cm, Cd = 0.15) is held 2 meters below the surface of a pool. While underwater the ball experiences a buoyant force equal to the volume of water displaced, or **ρwater* Volume of ball*g** (buoyant forces are ignored in the air unless the ball is filled with helium). The ball is released and moves upward. Construct a simulation of the motion and add code to determine the maximum height of the ball. Assume that the ball is a point mass (or a pancake) for the purposes of the transition from water to air. That is, the buoyant force is constant while the ball is underwater and becomes zero as soon as the center of the ball is above the surface of the water. Ignore surface tension on the ball. Plot complete graphs of Yf and Vfy versus time. Option : use calculus to describe the volume of the submerged at each point during the transition between water and air (two different expressions will likely be needed : one when the top half of the ball is breaking the surface with the bottom half completely submerged, and the second when the bottom half is breaking through the surface with the top half entirely in the air).

11. It seems impossible to throw a beach ball into the water such that it is completely submerged. Construct a simulation of the motion of a 100 gram ball (diameter = 25 cm, Cd = 0.15) that has a downward velocity **Voy** at the surface of the water. Ignore the splash produced by the impact; model the buoyant force as in problem # 10. Assume the ball behaves as a point mass, or a flat pancake, so that the transition from air to water is instantaneous, and such that the full buoyant force is present as soon as the center of the ball is in the water. Find the value of initial velocity that causes the center of ball to reach a point 13 cm below the surface of the water. Program the search for **Voy** to be automatic by using an outer **For…Next** loop that varies **Voy** over a certain range.

12. A 200 gram ball is dropped vertically from a building. The ball is observed to fall 16 meters in 4 seconds. Construct a simulation of the motion and then find the value of **b** consistent with the data as follows : Add an outer **For…Next** loop to search over a range of values of **b**. For each different **b** find the location of the ball at 4 seconds and determine the error with the desired displacement of 16 meters. Add code that identifies the **b** value with the minimum error. Once the program is working save it under a new name and modify it so that each value of **b** and associated error are written onto the worksheet. Plot **Error versus b**.

13. A rock and a ping-pong ball have the same cross-sectional area (about 2 cm^2) but the ping-pong ball has one-hundredth the mass (choose a reasonable value). Both are thrown vertically and are subject to the force from a person's that varies with distance as :

F_hand = 66.66y + mg + 0.01 for 0 <= y <= 0.75 meters
F_hand = –66.66y + 100 + mg + 0.01 for 0.75 < y <= 1.5 meters.

Refer to the box at the bottom of pg 6-19

where **mg** is the weight of the object. Using hidden loops and a **Pause** button, construct a simulation of the motion (try **deltaT** = 10 or 100 μs) Plot complete graphs of Yf , Vfy, and Fnet versus time for the period that the objects are in the air (use separate graphs). Interpret the graphs in particular explaining why the lighter ping-pong ball does not travel as high although subject to the same force of the hand. Which object has the greater velocity as it leaves the hand; explain why, or why not, this object should reach a greater maximum height.

14. A 2 kg box (area = 0.15 m^2) has a hole through its center and rides up and down on a vertical, frictionless pole. At the bottom of the pole is a spring (k = 40 N/m). The box has a downward velocity of 2 m/s when it is 50 cm above the unstretched spring. Construct a simulation of the motion which includes hidden loops and a pause button. Plot a complete graph of Yf and Vfy versus time. Choose a value of Cd that produces a "pleasing" damping. Add code to calculate kinetic energy, both potential energies, and total energy and plot all values on the same graph versus Xf (the graph will be "cluttered" when energy is not conserved); include a copy of the energy graph for Cd = 0 along with the Yf & Vfy versus time for a non zero value (be certain that the value of Cd is obvious along with values for deltaT and hidden loops).

15. A 5 kg box (cross-sectional area that is 10 cm x 10 cm, Cd = 0.5) is released from rest on a 60^o inclined plane (μk = 0.2). The box travels 2 meters down the incline amd enters water (the incline extends underwater). Construct a simulation for the block showing actual inclined motion (add a separate series showing two points on the incline with a straight line joining them).

16. Future: Spring loaded ball launch in ABS 4" drain pipe as per IB experiment. Use large compressive spring scale to determine k .

17. A toy car (mass = 100 grams) is accelerated across a floor with the force of a person's hand which can be described by :

F_hand = 26.66x + 3.5 + 0.01 for 0 <= x <= 0.75 meters
F_hand = –26.66x + 40 + 3.5 + 0.01 for 0.75 < x <= 1.5 meters.

Refer to the box at the bottom of pg 6-19

The car has rusty axles which combine to create an effective resistive force (including friction) of 3.5 Newtons (which was added to the force of the hand to prevent a backward net force occurring as motion just begins). Construct a trajectory simulation.

18. Modify **Application # 8 pg 6-5** so that the car can reverse direction and the thrust can be adjusted using a scroll bar slider. The rocket should provide thrust over the range – 8000 to + 8000 newtons. Include a **Pause** button and hidden loops. Add graphs that display the complete Vx versus t , X vs t , and Fnet vs t. Write the elapsed time into a single cell on the worksheet, and the current velocity into another cell. Add a **Reset** button that resets the **Thrust** and scroll bar to zero. It should be possible to "drive" the rocket car back and forth across the screen. Practice bringing the car to rest at some particular location. Determine the maximum speed under a constant engine thrust of 6000 Newtons; also find the time to travel 2000 meters and the velocity at that point. (1)

19. Modify problem # 18 so that the engine burns fuel at a rate of k = 10 kg/s for a thrust of 8000 Newtons. The expression for k as a function of thrust is : k = 10 (kg/s)*Thrust (N)/8000 (N) . The engine burns out after 1000 kg of fuel have been consumed. Again determine the maximum speed under a constant engine thrust of 6000 Newtons and find the time to travel 2000 meters and the velocity at that point. Compare with the previous results.

20. **Model Rocket. (Constant mass assumption).** A model rocket is launched vertically into the air starting from rest. The mass of the rocket is 0.1 kg and is assumed to be constant. The rocket engine produces a constant thrust of 6 newtons for 8 seconds after which the thrust is zero (the engine has burned out). Model the air resistance acting on the rocket as $b*v^2$ where b = 0.001 kg/m. Include hidden loops and a Pause button. Use a **deltaT** no greater than 0.0001 seconds. Plot a comet tail graph of the trajectory Y vs X, and complete graphs of Vy vs t, and Fnet vs t. Include a **Pause** button and hidden loops.

    Add **If** statements to determine Ymax and the associated time, Vymax and the associated time and Y location, Fnetmax and the associated time and Y location. Plot the current Ymax as a second data series with a single point. The point will follow the trajectory and then stop at maximum height. (2)

21. **Model Rocket. (Variable mass assumption).** Modify the analysis of problem # 20 to account for a decreasing mass for the 8 seconds that the engine burns fuel. The rate of fuel consumption is 6 g/s (or 0.006 kg/s). [Note: there is **no** need to use Thrust = Ve dm/dt .] Compare the maximum height with problem # 20. <u>Explain completely why the rocket did not travel as high for the same thrust but decreasing mass</u> (which seems somewhat counter intuitive). Extend your explanation by considering a golf ball and a ball of paper which have the same cross-sectional area and which are thrown horizontally with an identical force; explain why the paper ball will not travel nearly as far. (5) Include a **Pause** button and hidden loops.

22. Modify problem # 21. Use the value of burn rate **k,** and the average thrust and burn time (given in the notes) for the C6-5 engine (or available on the www.thrustcurve.com site) to find the maximum height of an **Estes Super Shot** rocket. Assume Cd = 0.45 . The mass of the rocket without the engine is 125 grams. Include a **Pause** button and hidden loops.

23. Modify problem # 20 (and eliminate the Fnet vs t data and graph). A 149 gram total mass rocket has an engine that produces an average thrust of 4.74 N for 1.86 s. Assume a constant mass. The height of the rocket after 3 seconds is observed to be 74 meters. Use the approach of problem # 10 to find Cd. That is, add an outer loop to vary Cd over a reasonable range and for each tested Cd find the height at t = 3 seconds. Determine the error with the experimental height and check if the latest value is the minimum value. Plot a graph of **Error versus Cd**. Include a **Pause** button and hidden loops.

24. A variation of problem # 23 is to add an additional piece of information by measuring the time and location of maximum height. The search on each simulation for different values of Cd would involve identifying the maximum height and computing the sum of the errors with the experimental values of tmax and Ymax. Since the value of tmax is much smaller than Ymax some sort of weighting would be useful. Plot a graph of Error versus Cd. [Note : this approach did not appear to produce consistent results in the April 2004 “trials”, possibly because of imprecision in determining when maximum height actually occurred. Relatively large variations in time, 0.1 seconds, are associated with only very slight variations in Ymax.] <u>Data</u> : Ymax = 106.4 m at tmax = 5.50 s. Include a **Pause** button and hidden loops.

25. Two methods can be used to experimentally obtain the position of a rocket at some time. The first uses a video camera and the built-in VCR time counter. The camera is maintained in a fixed orientation and a known reference distance, such as between two trees, is kept in the frame. The known distance provides a scale factor that can be used to determine any distance on the video. The second method involves attaching a very light fishing line to the rocket and using a stop watch. Modify problem # 20 (including the mass and engine data of problem # 23) to account for a heavy string that has a mass of 5 grams for every 20 meters. Find the difference in Ymax and tmax with and without the string. Include a **Pause** button and hidden loops.

28. A charge of 10 μC is placed on an object that is fixed in space. A 1 gram mass carrying a charge of 1 μC is moving directly toward the first charge and has a speed of 10 m/s when it is 20 meters away. Neglecting gravitational forces, how close does the moving charge get to the large charge before momentarily coming to rest ? <u>Use code statements to determine the distance; do not obtain the value from graphs</u>. Plot the comet tail trajectory of the moving charge (include the stationary charge in the plot). Also plot complete graphs of force vs distance and velocity vs distance. (8) Coulomb's Law gives the force on the moving charge at any distance "x" from the fixed charge : $F = k q_1 q_2 / x^2$ where $k = 9E09$ N m^2/C^2. Since the charges have the same polarity the force is repulsive. Remember that in a theoretical solution using conservation of energy the expression for Coulombic potential energy, $q_1 V = q_1 k q_2 / x$, must have positive values for the separation, x, between the charges. The sign of the potential energy results from the polarity of the charges.

30. Modify the Ball-in-a-Box application in this chapter by adding a hole somewhere in the top and somewhere in the right wall. The size of the hole should be a variable that can be read off the worksheet. Extend the bottom wall so that it extends from – 5 meters to + 10 meters. Once the ball is out of the box a different set of **If** statements must be used to have it bounce off the outside of the top and two side walls.

31. [Choose your own values]. A rocket with constant mass, **M**, travels directly away from the Earth (say in the X direction) in a 1-D motion. The engines burn until the rocket achieves **escape speed** at which time they are shut down. The gravitational force between the mass of the Earth, **Me**, and the rocket is **G Me M/x^2** where **G** is the gravitational constant, and **x** is the distance from the Earth. The escape speed at a distance **x** from the Earth is **Sqr(2 G Me/x)** . Do not consider the presence of the Moon. Construct a simulation that includes evaluation of kinetic energy, potential energy (**–GMe/x** yes, there's a negative sign that comes from integrating $1/x^2$), total mechanical energy (kinetic plus potential), and the work done by the non-conservative rocket engine. Plot the energies and the non-conservative work on a single graph and interpret the results. Add code to verify that the work done by non-conservative forces causes and appears as a change in the total mechanical energy of a system.

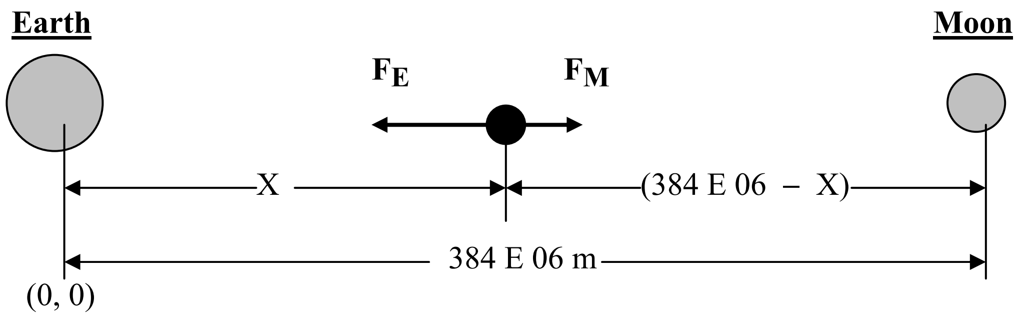
32. CSI bullet lab assignment adding tests and other guidance. Debug issue.
Spear gun versus bullet shot water problem

35. **1-D LUNAR LANDER.**

This application will “fly” a rocket with constant mass from the Earth to the Moon along the X axis under the control of a 1-D thruster that causes acceleration or deceleration. The amount of thrust will be controlled by means of a scroll bar slider. Both the Earth and the Moon are assumed to be stationary and not rotating. The problem is intended as a precursor to the more general 2-D space flight problem presented in Chapter 7.

Theory

The total gravitational force on the spacecraft due to the Earth and the Moon is normally a 2 (or 3) dimensional vector sum. Since this trajectory is assumed to be on a straight line between the Earth and the Moon, the net gravitational force is simply the difference between the two forces. The system origin is located at the center of the Earth, and both the Earth and the Moon are modelled as point masses[1] .



The net gravitational force on the spacecraft of mass “**m**” is **Grav_Force = $-F_E + F_M$**

where $$F_E = \frac{G\,M_E\,m}{X^2} \qquad \text{and} \qquad F_M = \frac{G\,M_M\,m}{(384\text{ E }06 - X)^2}$$

Important : The expression **Grav_Force = $-F_E + F_M$** is only valid between the Earth and the Moon. In the region to the left of the Earth both force contributions are positive, while in the region to the right of the Moon both components are negative. Various programming schemes can be used to set the correct component signs. The least elegant uses **If** , or **Select Case,** statements, to detect which of the 3 regions the spacecraft is currently in; a different combination F_E and F_M would be constructed for each region. The expression : **Sgn(Xe – Xo)*F_E + Sgn(Xm – Xo)*F_M** is valid in all three regions and is based on general expressions for the force components that are presented in Chapter 7, section 7.1 .

The total force on the spacecraft includes the engine thrust which will be positive when accelerating toward the Moon and negative (acting to the left) when decelerating to land on the Moon. [The return journey would begin with a negative thrust to accelerate off the Moon toward the Earth, and end with a positive thrust when decelerating to land on the Earth.]

Fnet = Thrust + Grav_Force

[1] the point mass assumption leads to singularities at the location of the Earth and the Moon. As the spacecraft approaches these singularities it will inevitably be subject to huge gravitational forces that may “catapult” it far into space. Singularities can be avoided by using a linear model of the field inside the mass : GMr/R^3 where R is the radius of mass M. Refer to Gauss’ Law applied to spherical charge distributions.

<u>Programming</u> **<u>(Version 1)</u>**

Modify an off-the-shelf 1-D incremental iterative program with a **Pause** button and hidden loops to account for the net force and initial conditions present in this situation : **Xo = 6.37 E 06** m (the surface of the Earth) ; **Vox = 0** m/s; **deltaT = 10** secs ; **t_max = 300,000** sec.

Specify the amount of **Thrust** using a **scroll bar** (**Max** = + 100, **Min** = – 100); the scroll bar value should be divided by 10 to give a **Thrust** range between – 10 and + 10 newtons. Use a spacecraft mass of 1 kg. Although the difference between the weight of the spacecraft on the surface of the Earth (about 9.833 N for this model) and the maximum **Thrust** would not seem to be large enough to get the craft to the Moon, very large speeds can, in fact, be acquired. [Give two reasons why this maximum thrust will be adequate.] Larger scroll bar thrust values may be used; however, flying the craft then becomes a bit trickier and excessive speed may result. Add a **Reset Thrust to Zero** button which sets the scroll bar value (and hence the thrust) to zero.

Test the net force contributions for zero thrust by writing out the values at the surface of the Earth and the surface of the Moon (9.83 Newtons and 1.62 Newtons respectively). [radius of Moon = 1.74E06 meters, which needs to be converted to a distance from the Earth.]

Although the basic calculations will involve standard MKS units, it is useful to display the position in kilometers and the velocity in km/hr. Also useful for landing are two cells which display the distance to the <u>surface</u> of the Moon and the Earth. A cell containing the value of total gravitational force will indicate when the spacecraft is under the Earth's or Moon's gravitational pull (the Moon's field should be predominate at about 9/10-th of distance to the Moon).

"Flying" the Spacecraft " To actually lift-off the surface of the Earth the scroll bar must be set at a maximum thrust of 10 Newtons. Be patient, the speed will start rising. Also be careful if you decide to change the maximum and minimum thrust values since the low mass means that the speeds can very quickly get out of control. Remember that the point mass models means that there's a singularity lurking inside both the Earth and the Moon if your spacecraft happens to overshoot the landing, or sink down on lift-off (due to insufficient thrust).

<u>Programming</u> **<u>(Version 2)</u>**

A number of improvements can be made to facilitate the flying of the spacecraft, a few possibilities are suggested here; feel free to add your own modifications.

a) Excessive speed build-up can be avoided by including a **Message Box** that detects and warns if the speed exceeds 54,000 km/hr (15,000 m/s). To avoid having the message box immediately re-appear once the **OK** button is pressed, the program should allow the pilot a period of time (say 2000 seconds) to reduce speed (that is, the message box cannot appear during the 2000 second interval).

b) To aid with launch and landing two additional graphs should be added: one with graph axis minimum set to the radius of the Earth and the maximum set to about 50,000 km (50E06 meters), the other with a minimum set to radius of the Moon and the maximum set to about 10E06. The first graph can use the trajectory data directly, the second graph needs to convert the position relative to the center of the Earth to position from the center of the Moon. As most users tend to favour vertical launch and landing the data sources should be switched to present motion along the Y axis.