Homework 4, Problem 1 (solutions)

Introduction

This problem is a test of several ODE solvers: 2nd order Runge-Kutta, 4th order Runge-Kutta, and adaptive 4th/5th order Runge-Kutta. The test ODE is a second order equation, simple harmonic motion:

d^2x/dt^2 = $-omega^2$ x

Code

With an intial condition of y(1)=1 and y(2)=0, the exact solution to the ODE is y(1,t)=cos(omega*t), where with omega=2pi the period will be exactly 1.

Each part involves three source files: a main program, the derivatives module, and the Runge-Kutta module. The derivatives module was exactly the same for all three versions; there are only small differences in the main programs. They are compiled like this:

```
gfortran -O2 deriv.f90 rk4.f90 main.f90
```

For each version the relative error is printed as a function of the independent variable t. The solution is zero at certain times making the relative error go to infinity. The code avoids printing the relative error if the solution is close to zero.

1. Derivatives module (common for both the 4th order and the 2nd order Runge-Kutta)

   deriv.f90

2. 4th order Runge-Kutta

   main.f90 rk4.f90

3. 2nd order Runge-Kutta

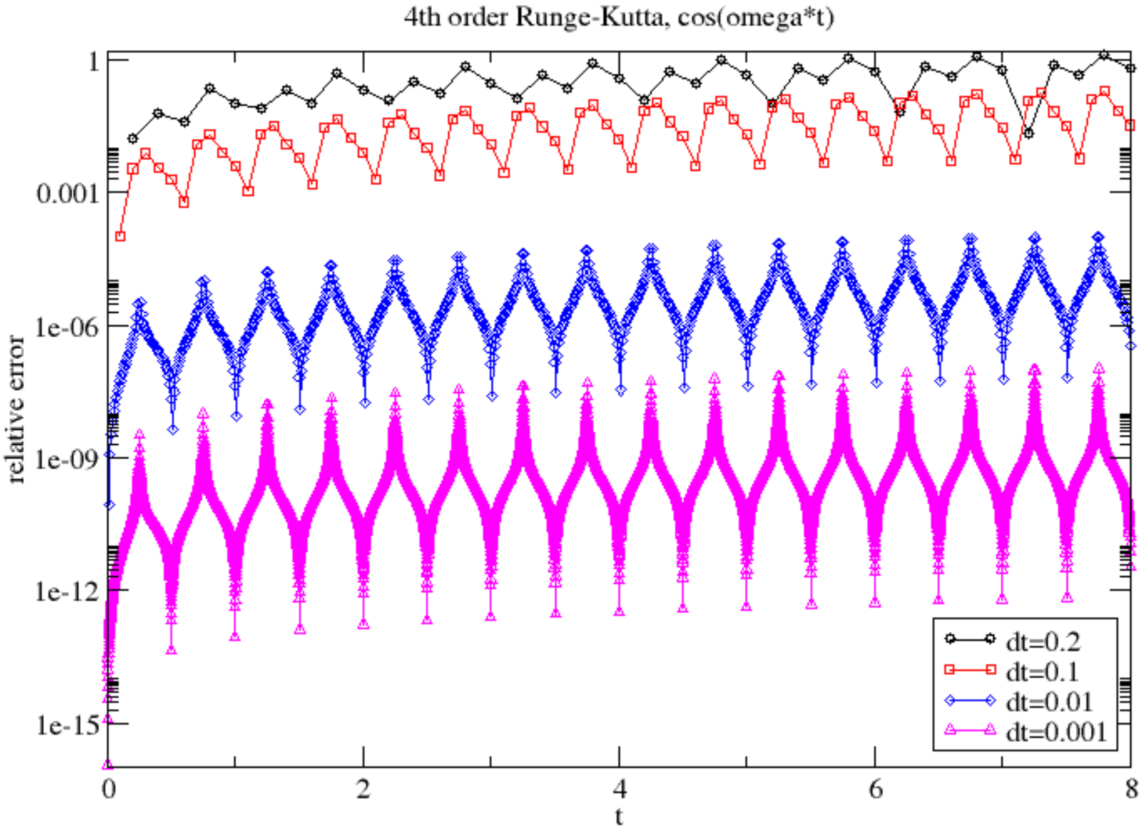
   main-rk2.f90 rk2.f90

4. Adaptive Runge-Kutta: this was a single program, somewhat modified from the textbook's version. This one did not use modules. I made a few changes from the version on mycourses: making the hmin smaller, and also only printing the error if the step is accepted.

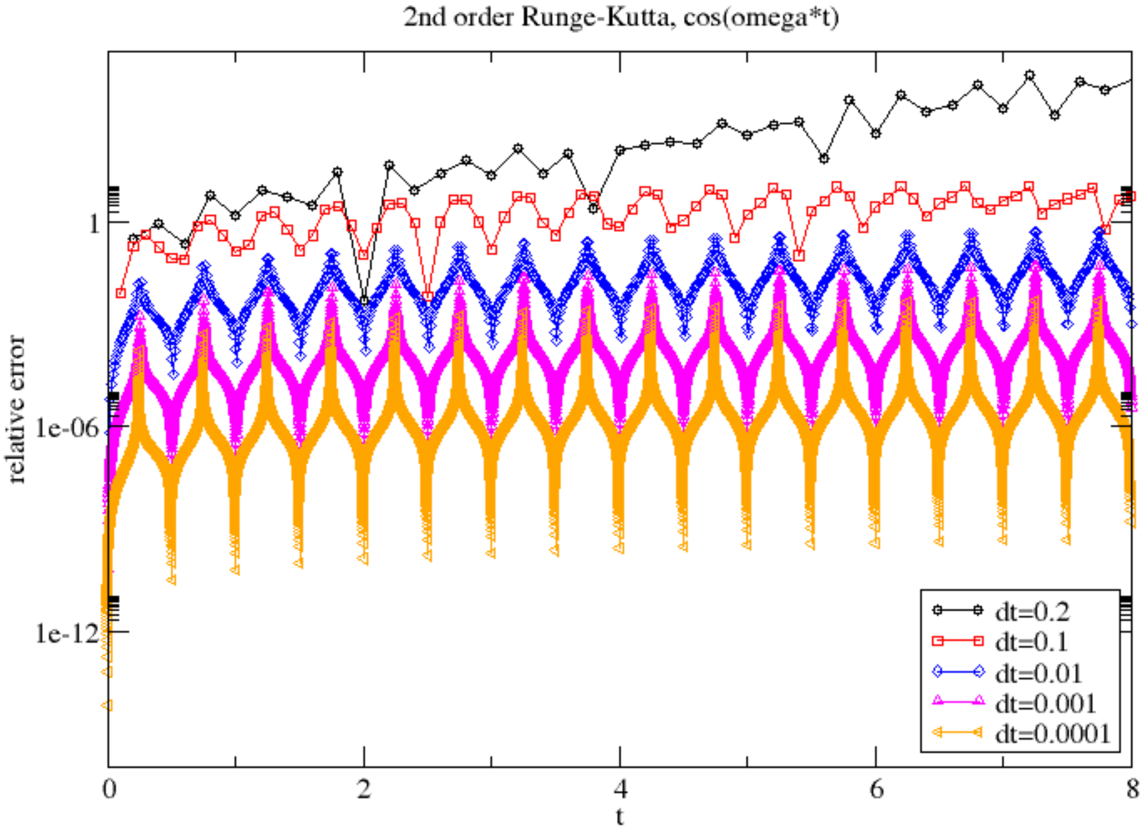
   rk45.f90

Results

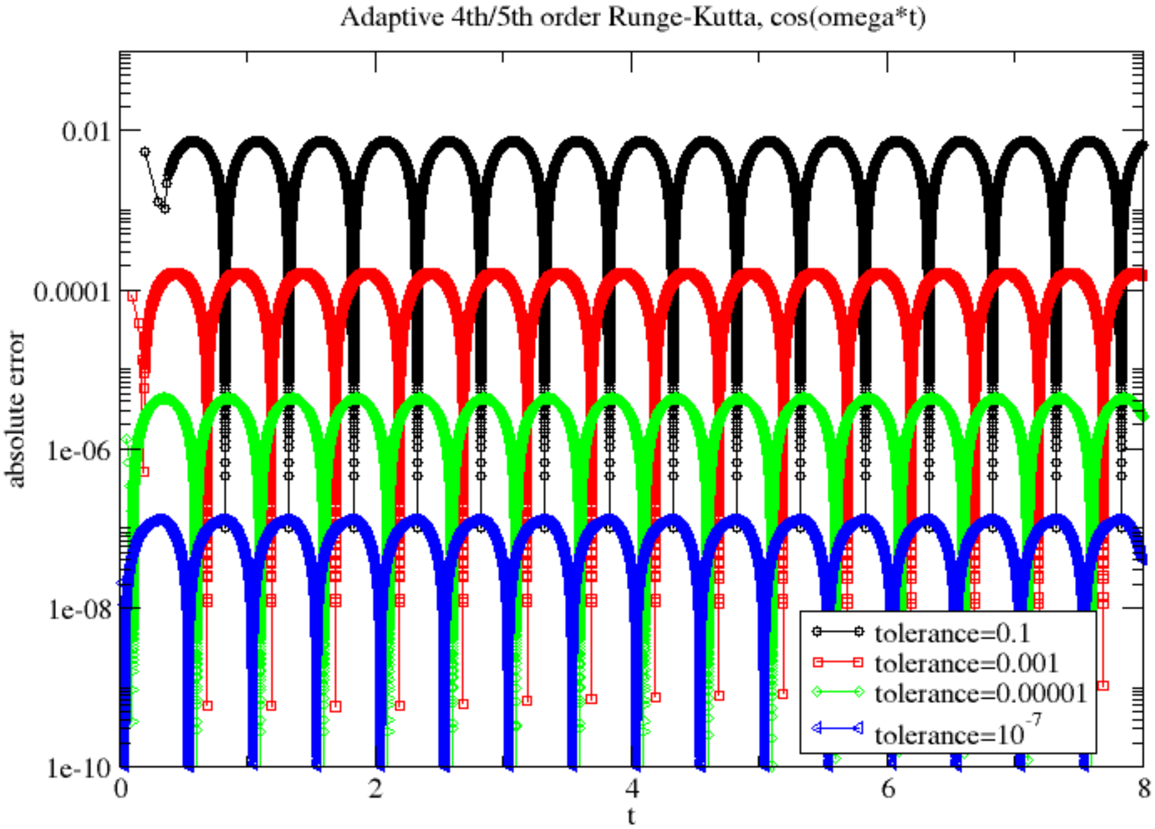
4th order Runge-Kutta



2nd order Runge-Kutta



Adaptive Runge-Kutta. It is clearer to plot the absolute error, since that is the criterion used to adjust the stepsize.



Discussion/Summary

4th order algorithm: notice that when the time step decreases by a factor of 10, the error decreases by a factor of 10^4; this is consistent with 4th-order accuracy. The relative error peaks twice per period when the solution goes through zero. Notice also that the overall error slowly increases with t. This is because errors from each time step accumulate over longer times; the predicted algorithmic error only applies to a single time step. For a single time step of about dt=0.0001, the error is close to machine precision.

2nd order algorithm: very similar error dependence as the 4th order algorithm, except now decreasing dt by 10 only decreases the error by 100.

Adaptive algorithm: for large tolerances this code does not do a very good job of adjusting the stepsize; with a large tolerance, at certain times the stepsize is made much smaller than necessary. For smaller tolerances, the actual maximum error is similar to the requested tolerance.

```
!
! Defines set of ODE's. In this example, N=2, simple harmonic motion
!
module deriv_mod
  implicit none

  private
  public:: deriv,n,pi,omega

  integer,parameter :: n=2
  real(kind=8),parameter :: pi=4.0d0*atan(1.0d0)
  real(kind=8),parameter :: omega=2.0d0*pi

contains

  function deriv(x, y, i)
    real(kind=8)::deriv
    real(kind=8),intent(in) :: x      ! independent variable; not used here
    real(kind=8),intent(in) :: y(:)   ! dependent variable
    integer,intent(in) :: i           ! index of derivative needed

    select case(i)
    case (1)
       deriv=y(2)
    case (2)
       deriv=-omega*omega*y(1)
    end select

  end function deriv

end module deriv_mod
```

```
! rk4.f90:  4th order Runge-Kutta solution for harmonic oscillator
!
! From: "A SURVEY OF COMPUTATIONAL PHYSICS"
! by RH Landau, MJ Paez, and CC BORDEIANU
! Copyright Princeton University Press, Princeton, 2008.
! Electronic Materials copyright: R Landau, Oregon State Univ, 2008;
! MJ Paez, Univ Antioquia, 2008; and CC BORDEIANU, Univ Bucharest, 2008.
! Supported by the US National Science Foundation
!
!
! code cleaned up/modernized/modularized by RTC 09/2009
!
module rk4_mod
  use deriv_mod
  implicit none

  private
  public :: rk4

contains

! rk4: 4th order Runge-Kutta method
!
! x : independent variable. The subroutine does not modify this.
! xstep : stepsize.
! y : dependent variables
!
  subroutine rk4(x, xstep, y)
    real(kind=8),intent(in) :: x, xstep
    real(kind=8),intent(inout),dimension(:) :: y

    real(kind=8),dimension(size(y)) :: k1, k2, k3, k4, t1, t2, t3
    real(kind=8) :: h
    integer :: i

    h=xstep/2.0_8
    do i = 1,size(y)
       k1(i) = xstep * deriv(x, y, i)
       t1(i) = y(i) + 0.5_8*k1(i)
    enddo
    do i = 1,size(y)
       k2(i) = xstep * deriv(x+h, t1, i)
       t2(i) = y(i) + 0.5_8*k2(i)
    enddo
    do i = 1,size(y)
       k3(i) = xstep * deriv(x+h, t2, i)
       t3(i) = y(i) + k3(i)
    enddo
    do i = 1,size(y)
       k4(i) = xstep * deriv(x+xstep, t3, i)
       y(i) = y(i) + (k1(i) + (2.0_8*(k2(i) + k3(i))) + k4(i))/6.0_8
    enddo

    return
  end subroutine rk4

end module rk4_mod
```

```
!
! solves ODE using 4th order Runge-Kutta
!
program rktest
  use deriv_mod
  use rk4_mod
  implicit none

  real(kind=8),dimension(n) :: y
  real(kind=8) :: t,dt,period
  integer :: i

  y(1)=1.0d0         ! initial conditions
  y(2)=0.0d0
  t=0.0d0            ! initialize independent variable
  period=2.0d0*pi/omega
  dt=period/1000000

  print *,’dt=’,dt
  open(31,file=’rk.dat’,status=’unknown’)

  do while (t<8*period)
     call rk4(t,dt,y)
     t=t+dt
     ! avoid printing relative error if y(1) is
     ! nearly zero
     if (abs(cos(omega*t))>1.0d-6) then
        write(31,*) t,abs((y(1)-cos(omega*t))/cos(omega*t))
     endif
  enddo
  close(31)
  print *,’results in rk.dat’

end program rktest
```

```
!
! solves ODE using 4th order Runge-Kutta
!
program rktest
  use deriv_mod
  use rk2_mod
  implicit none

  real(kind=8),dimension(n) :: y
  real(kind=8) :: t,dt,period
  integer :: i

  y(1)=1.0d0         ! initial conditions
  y(2)=0.0d0
  t=0.0d0            ! initialize independent variable
  period=2.0d0*pi/omega
  dt=period/1000000

  print *,’dt=’,dt
  open(31,file=’rk.dat’,status=’unknown’)

  do while (t<8*period)
     call rk2(t,dt,y)
     t=t+dt
     ! avoid printing relative error if y(1) is
     ! nearly zero
     if (abs(cos(omega*t))>1.0d-6) then
        write(31,*) t,abs((y(1)-cos(omega*t))/cos(omega*t))
     endif
  enddo
  close(31)
  print *,’results in rk.dat’

end program rktest
```

```
! 2nd order Runge-Kutta
!
module rk2_mod
  use deriv_mod
  implicit none

  private
  public :: rk2

contains

! rk2: 2nd order Runge-Kutta method
!
! x : independent variable. The subroutine does not modify this.
! xstep : stepsize.
! y : dependent variables
! n : number of equations
!
  subroutine rk2(x, xstep, y)
    real(kind=8),intent(in) :: x, xstep
    real(kind=8),intent(inout) :: y(:)

    real(kind=8),dimension(size(y)) :: k2,k1,t1
    integer :: i

    do i = 1,size(y)
       k1(i) = xstep * deriv(x, y, i)
       t1(i) = y(i) + 0.5_8*k1(i)
    enddo
    do i = 1,size(y)
       k2(i) = xstep * deriv(x+xstep*0.5d0, t1, i)
       y(i) = y(i) + k2(i)
    enddo
    return
  end subroutine rk2

end module rk2_mod
```

```
!  From: "A SURVEY OF COMPUTATIONAL PHYSICS"
!  by RH Landau, MJ Paez, and CC BORDEIANU
!  Copyright Princeton University Press, Princeton, 2008.
!  Electronic Materials copyright: R Landau, Oregon State Univ, 2008;
!  MJ Paez, Univ Antioquia, 2008; and CC BORDEIANU, Univ Bucharest, 2008.
!  Support by National Science Foundation
!
!   rk45.f90: ODE solver via variable step size rk, Tol = error
!
! cleaned up and rewritten in modern F90 style  by RTC 09/2013
program rk45
  implicit none

  real(kind=8) :: h, t, s, s1, hmin, hmax
  real(kind=8),parameter :: Tol = 1.0e-7
  real(kind=8),parameter :: Tmin = 0.0d0
  real(kind=8),parameter :: Tmax = 8.0d0
  real(kind=8),dimension(2) :: w, y, FReturn, ydumb, k1, k2, k3, k4, k5, k6, err
  real(kind=8),parameter :: pi=4.0d0*atan(1.0d0)
  integer :: i
  integer,parameter :: Ntimes = 10

  ! output to file
  Open(10, FILE = 'rk45.dat', Status = 'Unknown')

  ! initialize
  y(1) = 1.0d0 ; y(2) = 0.0d0

  ! tentative number of steps
  h = (Tmax - Tmin) / Ntimes

  ! minimum and maximum step size
  hmin = 1.0d-5
  hmax = 0.5d0

  t = Tmin

  do while (t < Tmax)
     If ( (t + h)  >  Tmax ) then
        h = Tmax - t    ! the last step
     endif
     ! evaluate both RHSs and Return in F
     call f(t, y, FReturn)
     do i = 1, 2
        k1(i) = h*FReturn(i)
        ydumb(i) = y(i) + k1(i)/4
     end do
     call f(t + h/4, ydumb, FReturn)
     do i = 1, 2
        k2(i) = h*FReturn(i)
        ydumb(i) = y(i) + 3.0d0*k1(i)/32 + 9.0d0*k2(i)/32
     end do
     call f(t + 3.0d0*h/8, ydumb, FReturn)
     do i = 1, 2
        k3(i) = h*FReturn(i)
        ydumb(i) = y(i) + 1932.0d0*k1(i)/2197.0d0 - 7200.0d0*k2(i)/2197.0d0 &
             + 7296.0d0*k3(i)/2197.0d0
     end do
     call f(t + 12*h/13.0d0, ydumb, FReturn)
     do i = 1, 2
        k4(i) = h*FReturn(i)
        ydumb(i) = y(i) + 439.0d0*k1(i)/216.0d0-8.0d0*k2(i) &
             + 3680.0d0*k3(i)/513.0d0-845.0d0*k4(i)/4104.0d0
     end do
     call f(t + h, ydumb, FReturn)
     do i = 1, 2
        k5(i) = h*FReturn(i)
        ydumb(i) = y(i) - 8*k1(i)/27.0d0 + 2*k2(i) - 3544.0d0*k3(i)/2565.0d0 &
             + 1859.0d0*k4(i)/4104.0d0 - 11.0d0*k5(i)/40.0d0
     end do
     call f(t + h/2, ydumb, FReturn)
     do i = 1, 2
        k6(i) = h*FReturn(i)
        err(i) = abs( k1(i)/360.0d0 - 128*k3(i)/4275.0d0 - 2197.0d0*k4(i)/75240.
```

```
0d0 &
             + k5(i)/50.0d0 + 2*k6(i)/55.0d0 )
     end do
     If ((err(1) < Tol).or.(err(2) < Tol).or.(h <= 2*hmin))     then
        ! accept approximation
        do i = 1, 2
           y(i) = y(i) + 25.0d0*k1(i)/216.0d0 + 1408.0d0*k3(i)/2565.0d0 &
                + 2197.0d0*k4(i)/4104.0d0 - k5(i)/5.0d0
        end do
        t = t + h
        if (abs(cos(2.0d0*pi*t))>1.0d-6) then
           write(10, *) t, abs(y(1)-cos(2.0d0*pi*t))

        endif
     endif
     If (( err(1) == 0).or.(err(2) == 0)) then
        s = 0.0d0! trap division by 0
     else
        s = 0.84d0*Tol*h/err(1)**0.25d0                     ! step size scalar
     endif
     If ( (s  <  0.75d0).and. (h  >  2*hmin) )then
        h =     h/2.0d0 ! reduce step
     else If ( (s  >  1.5d0).and.(2* h  <  hmax) )then
        h = h*2.0d0     ! increase step
     endif
  end do
  close(10)
  stop' Data stored in rk45.dat'

contains

  ! PLACE YOUR FUNCTION HERE
  subroutine f(t,y,FReturn)
    real(kind=8) :: t, y(2), FReturn(2)

    FReturn(1) = y(2) ! RHS of first equation
    FReturn(2) = - 4.0d0*pi*pi*y(1) ! RHS of 2nd equation

  end subroutine f

end program rk45
```

Homework 4, Problem 2 (solutions)

Introduction

In this problem we solve for the trajectory of a classical particle moving in two dimensions. The particle is given an initial velocity in the direction of a potential $V(x,y)=x^2y^2e^{-(x^2+y^2)}$ which is nonzero (positive) in the region close to the origin. The particle then scatters off the potential with a scattering angle theta from its initial direction of travel.

Because the potential has four different maxima at (x,y)=(-1,-1), (-1,1), (1,-1,), and (1,1), multiple scattering events can occur. These can lead to a chaotic dependence of the scattering angle with respect to the initial impact parameter of the particle.

Parameters: mass m=0.5. There are 4 independent variables in the ODE corresponding to x,dx/dt,y, and dy/dt. Initial conditions are:

- y(1)=x=b (the impact parameter, varied from -1 to 0)
- y(2)=y=-y0
- y(3)=dx/dt=0
- y(4)=dy/dt=0.5 (initial velocity)

Code

The code is broken into 3 modules:

1. deriv.f90 Contains ODE derivatives and parameters
2. rk4.f90 4th order Runge-Kutta, unchanged from previous problem
3. main-echeck.f90 Checks energy conservation, prints trajectory

I also made a separate main program to vary the impact parameter and print the scattering angle. I check two conditions to verify that the particle has completed scattering: t>10 and the ratio of PE/KE < 1.0e-10.

1. main.f90 main program varying impact parameter

Results

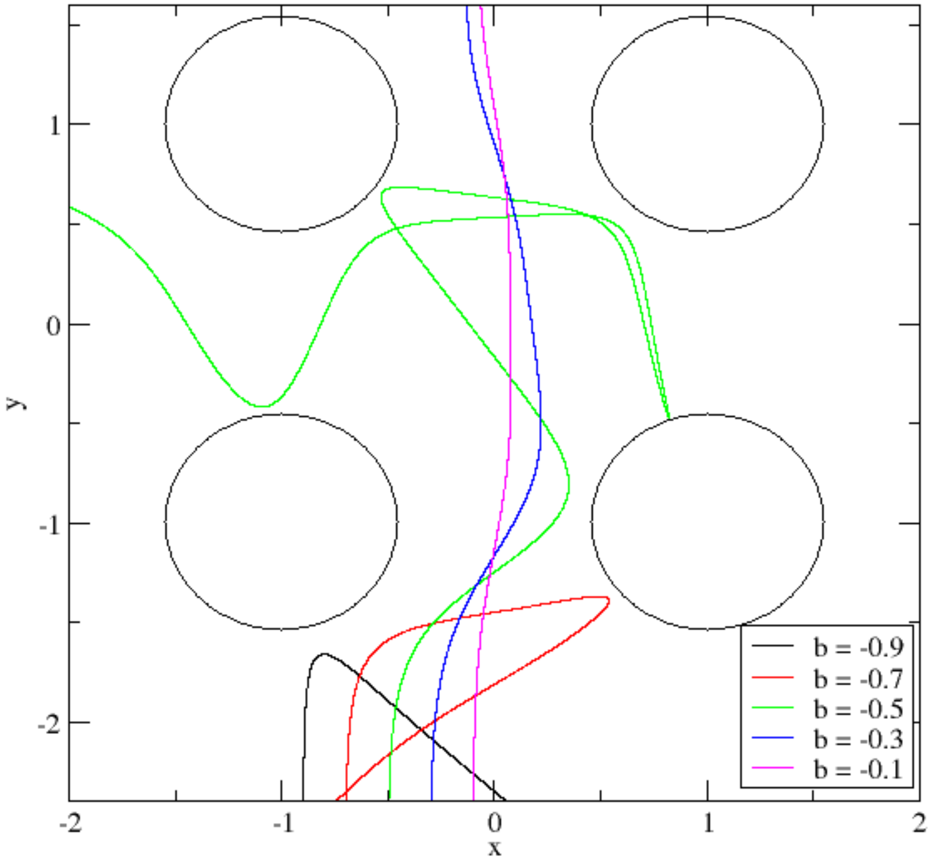
Checks of code

1. First I checked for conservation of energy. This shows that the integration of the ODE is (probably) correct. With a timstep of dt=0.001, the relative error in energy conservation for a total integration time of 50 was of order 10^{-15}. Probably a larger time step could be used.

2. To calculate the scattering angle correctly, the particle need to start far enough away from the potential to be moving essentially as a free particle. This can be checked by calculating the ratio PE/KE; I found for an initial y value of -6, the ratio was about $4.0e^{-14}$

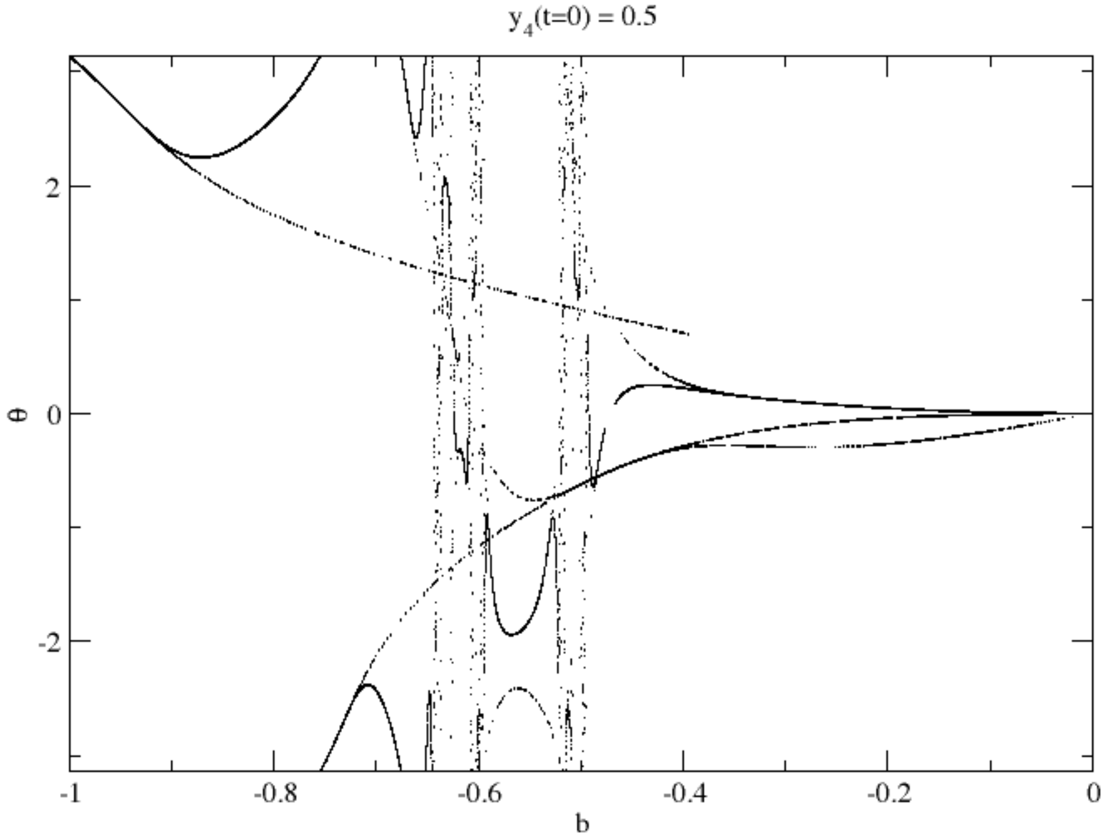
Trajectories

Here is a picture of the trajectories for a range of b. I also drew circles representing the location of the peaks in the potential. For b=-1 the particle scatters back at theta=pi; for b=0 it will travel through the middle with no deviation (theta=0). For intermediate b the particle may scatter off all 4 of the potential peaks before exiting.

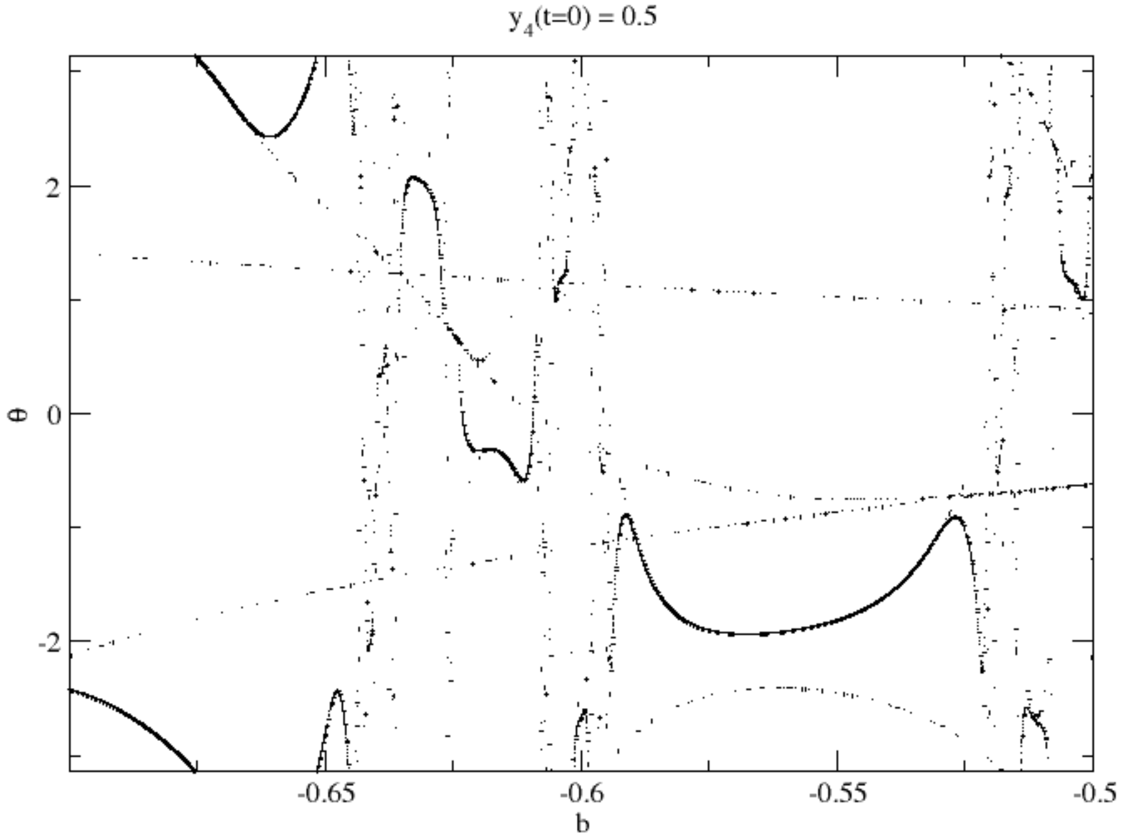


Scattering angle versus b

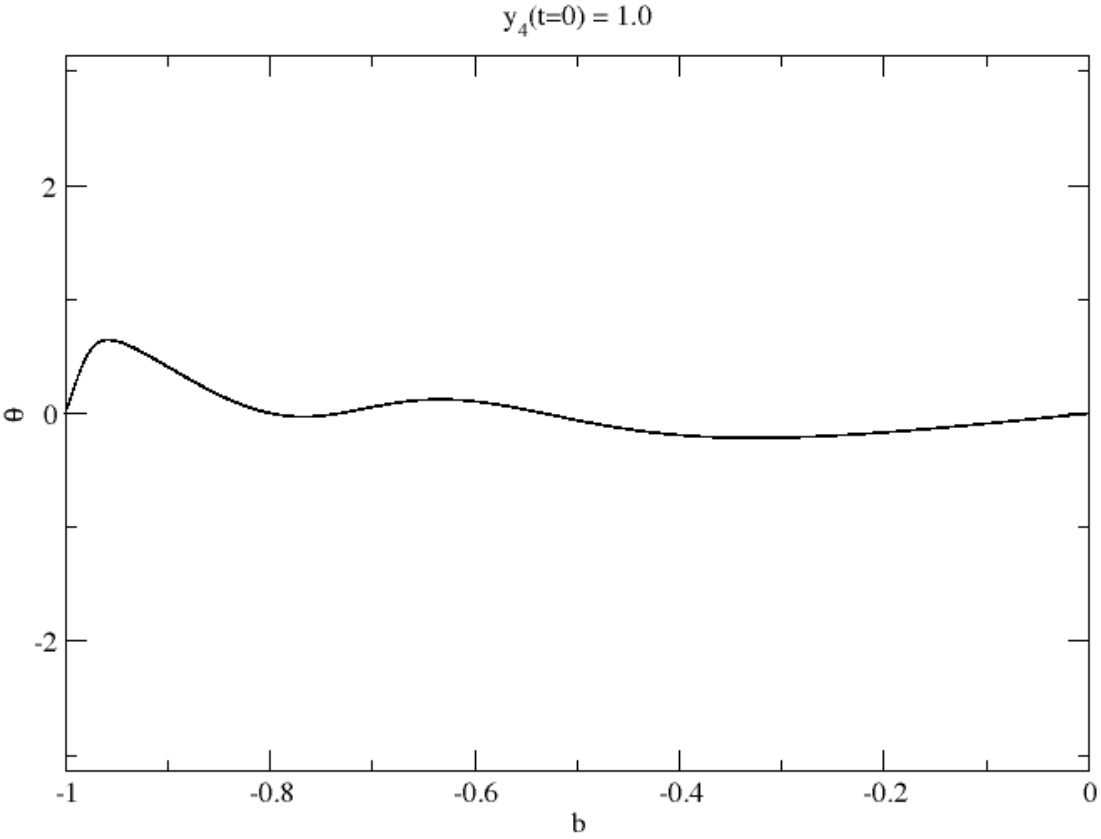
initial velocity = 0.5



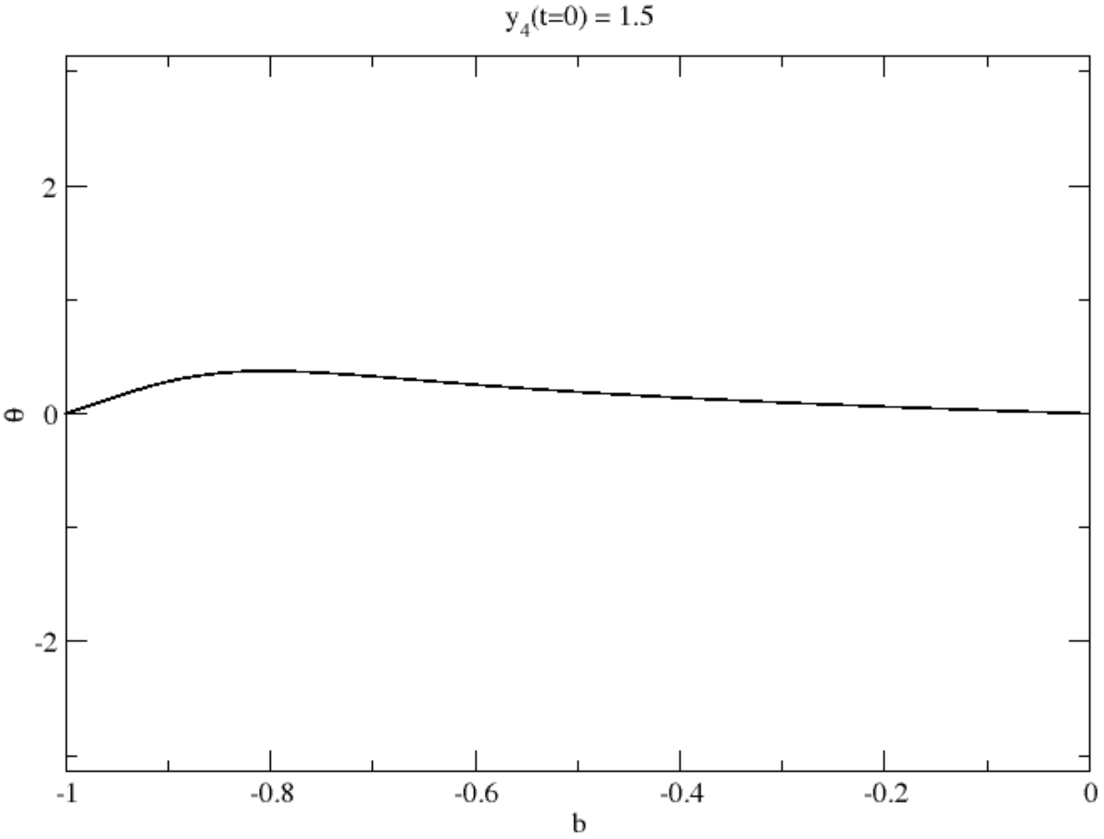
zoomed in plot for intermediate b:



initial velocity = 1.0



initial velocity = 1.5



Discussion/Summary

The scattering angle starts at pi for b=-1 because the particle bounces straight backwards. For b=0 the particle travels straight through without deviation and the scattering angle is zero. In between very complicated behavior takes place. One thing I found interesting was that for certain ranges of impact parameter (such as b between about -0.9 and -0.8), there are two or more solutions for the scattering angle. In these regions, changing the b by a tiny amount changed the scattering angle from one to the other angles.

If the initial velocity is increased, the kinetic energy of the particle will be larger that the peaks of the potential. In this case, the particle travels straight through for b=-1 instead of scattering straight backwards.

```
!
! 2D scattering derivatives
!
module deriv_mod
  implicit none

  private
  public:: deriv,n,m
  integer,parameter :: n=4
  real(kind=8),parameter :: m=0.5  ! particle mass

contains

  function deriv(x, y, i)
    real(kind=8)::deriv
    real(kind=8),intent(in) :: x
    real(kind=8),intent(in) :: y(:)
    integer,intent(in) :: i

    select case(i)
    case (1)
       deriv=y(3)
    case (2)
       deriv=y(4)
    case (3)
       deriv=-2.0d0/m*y(2)*y(2)*y(1)*(1.0d0-y(1)*y(1))*exp(-(y(1)*y(1)+y(2)*y(2)
))
    case (4)
       deriv=-2.0d0/m*y(1)*y(1)*y(2)*(1.0d0-y(2)*y(2))*exp(-(y(1)*y(1)+y(2)*y(2)
))
    end select

    return
  end function deriv

end module deriv_mod
```

```
!
! Classical Scattering of particle from potential in two dimensions
!
!  RT Clay 10/2015
program scatter
  use deriv_mod
  use rk4_mod
  implicit none

  real(kind=8),parameter :: maxt=50.0d0
  real(kind=8),parameter :: pi=atan(1.0d0)*4.0d0
   real(kind=8) :: dt,t,b, k,p,theta, r, e0,e
  real(kind=8),dimension(n) :: y
  integer :: i

  ! initial conditions
  b=-0.1d0
  t=0.0d0
  dt=0.001d0
  y(1)=b
  y(2)=-6.0d0
  y(3)=0.0d0
  y(4)=0.5d0
  r=1.0d0

  ! initial energy
  e0=ke(y)+pe(y)
  do while (t<maxt)
     e=ke(y)+pe(y)
     !     print *,t,pe(y)/ke(y),abs((e-e0)/e0)  ! for PE/KE, relative energy er
ror
     print *,y(1),y(2)   ! for x(t), y(t) trajectory
     call rk4(t,dt,y,n)
     t=t+dt
  enddo

contains

  function ke(y)
    real(kind=8),dimension(n) :: y
    real(kind=8) :: ke
    ke=0.5*m*(y(3)*y(3)+y(4)*y(4))
  end function ke


  function pe(y)
    real(kind=8),dimension(n) :: y
    real(kind=8) :: pe
    pe=y(1)*y(1)*y(2)*y(2)*exp(-(y(1)*y(1)+y(2)*y(2)))
  end function pe

end program scatter
```

```
!
! Classical Scattering of particle from potential in two dimensions
!
!  RT Clay 10/2015
program scatter
  use deriv_mod
  use rk4_mod
  implicit none

  real(kind=8),parameter :: maxt=50.0d0
  real(kind=8),parameter :: pi=atan(1.0d0)*4.0d0
  real(kind=8),parameter :: db=0.00005d0
  real(kind=8) :: dt,t,b, k,p,theta, r, e0
  real(kind=8),dimension(n) :: y
  integer :: i

  ! loop over impact parameters
  b=-1.0d0
  do while (b<0.001d0)
     ! initial conditions
     t=0.0d0
     dt=0.001d0
     y(1)=b
     y(2)=-6.0d0
     y(3)=0.0d0
     y(4)=0.5d0
     r=1.0d0
     do while (t<maxt)
        k=ke(y)
        p=pe(y)
        r=p/k
        if (t>10.0d0.and.r<1.0d-10) exit
        call rk4(t,dt,y,n)
        t=t+dt
     enddo
     theta=atan2(y(3),y(4))
     print *,b,theta,t
     b=b+db
  end do

contains

  function ke(y)
    real(kind=8),dimension(n) :: y
    real(kind=8) :: ke

    ke=0.5*m*(y(3)*y(3)+y(4)*y(4))
  end function ke


  function pe(y)
    real(kind=8),dimension(n) :: y
    real(kind=8) :: pe

    pe=y(1)*y(1)*y(2)*y(2)*exp(-(y(1)*y(1)+y(2)*y(2)))
  end function pe


end program scatter
```