Evaluación No. 1
Primera Evaluación de Programación Fortran
(2018-2)

García Castro, Jorge

Universidad de Sonora
Departamento de Física

Hermosillo, Son. 08 de noviembre de 2018

1. Introducción

En el presente reporte, parte de la primera evaluación de la materia Programación y Lenguaje Fortran impartida por el maestro Carlos Lizárraga Celaya, se adapta un código usado para resolver el movimiento de un proyectil estructurado el año 2006 propio de Alex Godunov de la Universidad .ºld Dominion Univercity".

En el código se pide modificar un programa con su respectiva subrutina, la cual contiene el método de Euler, para describir el moviemiento de un proyectil lanzado a diferentes inclinaciones y con o sin presencia de la resistencia del aire o fricción.

Sin más, quedan varias secciones donde aparece el código original y otra donde quedó el modificado, finalmente los resultados y uina conclusión del reporte.

2. Problema a resolver en el programa

Considera el movimiento realista en (x,y), de un proyectil esférico de masa Mp = 94.0 kg. y radio Rp = 0.10 m., disparado por un cañón en la Primera Guerra Mundial con una rapidez inicial v0 = 1600.0 m/s con un ángulo de 45.0 grados de inclinación desde la posición inicial (x0,y0) = (0.0,0.0). Toma el valor de la aceleración de la gravedad g = 9.81 m/s2.

Se trata de un movimiento donde se considera la fricción de aire. La densidad del aire esta dada por rho = 1.25 kg/m. Hay una constante de resistencia del aire para el proyectil, cuyo valor es C = 0.06 y una constante yrho = 1.0 x 104 (yrho = 1.0 x 1024 elimina el efecto de la resistencia del aire).

Se proporciona el código para resolver el movimiento del proyectil, y la subrutina del método de Euler. El código está preparado para 3 métodos de integración. En esta actividad, sólo utilizaremos el método de Euler modificado (key=0). Los métodos de Runge-Kutta los dejaremos para otra ocasión (key=1,2). Realiza los cálculos con un paso de integración dt=0.5 segundos. Se utiliza una variable de escala de unidades u=1.0 (metros), ó u=1000.0 (km). Use la que considere.

2.0.1. Códigos originales tomados de ODU por Alex Godunov

El primero es el código para resolver el movimiento del proyectil;

```
    Program projectile2
c-----------------------------------------------------------------
c Realistic projectile motion with air resistance
c method: program may call various ODU solvers
c   key = 0 modified Euler
c   key = 1 Runge-Kutta 4th order
c   key = 2 code Rkf45 (Runge-Kutta 4th-5th order)
c written by: Alex Godunov
c last revision: October 2006
```

```
c-------------------------------------------------------------------
c input from a file (self explanatory)
c    see file cannon.dat
c output ...
c    to a file named by a user
c-------------------------------------------------------------------
      implicit none
      Real*8 d1x, d2x, d1y, d2y, ti, tf
      Real*8 xi(2), xf(2), yi(2), yf(2)
      character output*12
      real*8 g, v0, angle, dt, C, rho, Rp, Mp, yrho, u
      real*8 rad, Cd0, energy, energy0, xc, yc, vxc, vyc
real*8 xfly(5000), yfly(5000), xrange
      integer*4 i, j, key, jmax
      integer iflag, iwork(5), ne
      real*8 y(4), relerr, abserr, work(27)
      parameter (rad=3.1415926/180.0, jmax=5000)
      parameter (relerr=1.0e-9, abserr=0.0)
      common/const/ Cd0, g, yrho
      external d1x, d2x, d1y, d2y, cannon
c*** read initial data from a file
      read 201,  output
      open (unit=7,file=output)
      read 202, key
      read 203, g
      read 203, xi(1)
      read 203, yi(1)
      read 203, v0
      read 203, angle
      read 203, dt
      read 203, C
      read 203, rho
      read 203, Rp
      read 203, Mp
      read 204, yrho
read 203, u
c*** end reading and set initial time to 0.0
      ti = 0.0

c*** end initial data
      xi(2) = v0*cos(angle*rad)
      yi(2) = v0*sin(angle*rad)

c Cd0 is the air resistance coefficient /Mp projectile
      Cd0 = C*rho*3.141592*Rp**2/Mp
```

```
c energy0 is the initial energy of the projectile
c later energy is calculated that is printed as a fraction of energy0
c if there is no frictional forces the energy must be conserved
      energy0= Mp*g*yi(1) + 0.5*Mp*(xi(2)**2+yi(2)**2)

      write(7,210)
      write(7,211) ti, xi(1), yi(1), xi(2), yi(2), energy0
c*** loop over time till the projectile hits the ground
      j=0
c rkf45 initial data and conditions for rkf45 and first call
c       it is very important to call rkf45 for the first time with
c       iflag = 1 (otherwise the code does not run)
      if(key.eq.2) then
   ne = 4
   iflag = 1
   y(1) = xi(1)
   y(2) = yi(1)
   y(3) = xi(2)
   y(4) = yi(2)
      end if

c*** loop till the projectile hits the ground i.e. yf=y1

      do while (yf(1).gt.-0.01)
        j = j+1
        tf = ti + dt

        if(key.eq.0) call euler22m(d1x,d2x,d1y,d2y,ti,tf,xi,xf,yi,yf)
        if(key.eq.1) call  rk4_d22(d1x,d2x,d1y,d2y,ti,tf,xi,xf,yi,yf)
        if(key.eq.2) then
    call rkf45(cannon,ne,y,ti,tf,relerr,abserr,iflag,work,iwork)
            xf(1)=y(1)
    yf(1)=y(2)
    xf(2)=y(3)
    yf(2)=y(4)
    if(iflag.eq.7) iflag = 2
  end if
        energy = Mp*g*yf(1) + 0.5*Mp*(xf(2)**2+yf(2)**2)
        energy = energy/energy0
        xfly(j) = xf(1)/u
  yfly(j) = yf(1)/u
        write(7, 211) tf, xf(1)/u, yf(1)/u, xf(2)/u, yf(2)/u, energy

c* TEST section
c  good test for the code: no air resistance
c  then one may compare with analytic solution
```

```
        xc = 0.0 + v0*cos(angle*rad)*tf
        yc = 0.0 + v0*sin(angle*rad)*tf-0.5*g*(tf)**2
        vxc= v0*cos(angle*rad)
        vyc= v0*sin(angle*rad)-g*(tf)
c remove comment from the next line to print
c      write(7, 211) tf,xf(1)/xc,yf(1)/yc,xf(2)/vxc,yf(2)/vyc,energy

c preparation for the next step
         ti = tf
         do i=1,2
            xi(i) = xf(i)
            yi(i) = yf(i)
         end do
c***  max number of time steps is 2000
if(j.ge.jmax) exit

      end do

c*** calculate max range (using linear interpolation on the last two points)
      xrange = xfly(j-1)
      xrange = xrange+(xfly(j)-xfly(j-1))*yfly(j-1)/(yfly(j-1)-yfly(j))
      write (7, 213) xrange

201   format (a12)
202   format (i5)
203   format (f10.4)
204   format (e10.2)
210   format('    time',7x,'X',11x,'Y',11x,'Vx',10x,'Vy',6x,'energy')
211   format (f8.2, 4f12.3,1pe12.3)
212   format (' Iflag from Rkf45 = ',i2,' -> increase time step')
213   format (/,' Range is =',f12.3)
      pause
      end

      Function d1x(t,x,y)
c---------------------------------------
c function dx/dt
c---------------------------------------
      implicit none
      Real*8 d1x, t, x(2), y(2)
       d1x = x(2)
      return
      end

      Function d1y(t,x,y)
c---------------------------------------
```

```
c function dy/dt
c--------------------------------------
      implicit none
      Real*8 d1y, t, x(2), y(2)
       d1y = y(2)
      return
      end

      Function d2x(t,x,y)
c--------------------------------------
c function d2x/dt2
c--------------------------------------
      implicit none
      Real*8 d2x, t, x(2), y(2), Cd0, g, v, yrho
      common/const/ Cd0, g, yrho
       v = sqrt(x(2)**2+y(2)**2)
       d2x = (-1.0)*(Cd0*exp(-y(1)/yrho))*v*x(2)
      return
      end

      Function d2y(t,x,y)
c--------------------------------------
c function d2y/dt2
c--------------------------------------
      implicit none
      Real*8 d2y, t, x(2), y(2), Cd0, g, v, yrho
      common/const/ Cd0, g, yrho
       v = sqrt(x(2)**2+y(2)**2)
       d2y = (-1.0)*(g + (Cd0*exp(-y(1)/yrho))*v*y(2))
      return
      end

      subroutine cannon(t, y, yp)
c---------------------------------------------
c first and second derivatives for rkf45
c definition of the differential equations
c y(1) = x    yp(1)=vx=y(3)
c y(2) = y    yp(2)=vy=y(4)
c y(3) = vx   yp(3)=d2x/dt2 = - Cd*v*vx
c y(4) = vy   yp(4)=d2y/dt2 = -g - Cd*v*vy
c---------------------------------------------
      implicit none
      Real*8 t, y(4), yp(4), Cd0, g, v, yrho
      common/const/ Cd0, g, yrho
      yp(1)  =  y(3)
      yp(2)  =  y(4)
```

```
c equation of motion
      v = sqrt(y(3)**2+y(4)**2)
  yp(3) = (-1.0)*(Cd0*exp(-y(2)/yrho))*v*y(3)
yp(4) = (-1.0)*(g + (Cd0*exp(-y(2)/yrho))*v*y(4))
      return
      end
```

Y el siguiente es la subrutina del método de Euler;

```
    Subroutine euler22m(d1x,d2x,d1y,d2y,ti,tf,xi,xf,yi,yf)
c==================================================
c euler22m.f: Solution of the second-order 2D ODE
c method:     modified Euler (predictor-corrector)
c written by: Alex Godunov
c last revision: 21 October 2006
c--------------------------------------------------
c input ...
c d1x(t,x,y)- function dx/dt   (supplied by a user)
c d2x(t,x,y)- function d2x/dt2 (supplied by a user)
c d1y(t,x,y)- function dy/dt   (supplied by a user)
c d2y(t,x,y)- function d2y/dt2 (supplied by a user)
c    where x(2) and y(2) (x(1)-position, x(2)-speed, etc.)
c ti  - initial time
c tf  - time for a solution
c xi(2) - initial position and speed for x component
c yi(2) - initial position and speed for y component
c
c output ...
c xf(2) - solutions (x position and speed) at point tf
c yf(2) - solutions (y position and speed) at point tf
c==================================================
      implicit none
      Real*8 d1x, d2x, d1y, d2y, ti, tf
      Real*8 xi(2), xf(2), yi(2), yf(2)
      Real*8 h,t, x1, x2, y1, y2
      Real*8 k1x(2),k2x(2),k3x(2),k4x(2),k1y(2),k2y(2),k3y(2),k4y(2)
      h = tf-ti
      t = ti
c*** Euler
      xf(1) = xi(1) + h*d1x(t,xi,yi)
      xf(2) = xi(2) + h*d2x(t,xi,yi)
      yf(1) = yi(1) + h*d1y(t,xi,yi)
      yf(2) = yi(2) + h*d2y(t,xi,yi)
c*** modified Euler
      xf(1) = xi(1) + (d1x(t,xi,yi)+d1x(t,xf,yf))*0.5*h
      xf(2) = xi(2) + (d2x(t,xi,yi)+d2x(t,xf,yf))*0.5*h
```

```
      yf(1) = yi(1) + (d1y(t,xi,yi)+d1y(t,xf,yf))*0.5*h
      yf(2) = yi(2) + (d2y(t,xi,yi)+d2y(t,xf,yf))*0.5*h
      Return
      End
```

3. Código modificado para la Evaluación No. 1

```
            Program projectile2
!------------------------------------------------------------------
! Realistic projectile motion with air resistance
! method: program may call various ODU solvers
!   key = 0 modified Euler
!   key = 1 Runge-Kutta 4th order
!   key = 2 code Rkf45 (Runge-Kutta 4th-5th order)
! written by: Alex Godunov
! last revision: 08 November 2018
!------------------------------------------------------------------
! input from a file (self explanatory)
!    see file cannon.dat
! output ...
!    to a file named by a user
!------------------------------------------------------------------
      implicit none
      Real*8 d1x, d2x, d1y, d2y, ti, tf
      Real*8 xi(2), xf(2), yi(2), yf(2)
      character output*20, tabla*20
      real*8 g, v0, angle, dt, C, rho, Rp, Mp, yrho, u
      real*8 rad, Cd0, energy, energy0, xc, yc, vxc, vyc
real*8 xfly(5000), yfly(5000), xrange
      integer*4 i, j, key, jmax
      integer iflag, iwork(5), ne
      real*8 y(4), relerr, abserr, work(27)
      parameter (rad=3.1415926/180.0, jmax=5000)
      parameter (relerr=1.0e-9, abserr=0.0)
      common/const/ Cd0, g, yrho
      !external d1x, d2x, d1y, d2y, cannon
      !*** read initial data from a file
      print*, " dame el nombre del archivo"
      read 201,  output
      read 201, tabla
      open (unit=7,file=output)
      read(7, 202) key
      read(7,203) g
      read(7, 203) xi(1)
      read(7, 203) yi(1)
```

```
      read(7, 203) v0
      read(7, 203) angle
      read(7, 203) dt
      read(7,203) C
      read(7,203) rho
      read(7,203) Rp
      read(7, 203) Mp
      read(7,204) yrho
      read(7, 203) u




!*** end reading and set initial time to 0.0
      ti = 0.0

!*** end initial data
      xi(2) = v0*cos(angle*rad)
      yi(2) = v0*sin(angle*rad)

! Cd0 is the air resistance coefficient /Mp projectile
      Cd0 = C*rho*3.141592*Rp**2/Mp

! energy0 is the initial energy of the projectile
! later energy is calculated that is printed as a fraction of energy0
! if there is no frictional forces the energy must be conserved
      energy0= Mp*g*yi(1) + 0.5*Mp*(xi(2)**2+yi(2)**2)


      open(unit=8,file=tabla,status='unknown')

 !write(7,210)
      write(7,211) xi(1), yi(1)
!*** loop over time till the projectile hits the ground
      j=0
! rkf45 initial data and conditions for rkf45 and first call
!       it is very important to call rkf45 for the first time with
!       iflag = 1 (otherwise the code does not run)
      if(key.eq.2) then
   ne = 4
   iflag = 1
   y(1) = xi(1)
   y(2) = yi(1)
   y(3) = xi(2)
   y(4) = yi(2)
```

```
      end if

!*** loop till the projectile hits the ground i.e. yf=y1

      do while (yf(1).gt.-0.01)
        j = j+1
        tf = ti + dt

        if(key.eq.0) call euler22m(ti,tf,xi,xf,yi,yf)
     !   if(key.eq.1) call  rk4_d22(d1x,d2x,d1y,d2y,ti,tf,xi,xf,yi,yf)
        if(key.eq.2) then
!     call rkf45(cannon,ne,y,ti,tf,relerr,abserr,iflag,work,iwork)
          !  xf(1)=y(1)
   ! yf(1)=y(2)
    !xf(2)=y(3)
    !yf(2)=y(4)
    if(iflag.eq.7) iflag = 2
  end if
        energy = Mp*g*yf(1) + 0.5*Mp*(xf(2)**2+yf(2)**2)
        energy = energy/energy0
        xfly(j) = xf(1)/u
  yfly(j) = yf(1)/u
        write(8, 211)  xf(1)/u, yf(1)/u

!* TEST section
!  good test for the code: no air resistance
!  then one may compare with analytic solution
        xc = 0.0 + v0*cos(angle*rad)*tf
        yc = 0.0 + v0*sin(angle*rad)*tf-0.5*g*(tf)**2
        vxc= v0*cos(angle*rad)
        vyc= v0*sin(angle*rad)-g*(tf)
! remove comment from the next line to print
!      write(7, 211) tf,xf(1)/xc,yf(1)/yc,xf(2)/vxc,yf(2)/vyc,energy

! preparation for the next step
         ti = tf
         do i=1,2
            xi(i) = xf(i)
            yi(i) = yf(i)
         end do
!***  max number of time steps is 2000
if(j.ge.jmax) exit

      end do

!*** calculate max range (using linear interpolation on the last two points)
```

```
      xrange = xfly(j-1)
      xrange = xrange+(xfly(j)-xfly(j-1))*yfly(j-1)/(yfly(j-1)-yfly(j))
      !write (7, 213) xrange

201   format (a12)
202   format (i5)
203   format (f10.4)
204   format (e10.2)
210   format(7x,'X',11x,'Y')
211   format (f8.2, 4f12.3,1pe12.3)
212   format (' Iflag from Rkf45 = ',i2,' -> increase time step')
213   format (/,' Range is =',f12.3)
      contains
    end program projectile2


      Function d1x(t,x,y)
!---------------------------------------
! function dx/dt
!---------------------------------------
      implicit none
      Real*8 d1x, t, x(2), y(2)
       d1x = x(2)
      return
    end Function d1x


      Function d1y(t,x,y)
!---------------------------------------
! function dy/dt
!---------------------------------------
      implicit none
      Real*8 d1y, t, x(2), y(2)
       d1y = y(2)
      return
    end Function D1y


      Function d2x(t,x,y)
!---------------------------------------
! function d2x/dt2
!---------------------------------------
      implicit none
      Real*8 d2x, t, x(2), y(2), Cd0, g, v, yrho
      common/const/ Cd0, g, yrho
       v = sqrt(x(2)**2+y(2)**2)
```

```
       d2x = (-1.0)*(Cd0*exp(-y(1)/yrho))*v*x(2)
      return
    end Function d2x


      Function d2y(t,x,y)
!--------------------------------------
! function d2y/dt2
!--------------------------------------
      implicit none
      Real*8 d2y, t, x(2), y(2), Cd0, g, v, yrho
      common/const/ Cd0, g, yrho
       v = sqrt(x(2)**2+y(2)**2)
       d2y = (-1.0)*(g + (Cd0*exp(-y(1)/yrho))*v*y(2))
      return
    end Function d2y


      subroutine cannon(t, y, yp)
!---------------------------------------------
! first and second derivatives for rkf45
! definition of the differential equations
! y(1) = x    yp(1)=vx=y(3)
! y(2) = y    yp(2)=vy=y(4)
! y(3) = vx   yp(3)=d2x/dt2 = - Cd*v*vx
! y(4) = vy   yp(4)=d2y/dt2 = -g - Cd*v*vy
!---------------------------------------------
      implicit none
      Real*8 t, y(4), yp(4), Cd0, g, v, yrho
      common/const/ Cd0, g, yrho
      yp(1)  =  y(3)
      yp(2)  =  y(4)
! equation of motion
      v = sqrt(y(3)**2+y(4)**2)
  yp(3) = (-1.0)*(Cd0*exp(-y(2)/yrho))*v*y(3)
yp(4) = (-1.0)*(g + (Cd0*exp(-y(2)/yrho))*v*y(4))
      return
    end subroutine cannon

      Subroutine euler22m(ti,tf,xi,xf,yi,yf)
!================================================
! euler22m.f: Solution of the second-order 2D ODE
!method:     modified Euler (predictor-corrector)
! written by: Alex Godunov
! last revision: 08 November 2018
!-------------------------------------------------
```

```
! input ...
! d1x(t,x,y)- function dx/dt   (supplied by a user)
! d2x(t,x,y)- function d2x/dt2 (supplied by a user)
! d1y(t,x,y)- function dy/dt   (supplied by a user)
! d2y(t,x,y)- function d2y/dt2 (supplied by a user)
!    where x(2) and y(2) (x(1)-position, x(2)-speed, etc.)
! ti  - initial time
! tf  - time for a solution
! xi(2) - initial position and speed for x component
! yi(2) - initial position and speed for y component
!
! output ...
! xf(2) - solutions (x position and speed) at point tf
! yf(2) - solutions (y position and speed) at point tf
!==================================================
      implicit none
      Real*8 d1x, d2x, d1y, d2y, ti, tf
      Real*8 xi(2), xf(2), yi(2), yf(2)
      Real*8 h,t, x1, x2, y1, y2
      Real*8 k1x(2),k2x(2),k3x(2),k4x(2),k1y(2),k2y(2),k3y(2),k4y(2)
      h = tf-ti
      t = ti
!*** Euler
      xf(1) = xi(1) + h*d1x(t,xi,yi)
      xf(2) = xi(2) + h*d2x(t,xi,yi)
      yf(1) = yi(1) + h*d1y(t,xi,yi)
      yf(2) = yi(2) + h*d2y(t,xi,yi)
!*** modified Euler
      xf(1) = xi(1) + (d1x(t,xi,yi)+d1x(t,xf,yf))*0.5*h
      xf(2) = xi(2) + (d2x(t,xi,yi)+d2x(t,xf,yf))*0.5*h
      yf(1) = yi(1) + (d1y(t,xi,yi)+d1y(t,xf,yf))*0.5*h
      yf(2) = yi(2) + (d2y(t,xi,yi)+d2y(t,xf,yf))*0.5*h
      Return
    End Subroutine Euler22m
```

3.1. Resultados Obtenidos

Como se puede observar en la siguiente tabla, el alcance es mayor en presencia de fricción, de ésto puedo inferir que al no haber fricción o resistencia la gravedad gana en ese caso haciendo que el objeto caiga cada vez más rápido y le gane al impulso inicial.

	Resultados		
	Ángulo	Con Fricción	Sin Fricción
Lanzamiento en km	15°	55.971	48.046
	30°	96.855	58.006
	45°	133.635	55.968
	60°	130.702	44.992
	75°	79.868	25.928

Cuadro 1: Tabla de resultados.

4. Conclusión

Tenemos que si tomamos en cuenta la Fricción el alcance es menor, lo que puedo pensar de ello es que éste caso al no haber resistencia horizontal ni vertical, el movimiento es más libre, sin embargo, al estar acelerado verticalmente por la gravedad, cae más rápido.

Se observó entonces que la diferencia de lanzamientos a diferentes ángulos fue mayor cuando eran 45 grados de inclinación y con friccioón.

4.1. Bibliografía

Department of Physics Old Dominion University
http://ww2.odu.edu/ agodunov/computing/programs/projects/proj02a.f
http://ww2.odu.edu/ agodunov/computing/programs/ode/euler22m.f

4.2. Gráficas

A continuación las gráficas correspondientes a cada lanzamiento.

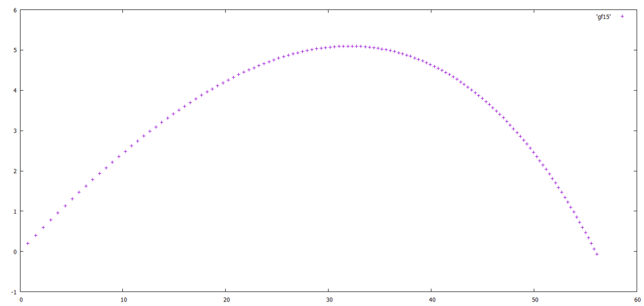

Figura 1: Gráfica con fricción a 15 grados

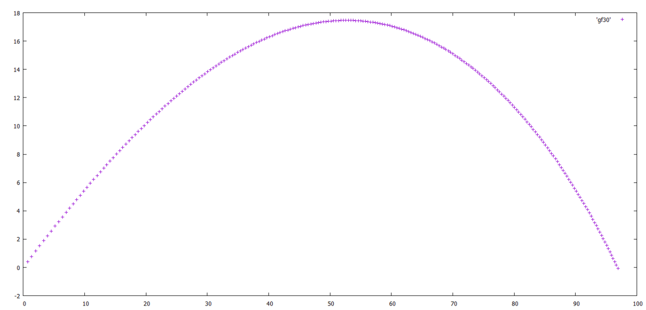

Figura 2: Gráfica con fricción a 30 grados

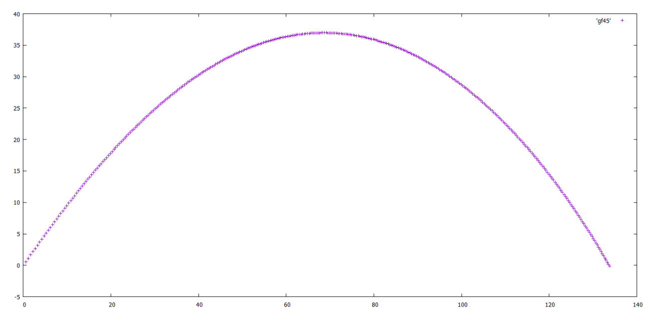

Figura 3: Gráfica con fricción a 45 grados

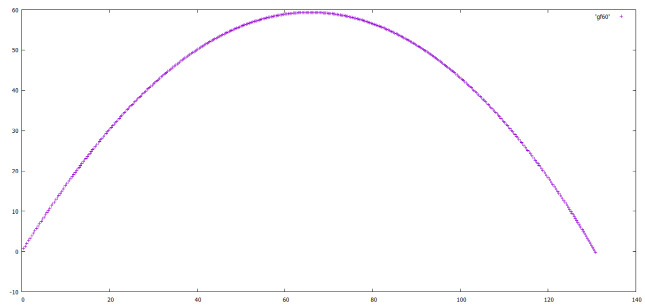

Figura 4: Gráfica con fricción a 60 grados

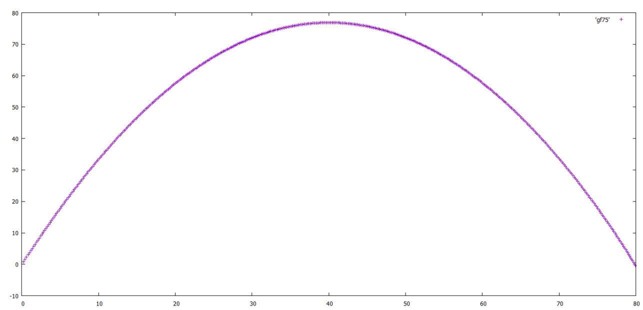

Figura 5: Gráfica con fricción a 75 grados

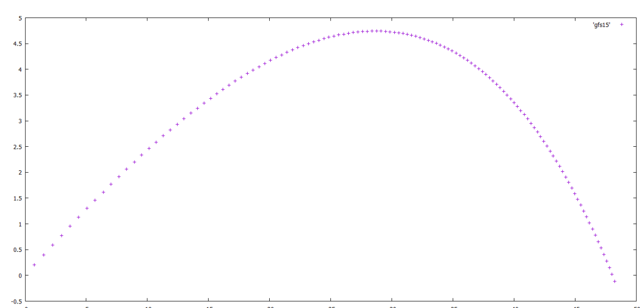

Figura 6: Gráfica sin fricción a 15 grados

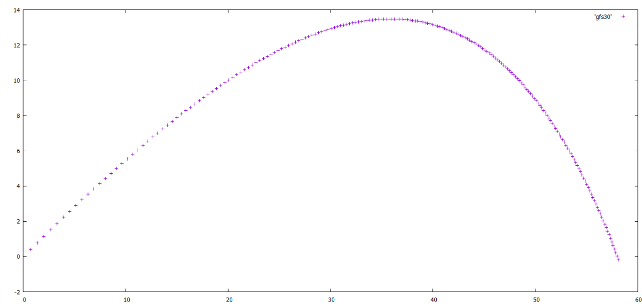

Figura 7: Gráfica sin fricción a 30 grados

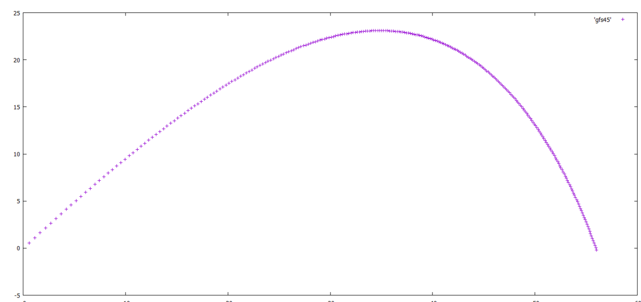

Figura 8: Gráfica sin fricción a 45 grados

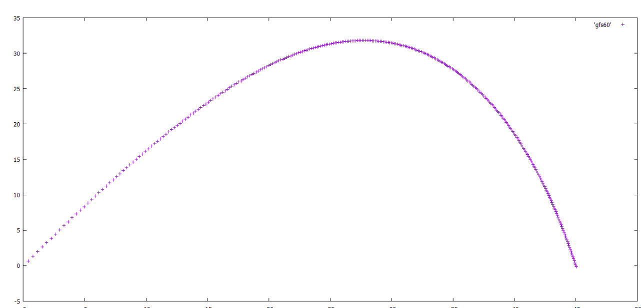

Figura 9: Gráfica sin fricción a 60 grados

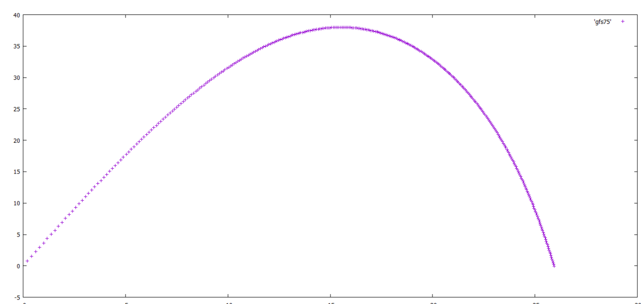

Figura 10: Gráfica sin fricción a 76 grados