CHAPTER # 7 2-D DYNAMICS: INCREMENTAL ITERATION APPLIED TO MORE COMPLEX TRAJECTORIES

For any motion, if the net force can be modeled, then the motion can be simulated using incremental iteration.

This chapter extends incremental iterative analysis to "genuine" 2-D problems; that is, where the net force has both X and Y components which change in magnitude as the object moves. By considering motion over small Δt the accelerations almost vary linearly which means that the errors will nearly cancel out if they're considered constant and the standard kinematics equations for 2-D are applied. [Refer to the discussion on page 8-1.]

$$X_f = X_o + V_{ox}\,t + \frac{1}{2}a_x\,t^2 \quad \text{... (1)}$$

$$V_{fx} = V_{ox} + a_x\,t \quad \text{....(2)}$$

where $a_x = F_{netx}/m$ .... (3)

$$Y_f = Y_o + V_{oy}\,t + \frac{1}{2}a_y\,t^2 \quad \text{....(4)}$$

$$V_{fy} = V_{oy} + a_y\,t \quad \text{....(5)}$$

where $a_y = F_{nety}/m$ .... (6)

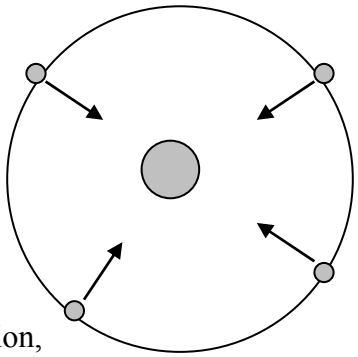
The chapter begins by presenting general expressions for the components of central forces and other forces that act either toward or away from a fixed point. These expressions will be used in most of the 2-D problems encountered such as those involving masses and charges, and circular motion problems particularly with masses moving in vertical circles. The chapter concludes with an application that creates a virtual spacecraft which can be steered by adjusting tangential and perpendicular thrusters. Gravitational forces of the Earth and the Moon will be added to the virtual spacecraft in one of the end of chapter problems to produce an interesting space flight simulation. [For those of you going into Mechanical Engineering next year : The method described here for determining the X and Y components of forces that act at fixed directions parallel and perpendicular to a vehicle which changes orientation can be extended to simulations such as jet aircraft provided provision is made for the fact that the aircraft may be oriented at some angle to the current velocity. That is, the velocity of the aircraft is not always along its longitudinal axis.]

7.1 General Components of 2-D Central Forces (The Gravitational Force)

Two dimensional simulations of objects moving under the influence of **central forces** such as gravity and the Coulombic force are complicated by the fact that that the forces, even when constant in magnitude, have continually varying directions. General expressions are presented in this section that will give the correct direction and magnitude of gravitational central force components. The technique presented can also be applied to certain circular motion situations in which a force, such as a tension or a normal force, acts toward the center of circle over which the object is locally moving.

The gravitational force acting on an object in the Earth's gravitational field varies in magnitude with the object's distance from the Earth, and varies in direction as the object changes location relative to the position of the Earth. The magnitude of the gravitational force of the Earth on an object of mass M is given by

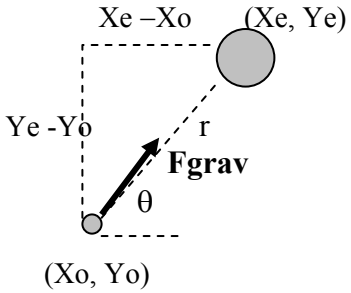
$$F_{grav} = \frac{G M_e M}{r^2}$$



The force is always be directed inward toward the center of the Earth (with an equal and oppositely directed force acting on the Earth itself). During a typical simulation, such as a satellite orbiting the Earth, the location of the object (Xo, Yo) is known at the beginning of each iteration, as is the location of the center of the Earth (Xe, Ye) which is assumed to be fixed. The value of **r** is easily found from **r = Sqr((Xo – Xe)^2 + (Yo – Ye)^2)** which is then used to calculate the magnitude of the gravitational force.

Method 1 : Calculating components using the side length ratio definitions of sine & cosine

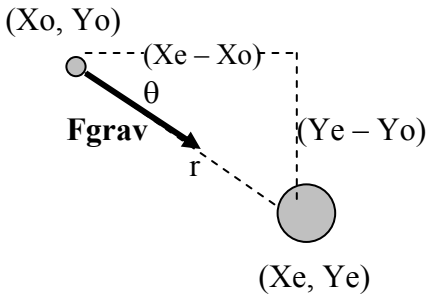
For the geometry of this two mass system there are 4 possible situations that must be considered corresponding to the object each of the four quadrants relative to the Earth. When the object is situated in the 3rd quadrant to the lower left of the Earth the gravitational force acts up and to the right so that both components are positive. Expressing the side lengths as (Xe – Xo) and (Ye – Yo) gives values of cos θ and sin θ which are both positive and hence produce positive components.



$$X_{comp} = F_{grav} \cos\theta = F_{grav} \frac{(X_e - X_o)}{r}$$

$$Y_{comp} = F_{grav} \sin\theta = F_{grav} \frac{(Y_e - Y_o)}{r}$$

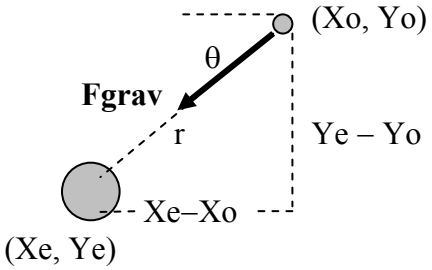
When the object is located in the 2nd quadrant above and to the left of the Earth, the gravitational force acts down and to the right; the cosine definition must produce a positive value to make the X component positive and the sine definition must produce a negative value to make the Y component negative. Here again expressing the side lengths as (Xe – Xo) and (Ye – Yo) gives correct signs for cosθ and sinθ Since (Xe – Xo) is positive and (Ye – Yo) is negative.

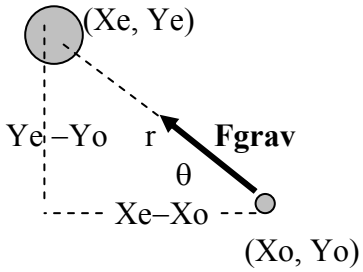


$$X\ comp = Fgrav \cos\theta = Fgrav \frac{(Xe - Xo)}{r}$$

$$Y\ comp = Fgrav \sin\theta = Fgrav \frac{(Ye - Yo)}{r}$$

A check of the remaining two locations reveals that (Xe – Xo) and (Ye – Yo) both give negative values when the object is in the 1st quadrant which lead to negative components consistent with the force acting down and to the left. Similarly, in the 4th quadrant the difference (Xe – Xo) is negative while the difference (Ye – Yo) is positive, both of which are consistent with the force acting up and to the left.





Substituting **Fgrav = GMeM/r²** into the general forms given above results in the following two expressions that can be used in simulations :

$$\mathbf{X\ comp = \frac{G\ Me\ M\ (Xe - Xo)}{r^3}} \qquad \mathbf{Y\ comp = \frac{G\ Me\ M\ (Ye - Yo)}{r^3}}$$

The similarity of the two expressions suggests that a single function can be defined with the value of (**Xe – Xo**) and (**Ye – Yo**) passed along with **r** via the argument. In fact, as demonstrated below, a single function can be used to describe the gravitational force due to any body acting on a mass.

The following code fragments will give the total gravitational force components due to both the Earth and the Moon acting on an object.

The value of the object's **mass** could be prescribed inside the **Run** routine but would then need to be passed as an argument to the **Grav_Force** function. I have yet to determine whether execution is faster with GG, M_Earth, and M_Moon defined globally (as here), or defined within the actual **Grav_Force** function.

```
Option Explicit
Const GG As Double = 6.67259E-11: Const M_Earth As Double = 5.98E+24
Const M_Moon As Double = 7.36E+22
Const mass As Double = 1
```

The **Grav_Force( )** function must be defined in the worksheet code window, and not in a module, since the value of **mass** of the object is not being passed through the argument. **M** represents the mass of the astronomical body such as the Earth or Moon. The value of **X** that is passed to the function is one of the displacement components **Xe – Xo** , **Ye – Yo** , etc.

```
Function Grav_Force(X, r, M)
Grav_Force = GG * M * mass * X / r ^ 3
End Function
```

Method 1 calculation of force components.

The order of Xo and Xe , etc is irrelevant here because of the squares.

Inside the main loop of the **Run** subroutine :

```
rE = Sqr((Xo - Xe) ^ 2 + (Yo - Ye) ^ 2)
rM = Sqr((Xo - Xm) ^ 2 + (Yo - Ym) ^ 2)
```

X component of gravitational force of the Earth.

X component of gravitational force of the Moon.

```
Fnetx = Grav_Force(Xe – Xo, rE, M_Earth)  + Grav_Force(Xm – Xo, rM, M_Moon)
Fnety = Grav_Force(Ye – Yo, rE, M_Earth)  + Grav_Force(Ym – Yo, rM, M_Moon)
```

Y component of gravitational force of the Earth.

Y component of gravitational force of the Moon.

Method 2 : Finding the components by determining the angle and using trig functions

Although I still use the method just described for calculating components, it was developed in 2001 when I was unaware of the **ATAN2(X, Y)** worksheet function and the **Application.WorksheetFunction.** method of accessing worksheet functions from the VB Editor. An alternative approach is to find the angular position **Theta** of the object and then form the components using the cosine and sine functions.

```
Theta = Application.WorksheetFunction.Atan2(Xe – Xo, Ye – Yo)
Fnetx = Grav_Force(rE, M_Earth) * Cos(Theta)
Fnety = Grav_Force(rE, M_Earth) * Sin(Theta)
```

Method 2 calculation of force componnents.

The definition of the gravitational force function has been simplified :

```
Function Grav_Force(r, M)
Grav_Force = GG * M * mass / r ^ 2
End Function
```

I have not yet tested the speed of using **Atan2** and **Cos** and **Sin** in this way but would not at all be surprised if the older approach executes faster, which can be an issue in certain simulations. While the VB function **Atn(Y/X)** can be used to determine the angle, a few **If** statements are needed to avoid overflows at angles of $\pm 90^{o}$ (for which X = 0), and to account for 2^{nd} and 3^{rd} quadrant angles.

7.2 The Motion of Charges and Masses in the Presence of Other Charges and Masses.

Charges and masses exert forces on one another as described by Coulomb's Law and Newton's Gravitational Law. This section uses the general expressions developed in the previous section to examine a satellite orbiting the Earth and the collision of a moving charged particle with a stationary charged particle.

7.2.1 Application # 18 : Orbiting Satellite

This simulation tracks the motion of the object around the Earth. The object is initially located at (Xo, Yo) with respect to an origin located at the center of the Earth, and is moving with a velocity (Vox, Voy). [The most convenient location is to assume the object is on the Y axis at (6.57E06 , 0) meters (about the height of the space shuttle = 200 km), and has an entirely horizontal velocity : **V** = (7793 , 0) m/s.]

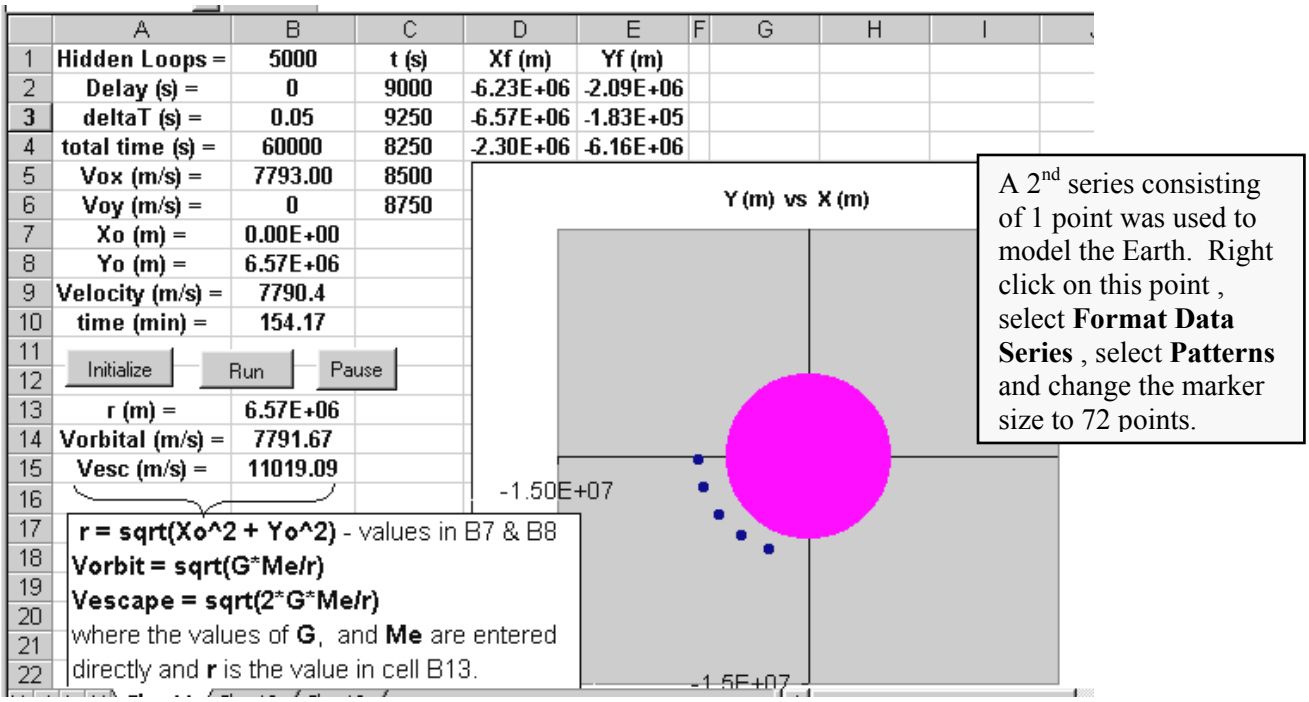
The situation is characterized by the presence of a single force, gravity, acting on the satellite. The direction of the gravitational force is always in toward the center of the earth. **Method 1** described in **section 7.1** is used to obtain the force components using the custom function **Grav_Force(X, r, M)**

```
Function Grav_Force(X, r, M)
Grav_Force = GG * M * mass * X / r ^ 3
End Function
```

Note : The fixed, or other object, must have its coordinates placed first .

```
Fnetx = Grav_Force(Xe – Xo, r, M_Earth)
Fnety = Grav_Force(Ye – Yo, r, M_Earth)
```

The 2-D projectile motion application (# 13) of chapter 6, with the addition of hidden loops, served as the shell for this application.



A 2nd series consisting of 1 point was used to model the Earth. Right click on this point , select **Format Data Series** , select **Patterns** and change the marker size to 72 points.

```
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 t As Double
Dim Xe As Double, Ye As Double, r As Double
Const GG As Double = 6.67259E-11: Const M_Earth As Double = 5.98E+24
' Const M_Moon As Double = 7.36E+22
Const mass As Double = 1

Private Sub cmdInitialize_Click()
Vox = Worksheets("sheet1").Range("B5")
Voy = Worksheets("sheet1").Range("B6")
Xo = Worksheets("sheet1").Range("B7")
Yo = Worksheets("sheet1").Range("B8")
Xe = 0: Ye = 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 Grav_Force(X, r, M)
Grav_Force = GG * M * mass * X / r ^ 3
End Function


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
Dim JJ As Integer, 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")

bStop = False
```

(Xe, Ye) is the center of the Earth.

Method 1 calculation of force component

```
Do
   For I = 1 To Hidden_Loops

      r = Sqr((Xe - Xo) ^ 2 + (Ye - Yo) ^ 2)

      Fnetx = Grav_Force(Xe - Xo, r, M_Earth)
      Fnety = Grav_Force(Ye - Yo, r, M_Earth)

      ax = Fnetx / mass
      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)

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

   Next I

   Call TimeDelay(Delay)

   Worksheets("sheet1").Cells(Row, 3) = t
   Worksheets("sheet1").Cells(Row, 4) = Xf
   Worksheets("sheet1").Cells(Row, 5) = Yf
   Worksheets("sheet1").Range("B9") = Sqr(Vfx ^ 2 + Vfy ^ 2)
   Worksheets("Sheet1").Range("B10") = t / 60
      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

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

Private Sub cmdClear_Click()
Worksheets("sheet1").Range("C2:E6") = 0
End Sub
```

7.2.2 General Components of Coulombic Forces

The general expressions developed to determine the components of the gravitational force can be modified to treat Coulombic forces. The difference between the two forces, of course, is that while the gravitational force is strictly attractive with positive masses positive, charges may have opposite polarities with the force being either attractive or repulsive. A **Method 2** calculation of first finding the angle and then applying the **Cos()** and **Sin()** functions is conceptually simpler, although **Method 1** calculations execute faster particularly when a large number of charge calculations is being described.

Following the approach of section 7.1 in which the coordinates of the "attractor" object are located first in the inverse tangent function the force components of the force of a charge 1 **on** a charge 2 are found as :

Method 2 calculation of Coulombic force components

```
F12 = k * Q1 * Q2 / r12 ^ 2

Theta = Excel.WorksheetFunction.Atan2(X1o - X2o, Y1o - Y2o)
F12x = –F12 * Cos(Theta): F12y = –F12 * Sin(Theta)
```

Negative signs have been included in the components in recognition that, unlike the gravitational force, it is charges of opposite polarity that that are attractive. The negative signs can be dispensed with if the attractor charge (q1 in this example) is placed second in the Atan2() components :

```
Theta = Excel.WorksheetFunction.Atan2(X2o - X1o, Y2o - Y1o)
F12x = F12 * Cos(Theta): F12y = F12 * Sin(Theta)
```

A **Method 1** calculation is made as follows :

```
F12 = k * Q1 * Q2 / r12 ^ 2

F12x = –F12 * (X1o - X2o) / r12: F12y = –F12 * (Y1o - Y2o) / r12
```

or by using a custom function to take the general component :

Method 1 calculation of force components.

```
Fnetx = –Coul_Force(X1o – X2o, r12, q1, q2)
Fnety = –Coul_Force(Y1o – Y2o, r12, q1, q2)
```

where the function is defined as :

```
Function Coul_Force(X, r, qA, qB)
Coul_Force = k * qA * qB * X / r ^ 3
End Function
```

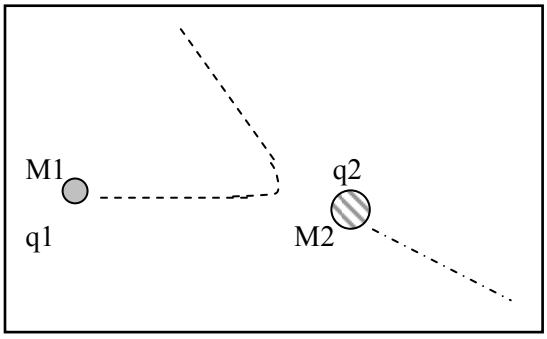
In either case, the force components of q2 on q1 are obtained by applying the 3rd Law :

```
F21x = –F12x: F21y = –F12y
```

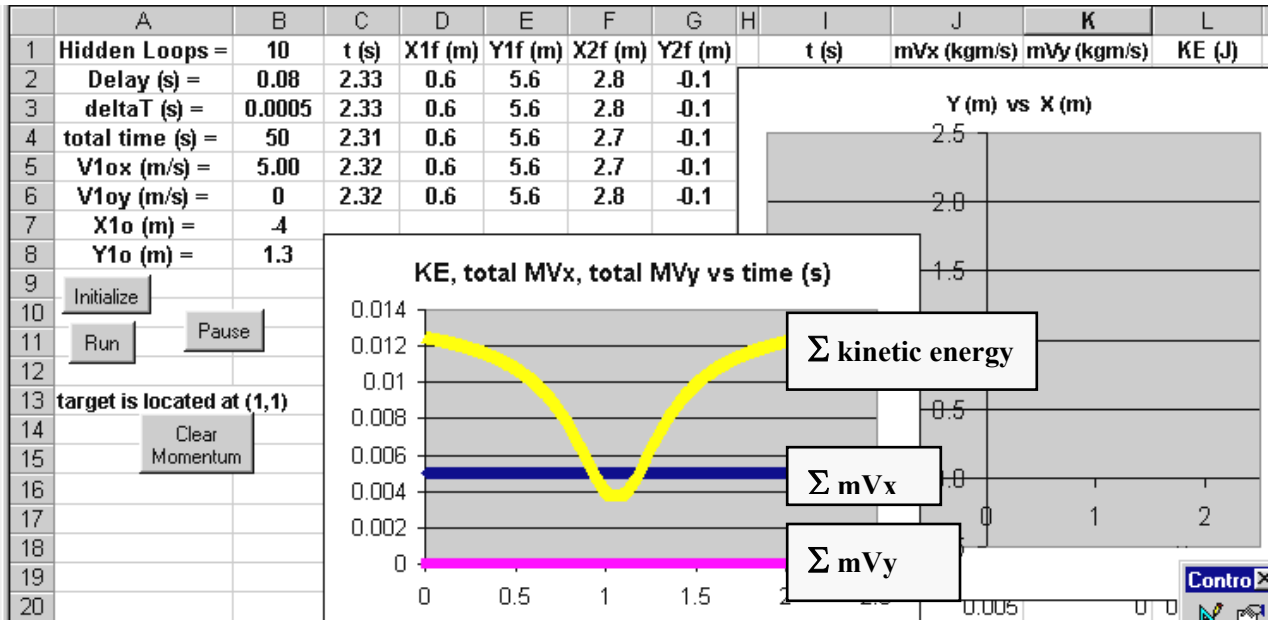
.

Application # 19 : A Moving Charge Collides with a Stationary Charge

Mass M_1 (1 gram) starts at location (– 4, 1.3) meters with a velocity (5, 0) m/s. It moves toward mass M_2 (4 grams) located at (1, 1) m which is at rest. Both masses carry a charge of 1 μC.



This simulation can be converted to model a 2-D collision of magnetic pucks on a Luctor table for an elastic collision experiment. Modifications to the net force description will also allow it to be used for a Rutherford scattering simulation.



```
Option Explicit
Dim Delay As Double, bStop As Boolean, row As Integer, row2 As Integer
Dim X1o As Double, Y1o As Double, X2o As Double, Y2o As Double
Dim V1ox As Double, V1oy As Double, V2ox As Double, V2oy As Double
Dim t As Double
Dim M1 As Double, M2 As Double, Q1 As Double, Q2 As Double

Private Sub cmdClear_Click()
Worksheets("sheet1").Range("I2:L5000").Clear
End Sub

Private Sub cmdInitialize_Click()
V1ox = Worksheets("sheet1").Range("B5"): V1oy = Worksheets("sheet1").Range("B6")
X1o = Worksheets("sheet1").Range("B7"): Y1o = Worksheets("sheet1").Range("B8")
X2o = 1: Y2o = 1: V2ox = 0: V2oy = 0
row = 2: t = 0: row2 = 2
Q1 = 0.000001: Q2 = 0.000001: M1 = 0.001: M2 = 0.004
Worksheets("sheet1").Range("C2:C6") = 0 ' next statements make Clear redundant
Worksheets("sheet1").Range("D2:D6") = X1o: Worksheets("sheet1").Range("E2:E6") = Y1o
Worksheets("sheet1").Range("F2:F6") = X2o: Worksheets("sheet1").Range("G2:G6") = Y2o
End Sub

Private Sub cmdPause_Click()
bStop = True
End Sub

Private Sub cmdRun_Click()
Dim deltaT As Double, Total_time As Double
Dim JJ As Integer, I As Integer, Hidden_Loops As Integer
Const k As Double = 8990000000#
Dim F12 As Double, F21 As Double, r12 As Double
Dim F12x As Double, F12y As Double, F21x As Double, F21y As Double
Dim X1f As Double, Y1f As Double, X2f As Double, Y2f As Double
Dim V1fx As Double, V1fy As Double, V2fx As Double, V2fy As Double
Dim a1x As Double, a1y As Double, a2x As Double, a2y As Double
Dim Theta As Double

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

bStop = False
```

```
Do
   For I = 1 To Hidden_Loops
      r12 = Sqr((X1o - X2o) ^ 2 + (Y1o - Y2o) ^ 2)

      F12 = k * Q1 * Q2 / r12 ^ 2

     Theta = Excel.WorksheetFunction.Atan2(X1o - X2o, Y1o - Y2o)
     F12x = –F12 * Cos(Theta): F12y = –F12 * Sin(Theta)

      F21x = –F12x: F21y = –F12y

      a1x = F21x / M1: a1y = F21y / M1
      a2x = F12x / M2: a2y = F12y / M2

      X1f = Position(X1o, V1ox, a1x, deltaT): Y1f = Position(Y1o, V1oy, a1y, deltaT)
      V1fx = Velocity(V1ox, a1x, deltaT): V1fy = Velocity(V1oy, a1y, deltaT)

      X2f = Position(X2o, V2ox, a2x, deltaT): Y2f = Position(Y2o, V2oy, a2y, deltaT)
      V2fx = Velocity(V2ox, a2x, deltaT): V2fy = Velocity(V2oy, a2y, deltaT)

      X1o = X1f: Y1o = Y1f: V1ox = V1fx: V1oy = V1fy
      X2o = X2f: Y2o = Y2f: V2ox = V2fx: V2oy = V2fy

      t = t + deltaT
   Next I

   TimeDelay (Delay)

   Worksheets("sheet1").Cells(row, 3) = t
   Worksheets("sheet1").Cells(row, 4) = X1f: Worksheets("sheet1").Cells(row, 5) = Y1f
   Worksheets("sheet1").Cells(row, 6) = X2f: Worksheets("sheet1").Cells(row, 7) = Y2f
   Worksheets("sheet1").Cells(row2, 9) = t
   Worksheets("sheet1").Cells(row2, 10) = M1 * V1fx + M2 * V2fx
   Worksheets("sheet1").Cells(row2, 11) = M1 * V1fy + M2 * V2fy
   Cells(row2, 12) = 0.5 * M1 * (V1fx ^ 2 + V1fy ^ 2) + 0.5 * M2 * (V2fx ^ 2 + V2fy ^ 2)
      For JJ = 1 To 50
        DoEvents: DoEvents

      Next JJ

   row = row + 1: row2 = row2 + 1
   If row = 7 Then row = 2
Loop Until t >= Total_time Or bStop = True
End Sub
```

The 3rd Law is used to obtain the force components acting on q1.

A separate force and kinematics description is used for each object that is present.

Complete data was printed out for the total momentum in the X & Y directions and the total kinetic energy so that momentum and energy variations could be examined.

Interpreting the graphs : The two momentum graphs shown on the worksheet indicate conservation of momentum. The KE graph shows that KE is conserved if points far enough before and after the collision are considered. However, in the vicinity of the collision the total KE decreases with the difference being stored as electrostatic potential energy. If a term describing the potential energy of the system, **k q1 q2/ r12** , had been added to the kinetic energy the total energy of the system would have remained constant. When the two masses are equal the angle between the scattered trajectories would be 90^{o} as expected for elastic collisions. This force **F12** definition in this application can be modified with an **If…Then…Else** statement to observe both "plum pudding" and Rutherford scattering. (problem # **8**)

7.3 "Contact" Collisions versus "Field" Collisions

Application # 19 described the motion of a moving positive charge as it approached a stationary positive charge (which was subsequently put into motion). The comment was made that the situation was equivalent to the mechanics lab involving the 2-D collision of two magnetic pucks on a horizontal air table [2]. The collision in both cases was not what you'd normally expect since there is no actual contact between the objects; instead the collision was between each object and the field of the other object. In fact, it might be imagined that the collision was between the two fields. Attractive forces can also lead to situations that are also collision-like; for example, one way of analyzing the Jupiter slingshot effect (find text ref : old H & R and The Physics Teacher) is to model it as an elastic collision and use 1-D conservation of momentum plus conservation of energy. All interactions between moving charges or masses might therefore be considered as collisions which sometimes occur at close encounters and look like "real" collisions, or at other times occur at larger distances and not really seem to be collisions at all [3]. I'll be referring to such collisions as **field collisions**. Collisions in which physical contact occurs, and which are encountered in any mechanics course, will accordingly be referred to as **contact collisions**.

While field collisions are easy to model and simulate using the dynamics and kinematics of ordinary incremental iteration, contact collisions can only be modelled by explicitly applying the X & Y momentum equations plus arbitrary assumptions as to the variation in kinetic energy of each object [4]. We cannot just let time march along in an incremental fashion and recalculate net forces, velocities, and positions because the nature of the bodies colliding is not known. The shape and elasticity of the surfaces and how they deform, or fracture, and the implications on the energy variations are simply not available (and would be extremely difficult to account for even if they were available) . Contact collisions are therefore not easy to deal with and will generally be avoided unless specific information is given concerning the energy variations of each object. The coalescing matter problem (# 16) has 3 small but dense masses which are allowed to coalesce or come together in much the same way that it is believed the universe formed. The simulation uses field collisions when the masses are not in contact and a special subroutine when they do contact. [E-mail me if you're interested : kenton@champlaincollege.qc.ca]

---

[2] Application # 19 could be used to run simulations of the mechanics collision lab provided the strength of the two electric fields could be made equivalent to the strength of the two magnetic fields. One way of doing this would be to hook a spring scale onto one of the pucks and then pull it toward the second puck noting the force required to produce contact. The center to center distance between the pucks can be recorded on a sheet of newsprint and measured. The magnitude **q** of two equal charges separated by the same distance and producing the same force is then determined from Coulomb's Law and used along with initial location and velocity data for the moving puck obtained from the collision lab data sheet. Note that the center of the stationary puck should not be along the X axis as defined by the initial velocity of the moving puck (otherwise a 1-D collision would be obtained). I'll be trying this out in November 2004.

[3] Whatever the situation, momentum will be conserved for the entire system (it always is). And for non contact "field" collisions energy will also be conserved since no non-conservative forces are present.

[4] Which must be tested to ensure that momentum is conserved and that the final total energy is not greater than the initial total energy.

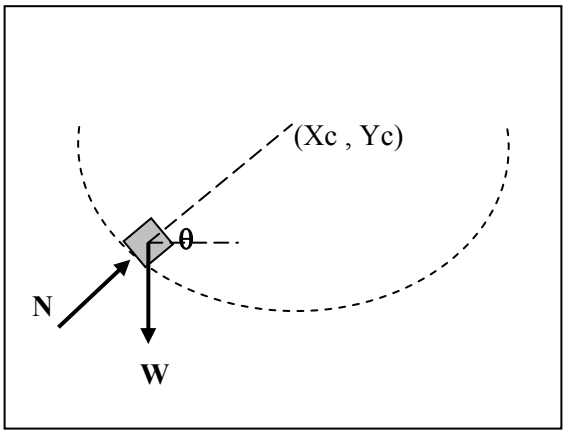
7.4 Iterative Analysis of Circular Motion Problems

Circular motion problems have one or more forces that are directed toward the center of the circle over which the object is locally moving, and as the object moves these forces will change orientation. As you'll recall from your mechanics course, it is the force components acting in the radial direction (toward the center of the circle) that cause the object to turn, or deviate from motion in a straight line. A conventional analysis uses a **radial** and **tangential** coordinate system to separate the forces that cause a change in direction of the velocity and those that cause a change in magnitude of the velocity (the tangential forces). The general expressions given in section 7.1 for obtaining the components of central forces can be usefully extended to circular motion problems since these forces are also directed either toward or away from some central point. The example presented below is a typical problem that describes a block sliding in a frictionless bowl; it can be modified by replacing the normal force, **N**, with a tension to create a pendulum, or linked to an inverted circular section to form a roller coaster (refer to the problems at the end of the chapter).

7.4.1 Application # 20 : Block Sliding in a Frictionless Bowl

For the purposes of an incremental iterative analysis the X and Y components of the forces are required.

$$\Sigma F_x = N \cos\theta$$
$$\Sigma F_y = N \sin\theta - W$$



Presumably the fact that the block has its motion constrained to follow a circular path needs to be included in the analysis. One possible approach clearly would be to constrain the block to follow the circle by programming the object's coordinates to lie on a circle with fixed local radius. This would not only defeat the point of the analysis but is also unnecessary. The key lies in the value of **N**.

In circumstances where an object moves along a horizontal or inclined surface the magnitude of **N** is based on the component of the weight (as well as possibly other force components present). This situation, however, is much subtler than it appears as inspection of the radial dynamics equation reveals :

$$\Sigma F_r = -W\cos(90-\theta) + N = m V^2/r$$ (recall that the radial direction is positive inward)

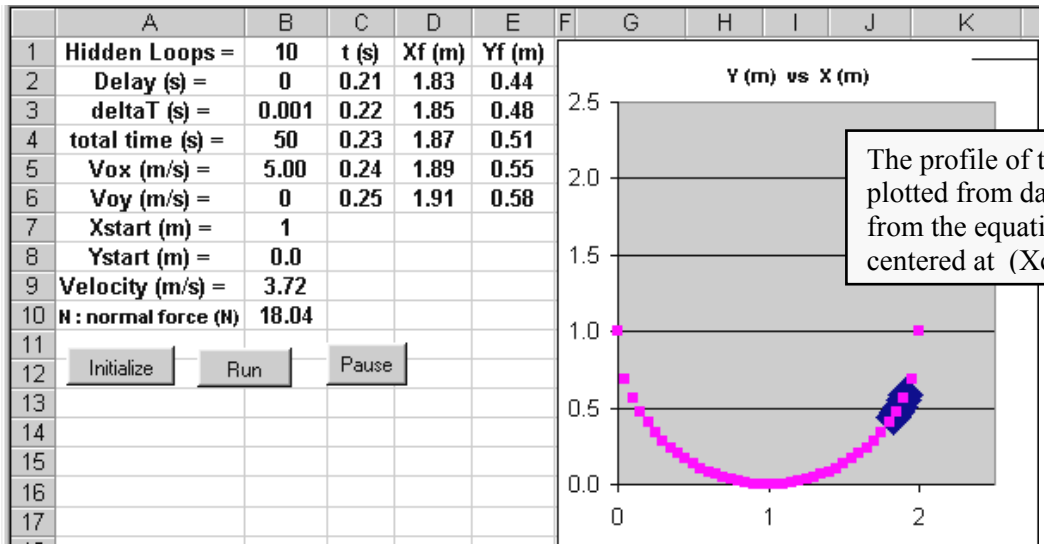
$$N = W\sin\theta + m V^2/r$$

Here normal force **N** is what might be described as a **"feedback" force** since it's value varies in response to the speed of the object such that the faster the object is moving the greater the magnitude of **N** [1].

---

[1] This is very interesting because we usually tend to view forces as being strictly **causal** and fixed in magnitude as an object moves over a particular trajectory; that is, that they can only be the cause of change in motion (really rate of change of motion). Yet here is a force which responds to the motion itself by adopting a value which depends on the speed of the object. The faster the block is moving the greater the contribution of the $m V^2/r$ term to the value of **N**. [We need to be careful here: the effect produced by the net force is not velocity but rate of change of velocity. The feedback is not from the effect, but from "the effect of the effect" in terms of the new velocity state that the object may have reached.]

The 2-D projectile motion application (# **13**, chapter 6), with the addition of hidden loops, served as the shell for this application.



The profile of the bowl was plotted from data obtained from the equation of a circle centered at (Xc, Yc)

```
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 t As Double
Dim Xc As Double, Yc As Double, r As Double

Private Sub cmdInitialize_Click()
Vox = Worksheets("sheet1").Range("B5")
Voy = Worksheets("sheet1").Range("B6")
Xo = Worksheets("sheet1").Range("B7")
Yo = Worksheets("sheet1").Range("B8")
Xc = 1: Yc = 1
r = Sqr((Xc - Xo) ^ 2 + (Yc - Yo) ^ 2)  '  discuss fixed r versus variable r
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 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 JJ As Integer, I As Integer, Hidden_Loops As Integer
Dim Theta As Double, N As Double, W As Double, V As Double
```

r does not need globally declared since its value will be calculated at the beginning of each iteration, and there'' no real need for a value in the **Initialize** routine.

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

mass = 1: W = mass * 9.81
bStop = False

Do
   For I = 1 To Hidden_Loops

      Theta = Application.WorksheetFunction.Atan2(Xc - Xo, Yc - Yo)
       r = Sqr((Xc - Xo) ^ 2 + (Yc - Yo) ^ 2)
      V = Sqr(Vox ^ 2 + Voy ^ 2)
      N = W * Sin(Theta) + mass * V ^ 2 / r

      Fnetx = N * Cos(Theta)
      Fnety = N * Sin(Theta) - W

      ax = Fnetx / mass
      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)

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

   Next I
   Call TimeDelay(Delay)
   Worksheets("sheet1").Range("B9") = Sqr(Vfx ^ 2 + Vfy ^ 2)
   Worksheets("sheet1").Range("B10") = N
   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
```

The expressions for **r** and **V** could be substituted directly into the expression for **N**.

Although it might intuitively seem that **r** should be fixed at its initial value, this is unnecessary. Provided **Δt** is small enough, the circular motion "spontaneously" reproduces itself. Try increasing the value of **Δt** and write out and observe how approximation errors eventually corrupt the motion by altering **r** .

```
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

Private Sub cmdClear_Click()
Worksheets("sheet1").Range("C2:E6") = 0
End Sub
```

For the data given on the worksheet the value Vox = 5 m/s causes the block to slide above a horizontal axis through the center of the bowl into an impossible situation in which the normal force **N** has to become negative. Code needs to be added to detect this and then set the net force components to the freefall condition.

"Wonky" graphs : Due to approximation error (**Δt** needs to be very small) the system will become unstable such that the circle of motion is broken (ultimately accelerating the object off the graph -- and into the Delta quadrant). The effect is more pronounced if unrealistic data is used leading to a negative contact force **N**.

Adding Friction : The presence of friction in this and similar problems could not be treated in your mechanics course because the value of the frictional force varied with the changing value of normal force. This is not an issue here; all that is required is the addition of the X and Y components of the friction force to the expressions for the net force (plus, of course, **FF = μk*N**) .

One of the advantages of an incremental analysis is that it exactly reflects the behaviour of the real system. For example, the motion of a pendulum in real life occurs as an infinite series of incremental motions. The system at each moment responds to the current net force, moves ever so slightly so that the direction of the weight vector and magnitude of the tension change by a small amount, and then responds again to the new, slightly varied net force.

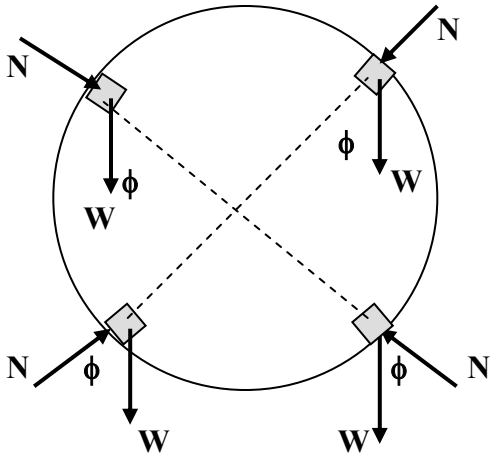
With incremental iteration there is no need for introducing the complexities, and approximations, of the differential equation solution; it mimics the causal relationship that actually occurs in nature where on an incremental level the behaviour is reassuringly that of a system moving under uniform acceleration.

A physical system responds to simple physical constraints as described by Newton's Laws and exists outside the sophisticated mathematical tools that may be devised to analyze it. Advanced computing power allows us to get back to the basics.

7.4.2 General Expressions for Objects Moving on the Inside and Outside of Circular Tracks.

The simulation of the block in the frictionless bowl (**Application # 20**) was developed assuming that the block would remain in the bowl; that is, below the center of the circle. However, for the given initial condition Vox = 5 m/s, the block rises just above the bowl and appears to continue to follow a circular path. Changing Vox to a larger value, say 15 m/s, indeed confirms that the trajectory above the bowl is circular, although it shouldn't be because the object is in freefall. There are two reasons that this happens : first, the net force has not been programmed to change to a freefall value once the block leaves the bowl so that the circular motion description of the net force continues, and second, the net force description in the lower portion of the bowl is, in fact, general and would apply everywhere inside a bowl that formed an enclosed sphere. This section will examine the radial circular motion equation for general points around the inside of a circular track and on the outside of the top of a circular track. The expressions developed will allow you to simulate motion around any vertical, circular track.

Situation A **Motion along the inside of a circle (equivalent to a mass on a string)**



ϕ is the acute angle between the **W** vector and the **r** direction. θ (not shown) is the total angle (measured w.r.t. the positive X axis) of the inward contact force **N** (or tension **T** in the case of a mass on a string).

Below the center of the circle : (With the "r" direction positive toward the center of the circle)

$$\Sigma F_r = N - W\cos\phi = m v^2/r \quad \text{and therefore} \quad N = +W\cos\phi + m v^2/r$$

For the two typical situations given above it can be shown that $\cos\phi = \sin\theta$ so that

$$N = W\sin\theta + m v^2/r \qquad \ldots (1)$$

Above the center of the circle : (With the "r" direction positive toward the center of the circle)

$$\Sigma F_r = N + W\cos\phi = m v^2/r \quad \text{and therefore} \quad N = -W\cos\phi + m v^2/r$$

For the two typical situations given above it can be shown that $\cos\phi = -\sin\theta$ so that again

$$N = W\sin\theta + m v^2/r$$

Since θ is measured with respect to the positive X axis the cosine and sine functions will give the correct signs at <u>any point</u> on the circle and the net force component expressions become :

Fnetx = N cos θ and Fnety = N sin θ – W … (2)

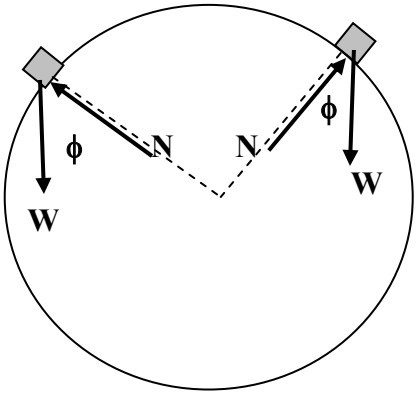
Equations (1) and (2) are thus general. Substituting T for N gives the description of a mass on a string moving in a vertical circle or as a pendulum.

The angle θ is obtained by a **Method 2** type calculation where the first term in the X and Y components is the fixed center of the circle Xc and Yc (or, similar to the Gravitational force discussion of section 7.1, the fixed location of what is "attracting" the object). Programmatically,

Theta = Application.WorksheetFunction.Atan2(Xc – Xo, Yc – Yo) … (3)

<u>Situation B</u> **<u>Motion along the outside of a circle</u>**

Assuming that the object is not rigidly attached to the track, it will be impossible for the object to move over the bottom of the circle (or even over the top beyond 48^o from the vertical when frictionless).



ϕ is the <u>acute</u> angle between the **W** vector and the **r** direction. θ (not shown) is the <u>total angle</u> (measured w.r.t. the positive X axis) of the inward contact force **N**

With the "r" direction positive toward the center of the circle :

$$\Sigma F_r = -N + W\cos\phi = m v^2/r \quad \text{and therefore} \quad N = +W\cos\phi - m v^2/r$$

At both typical locations of the block it can be shown that $\cos\phi = \sin\theta$ so that

$N = +W\sin\theta - m v^2/r$ …. (4)

Since θ is measured with respect to the positive X axis the cosine and sine functions will give the correct signs at <u>any point</u> on the circle and the net force component expressions are identical to equation (2) :

Fnetx = N cos θ and Fnety = N sin θ – W … (2)

The direction of outward vector N is determined by reversing the order of the components in the Atan2 expression on page 7-5 :

Theta = Application.WorksheetFunction.Atan2(Xo – Xc, Yo – Yc) …. (5)

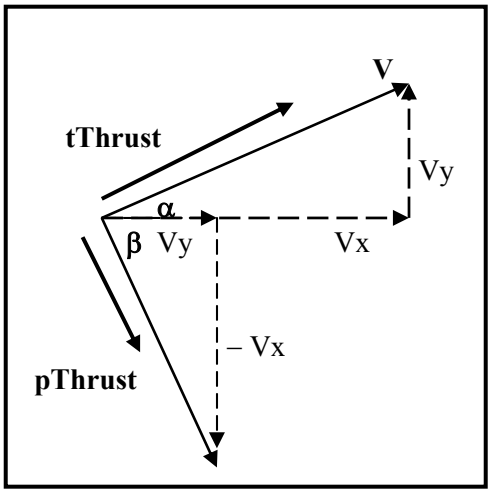
7.5 The Virtual Space Craft

The application presented in this section creates a virtual spaceship whose motion is controlled by rockets (thrusters) that act tangentially and perpendicularly to the trajectory[5] enabling the user to manoeuvre the space craft over the 2-D plane. [Recall that tangential force components cause an object to change speed while perpendicular force components are responsible for changes in direction. To begin with the gravitational forces of the earth and the Moon will be ignored; the spaceship is moving around empty space. Gravitational forces will ultimately be added to simulate full-blown space flight.

A standard 2-D incremental iteration program, such as used with the orbiting satellite application, is used. The major difficulty lies in determining the force components. Although the tangential and perpendicular force magnitudes may remain constant over relatively large intervals of time, the changing orientation of the spacecraft itself (due to the perpendicular thrust) means that the direction of the net force varies. And, unlike the previous applications in this chapter, these forces are not directed at a fixed point so that none of the previous techniques are available.

In order to convert the tangential and perpendicular thrusts into X and Y components it will be necessary to determine the tangential and perpendicular directions α and β and then use :

$$F_{netx} = tThrust\ \cos\alpha + pThrust\ \cos\beta$$
$$F_{nety} = tThrust\ \sin\alpha + pThrust\ \sin\beta$$



The positive direction for **tThrust** is in the tangential direction defined by the velocity. A negative **tThrust** acts opposite to the velocity and causes the object to decelerate. The positive direct ion for **pThrust** is in the right perpendicular direction and causes the object to turn to the right; a negative **pThrust** is in the left perpendicular direction and causes the object to turn left.

The angle α can be obtained from the velocity by recognizing that the instantaneous velocity of an object is by definition tangent to the trajectory. However, since it is **cosα** and **sinα** that we are after there is no need to actually find α ; instead the definitions **cosα = Vx/ V** and **sinα = Vy/ V** will be used. Values for **cosβ** and **sinβ** can be obtained by recognizing the following property of a vector : for any vector **A** = (A_x , A_y) the direction right perpendicular to **A** can be expressed as $(A_y , -A_x)$. In other words, a vector with an X component of A_y and a Y component of $-A_x$. [By right perpendicular we are referring to the perpendicular direction to the right of the original vector.] It follows that **cosβ = Vy/ V** and $\sin\beta = -Vx/V$.

[5] The possibility of the space craft slipping sideways such that the longitudinal axis is not aligned with the direction of the instantaneous velocity is real and presents the same problem as the jet aircraft referred to on pg 7-1. Fortunately, we'll assume that the spaceship is equipped with "smart" thrusters which automatically re-orient themselves tangential and perpendicular to the trajectory.

The contribution of **tThrust** and **pThrust** to F_{netx} and F_{nety} are then :

$$F_{netx} = tThrust\ \cos\alpha + pThrust\ \cos\beta$$

$$= tThrust\left(\frac{V_x}{V}\right) + pThrust\left(\frac{V_y}{V}\right)$$

$$F_{nety} = tThrust\ \sin\alpha + pThrust\ \sin\beta$$

$$= tThrust\left(\frac{V_y}{V}\right) + pThrust\left(-\frac{V_x}{V}\right)$$

The value of V in the program will be given the variable name **hypot** (for hypotenuse) and will be found from the values of V_{ox} and V_{oy} at the start of each interval. The code lines corresponding to the above expressions are :

```
hypot = Sqr(Vox ^ 2 + Voy ^ 2)
Fnetx = tThrust * (Vox / hypot) + pThrust * (Voy / hypot)
Fnety = tThrust * (Voy / hypot) + pThrust * (–Vox / hypot)
```

Similar to the Method 2 alternatives developed earlier, the **ATAN2( )** worksheet function can be used to determine α, from which $\beta = \alpha - \pi/2$.

```
Alpha = Application.WorksheetFunction.Atan2(Vox, Voy)
Beta = Alpha – 3.1415926535897/2
Fnetx = tThrust *cos(Alpha) + pThrust *cos(Beta)
Fnety = tThrust *sin(Alpha) + pThrust *sin(Beta)
```

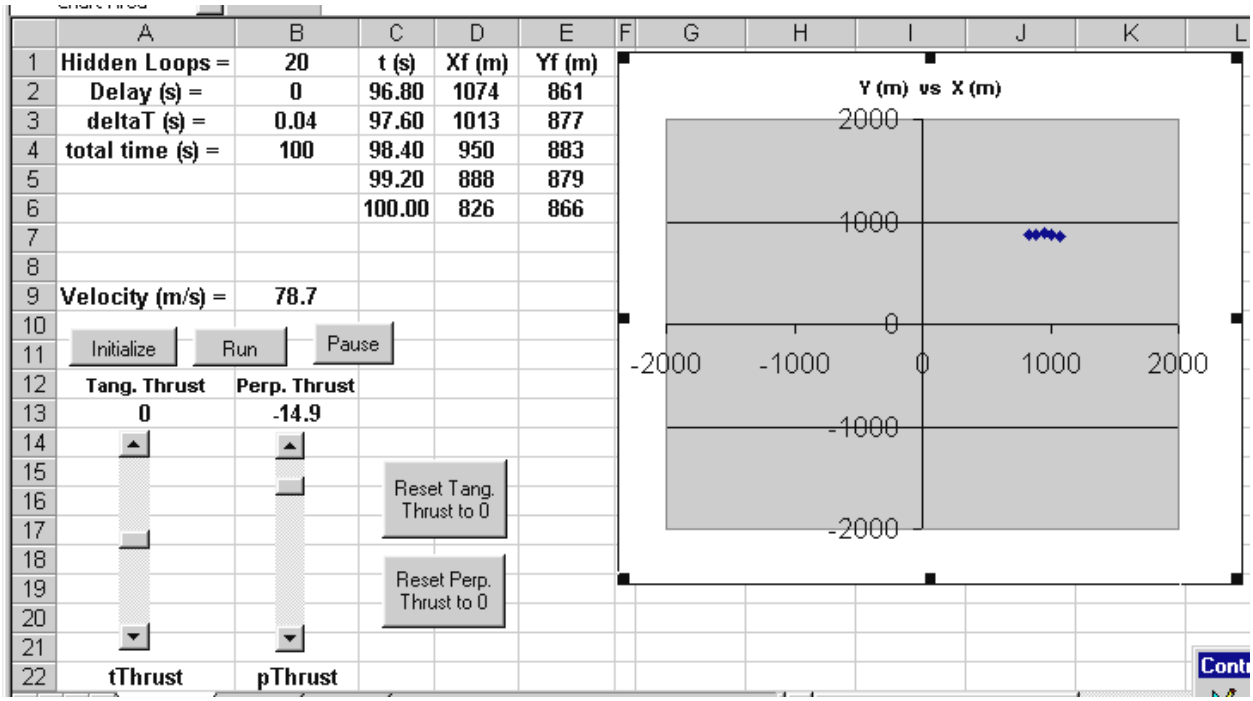
The huge benefit of this approach is that the code is that the code is much more readable, and familiar.

[The **ATAN2( )** approach is not used in this application as it is expected, perhaps erroneously, to run slower; and I'd been using Excel 97 on an old Dell up to September 2004]

Caution : Overflows will result in both approaches if the velocity magnitude is zero, such as when the spaceship starts from rest, or ever inadvertently (and very unlikely) decelerates to a zero velocity. It will therefore be necessary to add a small starting velocity. The program uses : **Vox = 0.1: Voy = 0**

Application # 21 : The Virtual Spaceship

Any 2-D incremental iteration program, such as the Orbiting Satellite can be used as the shell for this program. Set scroll bars **scbTangThrust** and **scbPerpThrust** so that **Min** = - 200 and **Max** = 200.




```
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
Dim tThrust As Double, pThrust As Double

Private Sub cmdInitialize_Click()
scbPerpThrust = 0: scbTangThrust = 0
Vox = 0.1: Voy = 0
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
```

Vox = 0.1 is used to avoid overflows in the calculation of **cosα**, **sinα**, **cosβ**, and **sinβ** at launch. **Zero velocity launch conditions <u>cannot</u> be used.**

```
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 JJ As Integer, 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 = 1
bStop = False
ax = 0

Do
   For I = 1 To Hidden_Loops
      tThrust = scbTangThrust.Value / 10
      Range("A13") = tThrust
      pThrust = scbPerpThrust.Value / 10
      Fnetx = Thrust_x(tThrust, pThrust, Vox, Voy)
      Fnety = Thrust_y(tThrust, pThrust, Vox, Voy)
      ax = Fnetx / mass
      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)

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

   Next I
   Call TimeDelay(Delay)
   Worksheets("sheet1").Range("B9") = Sqr(Vfx ^ 2 + Vfy ^ 2)
   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 cmdReset_pThrust_Click()
scbPerpThrust = 0
Range("B13") = 0
End Sub

Private Sub scbPerpThrust_Change()
Range("B13") = Format(scbPerpThrust / 10, " 0.0")
End Sub

Private Sub cmdReset_tThrust_Click()
scbTangThrust = 0
Range("A13") = 0
End Sub

Private Sub scbTangThrust_Change()
Range("A13") = Format(scbTangThrust / 10, " 0.0")
End Sub

Function Thrust_X(tThrust, pThrust, Vox, Voy)
Dim hypot As Double
hypot = Sqr(Vox ^ 2 + Voy ^ 2)
Fnetx = tThrust * (Vox / hypot) + pThrust * (Voy / hypot)
End Function

Function Thrust_Y(tThrust, pThrust, Vox, Voy)
Dim hypot As Double
hypot = Sqr(Vox ^ 2 + Voy ^ 2)
Fnety = tThrust * (Voy / hypot) + pThrust * (-Vox / hypot)
End Function

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

Private Sub cmdClear_Click()
Worksheets("sheet1").Range("C2:E6") = 0
End Sub
```

It would be more efficient to calculate the hypotenuse once in the main loop, and declare it globally under **Option Explicit**.

Run your program and familiarize yourself with the controls. Positive tangential thrust increases the speed; negative tangential thrust decreases the speed. Positive perpendicular thrust causes right turns; negative perpendicular thrust causes left turns. See how long you can maintain flight at speeds above 4 m/s (without adding **Delay** !).

The scroll bars can be difficult latch onto and to un-latch if thrust adjustments are attempted on-the-fly. It is less annoying, but less fun, to use the **Pause** button to leisurely adjust the thrust. The first adjustment to either scroll bar during a **Pause**, or after **Initialize**, is not displayed; however, the new value is used and will appear when the program is **Run**. A better approach is to replace the scrollbars with plus and minus increment buttons that add or subtract a fixed amount of thrust (ask me to show a sample).

Physics Digression :

Set the tangential thrust to zero once you're moving at about 1 m/s. Add enough perpendicular thrust so that a circular motion results. Watch to see if the velocity changes. It shouldn't, but it will ! Theoretically only tangential thrust should cause changes in speed (since force components perpendicular to the displacement do no work and therefore cannot change the KE). This means that if the tangential thrust is set to zero, the presence of a perpendicular thrust should only produce a <u>circular motion at constant speed</u>. A few test simulations will show this not to be the case, particularly at larger speeds (above 1 to 2 m/s in the current simulation).

This should be perplexing because we seem to be violating one of the principles of physics. There is a good explanation. Our model assumes that the directions of the perpendicular and tangential are fixed over the time interval Δt and produce constant values for **ax** and **ay** (remember, the kinematics equations are only valid for constant acceleration). Unfortunately the spaceship continually changes direction, even over the short time interval Δt , so that the initial directions of tangential and perpendicular thrust do not, in fact, remain tangential and perpendicular to the spacecraft. This leads to some of the perpendicular component acting along the altered tangential direction, and this is what causes the speed to vary. [Put another way : the program assumes that **tThrust** and **pThrust** have the same direction over Δt whereas they are actually changing direction as the spacecraft slightly changes orientation.] For an extremely small Δt the change in orientation of the spacecraft will not result in a significant component of the **pThrust** acting in the tangential direction, but execution will slow right down.

Problems [Chapter 7]

**** Include a Pause button and hidden loop in all programs ****

1. **2-D projectile motion with drag**. Modify the 2-D projectile application (# 13, chapter 6), to account for the presence of drag. The drag force, **Drag = bV^2**, will act opposite to the velocity and can be programmed in the same way as the virtual spaceship's tangential thrust. If α is the current angle of the velocity then **dragX = – Drag*cos(α)** and **dragY = – Drag*sin(α)** where α is obtained from the worksheet function **Atan2(Vox, Voy)** . Alternatively : **dragX = –Drag*Vox/V**; **dragY = –Drag*Voy/V**.

2. **Charges-in-a-Box**. This application builds on the Ball-in-a-Box program of Chapter 6 (**Application # 15,** pg 6-27) except that there are now 3 moving objects which are positive charges confined to a box 4 meters wide and 3 meters high. The initial velocities and starting points can either be read into the **Initialize** routine, or simply specified (refer to the test data at the end of the problem). Collisions with the walls will be assumed elastic such that the incident angle and "bounce" angle are equal, and can be programmed as in the Ball-in-the-Box by reversing the affected velocity component. No "contact" collisions will occur since the repulsive forces between the objects become huge at small distances; however, elastic "field" collisions between pairs of charges will take place. Because of the small dimensions of the enclosed space each charge will continually interact with the remaining two charges. [Each **Fnet** component of each charge will always have two terms.] Program a single outcome **If** statement to add the weight of each object to its Y net force if a Boolean flag has been activated (by pushing a button) : **If bWeight = True Then F1netY = F1netY – W1 : F2netY = …** **Test data** : ***(not yet available)*** q1 = X.YY E-06, X1o = m, Y1o = m; V1ox = m/s, V1oy = m/s ; q2 = X.YY E-06, X2o = m, Y2o = m; V2ox = m/s, V2oy = m/s ; q3 = X.YY E-06, X3o = m, Y3o = m; V3ox = m/s, V3oy = m/s ; q4 = X.YY E-06, X4o = m, Y4o = m; V4ox = m/s, V4oy = m/s **[***Add values using January, 2005 data]**

3. **Hole in the Box**. Modify the **If** statements that describe the walls of the box in problem # 3 to create a 0.5 meter hole somewhere in the ceiling (top) and right side. Extend the floor to between – 5 m and + 10 m. *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. While a bounce off the outside of the side walls is unlikely, watch out for bounces off the top*.

4. **Pinball Machine**. A simulation of a pinball machine can be made by modifying problem # 3 with the addition of a third stationary charge, and such that only one of the charges can move. The goal of the game is to get the moving charge out of either hole; and the shorter the time to do this, the greater the points awarded. Three scrollbars should be added to vary the magnitude of each of the fixed charges. By increasing a charge the user may (I haven't tried this yet) be able to "bump" the moving charge out one of the holes; conversely, decreasing a fixed charge value to zero would allow the moving charge to pass by undisturbed.

5. **Ideal Gas in a Container.** The objective is to modify the Ball-in-a-Box application (# 15 chapter 6) so that it very crudely simulates gas molecules moving in a container and calculates the pressure of the gas on the container. Add two more masses to the application and a scrollbar **scbTemp** (**Min = 0**, **Max = 5000**) that represents the temperature of the system temperature. The kinetic energy of the system will be assumed to vary by the factor **^{0}Kelvin/293** with room temperature having the factor 1. Each collision with a wall results in a change in momentum over a small period of time and causes a force on the wall (and on the mass). Although the duration of each collision is not known an average force can be obtained by calculating the total momentum change over a certain period of time and dividing by that time interval : **Faver** $= -\Sigma\Delta(m\mathbf{V})/\Delta t$ (the negative sign is a 3rd Law reversal of the

force on the charges to give the force on the wall) . Momentum is a vector quantity so that it is usually necessary to add the momentum change components vectorally. However, as the objective here is to find the scalar pressure (= Force/Area) , and since it has been assumed that the momentum changes only occur perpendicular to the walls, the magnitudes of all the momentum changes can be added and divided by the total "area", where the "area" is effectively the perimeter of the box 2*(L+H) . [We've only considered one 2-D slice of a 3-D box; if approximately the same activity was occurring in all slices over a width "W" then the total momentum change would be scaled up by a factor W, but so would the surface area. Real surface area (excluding the front and rear sides) = perimeter of a slice * width = 2*(L + H)*W where L is the length of a slice and H is its height. The W will cancel out so that the slice momentum change need only be divided by 2*(L + H).]

The **Initialize** routine should set the scrollbar value to 293 (room temperature in Kelvin). The factor **kk** by which each velocity component is multiplied should be defined as **kk = Sqr(scbTemp/293)** . The kinetic energy is being increased by the factor **kk** which is taken as 1 at room temperature; the square root recognizes that kinetic energy depends on V^2. Use initial velocity components of 15 m/s for each mass; increase the mass sizes to give more pressure. [If you're interested you might examine **Ideal Gas theory** to see if there's a relationship between temperature and kinetic energy !]
Test data : ***(not available yet)***

6. **Air table collision simulation** : Book time with the physics technician to use an air table to create a 2-D collision between two magnetic pucks (as you did in Mechanics). Measure the mass of each puck and use the procedure described in note 2, pg 7-12 to find the equivalent charge for the simulation. Modify Application # 19 pg 7-9 so that the moving charge is on the X axis and the stationary charge is off the axis. Compare the experimental final velocity components of each mass with the simulation results, also compare initial and final total energies (start the simulation with q1 farther away and note whether any initial kinetic energy is stored in the field at the experimental initial position).

7. **2-D Collision of two moving charges**. Modify the colliding charges application # 19 pg 7-9 so that both charges are initially moving from the left at some angle to the X axis. Since it since may be difficult to synchronize the collision (so that one charge doesn't arrive well before the other) you'll want to program a data testing button that uses the two initial locations and one of the velocities (magnitude & direction) to determine the magnitude and direction of the second velocity such that a simultaneous collision point is determined.

8. **Future : Rutherford Scattering**. Modify the colliding charges application # 19 (pg 7-9) to model the atom as a "plum pudding"; use an **If** statement plus Boolean variable to also allow a nucleus model to be used. ?? use two data sets so that the motion for each model can be simultaneously observed ?? use data from February 2004 simulation.

9. ***Future :*** **Energy to Assemble.** Calculate the energy to assemble a system of positive charges by evaluating the work to move each charge at constant speed from infinity, say (–20, 0) m, along a straight line. At each point on the trajectory the net force due to the remaining charges is calculated and the force required to keep the moving charge on its straight line path at constant speed is determined. The work associated with each incremental displacement would be kept is a running total. Each time the **Run** button is pressed a new charge would be brought in and the motion displayed on a graph (using different series as in the equipotential plot application). [Note that the running total for the energy must be declared globally.]

10. ***Priority*** : Add a magnetic field to perpendicular to the plane of motion of a charge. Use virtual spaceship perpendicular thrust method to obtain components of magnetic force (Fmag = qVB) . Spectrometer and particle simulations, add drag force (as in problem #1) to damp out the motion and create inward spiral. Simulate accelerator/ collider ?

11. ***Future* : Jupiter Slingshot** : Modify the orbiting satellite application (# 18 pg 7-6) so that the central mass is in also motion (all velocities wrt the Sun.) Use old textbook data, but may be difficult to find the right incoming location of the space craft.

12. **Space Flight Simulator**. Modify the Virtual Spaceship application (# 21 pg 7-19) by adding terms for the Earth's gravity and later the Moon's gravity. Display the orbital speed and escape speed at each location (relative to both the Earth & Moon). Remember that a zero velocity causes overflows if the first method is used to find the components so the craft will need a small velocity at launch. Achieving Earth orbit is difficult (unless you construct some sort of auto-pilot) so you'll probably want to start in some low Earth orbit (say 6.57 E06 m, or 200 km)

13. ***January 2005*** : **E Field Line Plotting**. Find old E field plot program using a single moving charge with a = 0 (constant velocity motion with direction adjusted to latest direction of Fnet). Complete data display. Add multiple lines as per equipotential application.

14. ***January 2005 :* 2-D Contact Collision** : as per December 2003 coalescing matter, but ignore gravitational effects (i.e. passive). Develop actual collision as subroutine for use in Coalescing matter. ****Derive the equations as part of section 7.3***. Emphasize that incremental iteration is not used in collision part of this application.

15. ***Future :* Collide and stick together collision**: Also requires momentum equations to model the collision. Test on stationary target, then use a routine (refer to problem # 7) to determine the data consistent with a simultaneous collision. Develop collision as a subroutine that can be used with random particle motion in which two moving objects that are within a certain distance "bond" to one another. Follow KE to check for patterns with different angles, probably need one of the objects to be stationary.

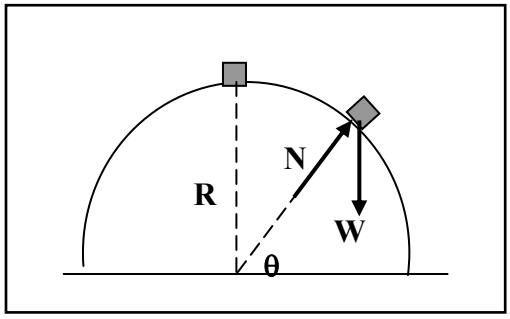
16. ***January, 2005 ?* : Coalescing matter :** Three dense spherical masses The gravitational forces act to bring the masses together in much the same way as many of the objects in the universe were formed. Two masses will collide when the distance between their centers is less than the sum of their radii; use the 2-D contact collision subroutine (problem # 14). The program, other than the contact collision, will be similar to the motion of the positive charges in the box, but with attractive rather than repulsive forces. [The collisions in that problem were not explicitly modelled as they are here.] Test data : use December 2003 values (M1 = kg, r1 = m; M2 = kg, r2 = m; M3 = kg, r3 = m) are initially located at (X1o = m, Y1o = m), (X2o = m, Y2o = m), and (X3o = m, Y3o = m).

Future problem development

[?? Model of charging capacitor : prohibitive due to number of charges and forces, and dynamic redistribution of charges along plates -- ?? model charges to be able to move in 1-D along the plates ?? use $q\Delta V$ and assume ΔV not affected by redistribution ??]

[coalescing charges: collide and stick together: similar to coalescing masses but simpler contact collsion; The collision must be analyzed using conservation of momentum. track system energy. ? confined to a box ? all start from rest ? convert lost KE to heat ? push button to release each mass and follow its trajectory]

20. **Ice Cube on a Hemisphere :** An ice cube has a velocity Vox at the top of frictionless hemisphere. It slides down along the hemisphere until it loses contact (N = 0) and then enters 2-D free fall. Construct a simulation of the motion of the ice cube along the hemisphere **and** then in free fall. Plot the semi-circular track on your graph.



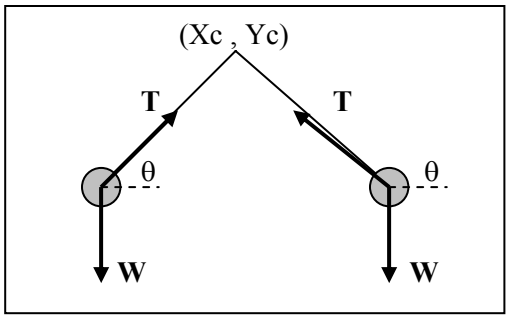
From a circular motion analysis :

$\Sigma F_r = W\cos(90 - \theta) - N = mV^2/r \quad \ldots (1)$

The value of N determined from (1) can be used to find the X and Y components of net force : $\Sigma F_x = N\cos\theta$ and $\Sigma F_y = N\sin\theta - W$.

Use an Energy analysis to theoretically find the angle at which the block flies off the hemisphere when starting from rest. Use R = 1 m and mass = 0.5 kg.

21. **Simple Pendulum :** The classical differential equation solution for the simple harmonic motion of a pendulum is valid only for small amplitude oscillations. An incremental iterative solution allows oscillations of any amplitude. Construct a simulation of the motion and add a second series consisting of the coordinates (Xc, Yc) and the middle (3^{rd}) comet tail point. Connect the two points of the second series with a straight line to represent the rope (no need to display the equation.



$\Sigma F_r = T - W\cos(90 - \theta) = mV^2/r \quad \ldots (1)$

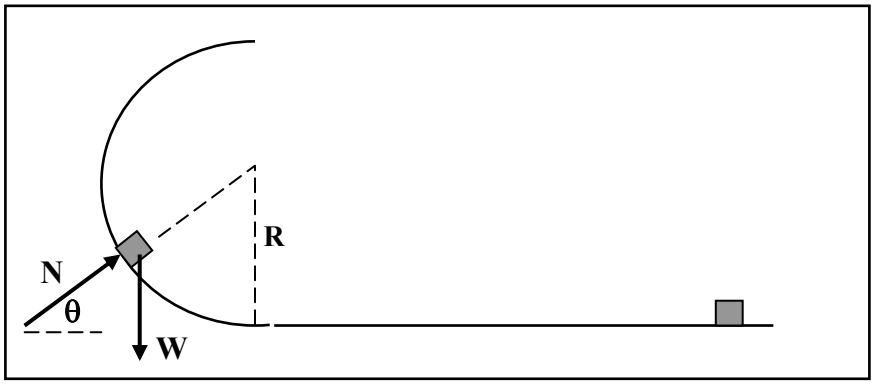
The value of tension T determined from (1) can then be used to find the X and Y components of the net force as $\Sigma F_x = T\cos\theta$ and $\Sigma F_y = T\sin\theta - W$.

Data : Xc = 2 m, **Yc = 2** m, starting point **Xo = 1** m, **Yo = 2** m, **Delay = 0**, **Hidden_Loops = 10**, **deltaT = 0.001** s For this starting point and **deltaT** accumulated error will cause the pendulum to go "wonky" after about 4 or 5 oscillations (the rope will lengthen, the pendulum will move above the center of the circle with the tension becoming negative, and finally the pendulum will fly off the graph). Try decreasing **deltaT** (and increasing **Hidden_Loops**) and then starting at a lower point. With sufficient velocity the pendulum will make a complete circle.

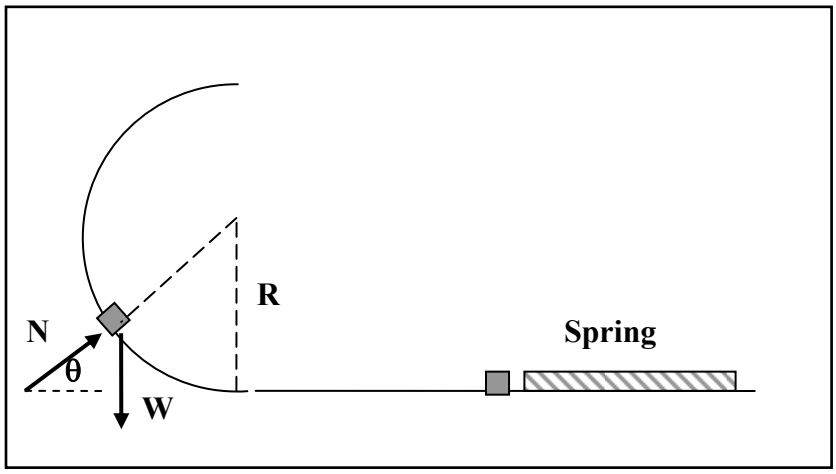
Write out complete data for the calculated length of the rope **Sqr((Xc – Xo)^2 + (Yc – Yo)^2)** ' the tension, and the angle (degrees) checking if the tension is negative and the rope lengthens. Plot on a single graph (K + Ug), K, and Ug versus time and interpret the results.

Option : Use an **If** statement to change the **Fnet** components to free fall when the tension goes negative. Then add an additional condition to the **If** to detect when the mass reverts to the circular motion. This will occur when the calculated rope length is greater than or equal the real (or starting) length, or, equivalently, when the latest point lies outside the equation of the circle. The "recaptured" circular trajectory will be at a slightly larger radius; you may wish to move it back on the original trajectory.

22. A 1 kg block has an initial velocity Vox and is travelling over a horizontal surface (μk = 0.2). The block encounters a curved, frictionless, vertical track. Construct a program that displays the trajectory of the motion. Plot the semicircular track on the graph.
Test your program by calculating two theoretical solutions : a) for the situation where the block falls off before reaching the top of the circle, and b) such that the block just reaches the top. Make your own choice as to which value(s) will be verified : N, V, or θ .
Option : plot complete graphs for (K + Ug), K, Ug , and Wnc versus time (all on the same graph).

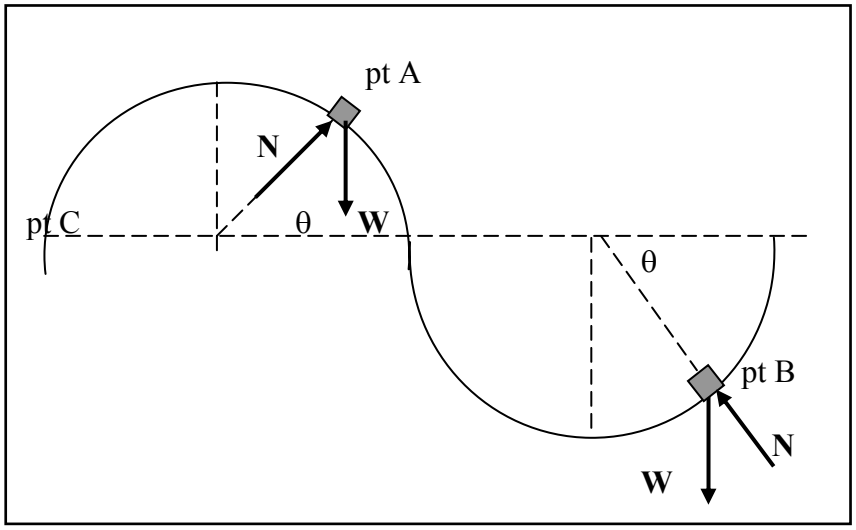


23. The previous problem is modified by the addition of a spring attached to a wall (not shown). The block is pushed against the spring (k = 20 N/m) compressing it by 0.5 meters and is released from rest. The spring is not attached to the block. The block moves over the horizontal surface (μk = 0.2) and encounters a curved, frictionless, vertical track. Construct a simulation that displays the trajectory of the motion. **Option :** account for the block going into freefall when it falls off before reaching the top, and when it's moving fast enough to reach the top and launch horizontally backward.
Option : add friction to the circular track as briefly described on page 7-16. Assume μk = 0.2.



24. A roller coaster is formed out of two back-to-back circles as shown below. Construct a simulation that shows the car moving left to right from pt C to pt A. Include the track profile in the graph. Temporarily add an outer loop that searches for the velocity at pt C that allows it to just clear the top of the left track. Find the apparent weight at pt B under those circumstances. Compare with the theoretical analysis.

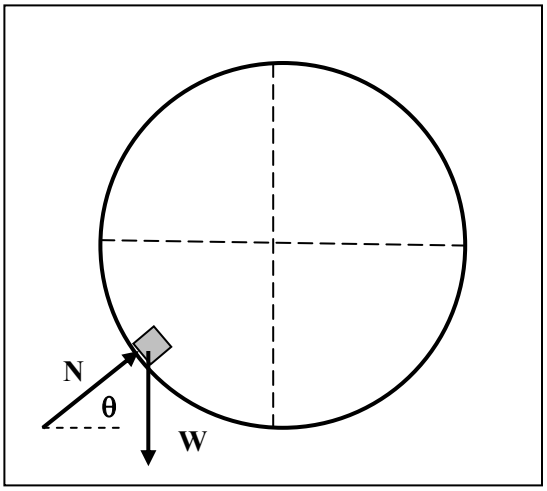
    Assume that it is now moving from the right to the left. How fast must it move at point B such that it just loses contact at point A.
    Option : Plot apparent weight N versus X on a graph that includes the track profile .

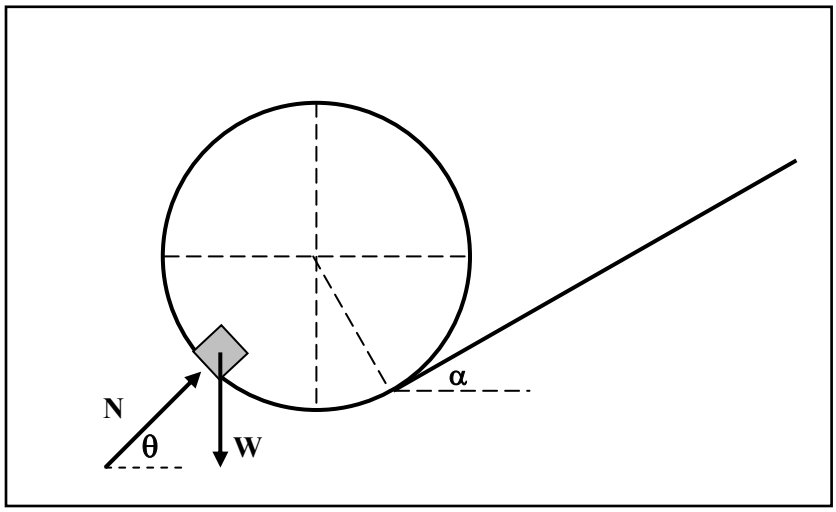


Note : This diagram is incorrect; a half circular track cannot be used for the over-the-top portion of the roller coaster since the coaster would be in free fall well before it encountered the lower, concave section of track.

I have specificationsand solutions for problems 22 to 24. Individuals not enrolled in the course can contact me at Kenton@champlaincollege.qc.ca .

25. A block is has a velocity of Vox at the bottom of a frictionless track that forms a vertical circle. Construct a simulation of the motion. Plot complete energy graphs versus X.
    Save the file under a new name and add friction as briefly described on pg 7-16. Assume μk = 0.1 .





Can you make it fall off the track twice (and recapture the track after a period of 2-D projectile motion) ?

Check for possible Trebuchet analysis perhaps involving the Fall, 2003 $\tau = I\alpha$ simulation.