

Exercises Lecture X

Chaos and determinism

1. Logistic map

The program `map.f90` calculates the values x_n of the terms of a sequence obtained from the starting point x_0 by applying iteratively the logistic application:

$$x_{n+1} = 4rx_n(1 - x_n)$$

with a chosen parameter r up to a given index $n = n_{max}$.

- (a) Calculate the terms of the series up to $n = 15$ with the parameter $r_1 = 1/3$ and starting from $x_0 = 1/4$, and those for the parameter $r_2 = 0.5$ starting from $x_0 = 1/2$. Verify that the two values are *stable equilibrium points* for the corresponding parameters r .
- (b) For $r = 0.875$, calculate up to $n = 50$ the terms of the sequence starting from $x_0 = 0.3$, then from $x_0 = 0.6$ and from $x_0 = 0.9$. Determine the period of the trajectory and calculate the fixed points with 5 significant digits.
- (c) Calculate the Lyapunov exponent for some values of r .

2. Regular and chaotic billiards

Consider a two-dimensional planar geometry in which a particle moves with constant velocity along straight line orbits until it elastically reflects off the boundary. This straight line motion occurs in various “billiard” systems. A simple example of such a system is a particle moving with fixed speed within a circle. Suppose that we divide the circle into two equal parts and connect them by straight lines of length L . This geometry is called a *stadium billiard*. It has a similar geometry to that known as the *Sinai billiard model*. The program `billiard.f90` produces the trajectory for a particle moving in a circular or stadium billiard according to the choice of L . Use the radius of the circular parts as unit length.

The algorithm for determining the path of the particle is as follows:

- (a) Begin with an initial position (x_0, y_0) and momentum (p_{x0}, p_{y0}) of the particle such that $|p_0| = 1$.
- (b) Determine which of the four sides the particle will hit. The possibilities are:
 - (1) top line segment
 - (2) bottom line segment
 - (3) right semicircle
 - (4) left semicircle.
- (c) Determine the next position of the particle from the intersection of the straight line defined by the current position and momentum, and the equation for the segment where the next reflection occurs.

- (d) Determine the new momentum, (p'_x, p'_y) , of the particle after reflection such that the angle of incidence equals the angle of reflection. For reflection off the line segments we have $(p'_x, p'_y) = (P_x, -p_y)$. For reflection off a circle we have:

$$p'_x = (y^2 - (x - x_c)^2)p_x - 2(x - x_c)yp_y$$

$$p'_y = -2(x - x_c)yp_x + ((x - x_c)^2 - y^2)p_y$$

where $(x_c, 0)$ is the center of the circle.

- (e) Repeat steps (b)-(d).

3. Observe qualitatively that the dynamics is chaotic if $L \neq 0$.
4. The divergence of the trajectories even with slightly different initial conditions can be described by the Lyapunov exponent, defined by the relation:

$$|\Delta s| = |\Delta s_0|e^{\lambda n}$$

where n is the number of reflections and Δs is the difference of the two phase space trajectories defined by

$$\Delta s = \sqrt{|\mathbf{r}_1 - \mathbf{r}_2|^2 + |\mathbf{p}_1 - \mathbf{p}_2|^2}.$$

Estimate the largest Lyapunov exponent for $L=1$. One way to do it, is to consider two particles starting with almost identical positions and/or momenta (varying by say 10^{-5}) and make a *semilog* plot of Δs versus n . Why does the exponential growth in Δs stop for sufficiently large n ? Repeat your calculation for different initial conditions and average your values of Δs before plotting. Repeat the calculation for $L = 0.1, 0.5$, and 2.0 and determine if your results depend on L .

5. Another test for the existence of chaos is the reversibility of the motion. Do this simulation for $L = 1$ and $L = 0$. Reverse the momentum after the particle has made n reflections, and let the drawing color equal the background color so that the path can be erased. What limitation does roundoff error place on your results? Hint: use single and double precision.


```

!      real,dimension(n+1)::x,y,px,py

if(px(i)/=0.0) then
! case px(i)/=0
m=py(i)/px(i)
q =y(i)-m*x(i) ! line through (x,y) and with
! the direction of the momentum
if(m>0.0) then
! subcase m>0
if(py(i)>0.0) then
if(((1.0-q)/m)>L/2) then
ro=3
else
if(((1.0-q)/m)>(-L/2)) then
ro=1
else
ro=4
end if
end if
else
if(((1.0-q)/m)<(-L/2)) then
ro=4
else
if(((1.0-q)/m)<L/2) then
ro=2
else
ro=3
end if
end if
end if
end if

if(m<0.0)then
! subcase m<0
if(py(i)>0.0) then
if(((1-q)/m)<(-L/2)) then
ro=4
else
if(((1.0-q)/m)<L/2) then
ro=1
else
ro=3
end if
end if
else
if(((1.0-q)/m)>L/2) then

```

```

        ro=3
    else
        if(((−1.0−q)/m)>(−L/2)) then
            ro=2
        else
            ro=4
        end if
    end if
end if

! subcase m=0 (within the case px(i)≠0)
if(m==0.0) then
    if(px(i)>0.0) then
        ro=3
    else
        ro=4
    end if
end if

else

    ! case px(i)=0
    if(py(i)>0.0) then
        if(abs(x(i))<L/2) then
            ro=1
        else
            if(x(i)>=L/2)ro=3
            if(x(i)<=−L/2))ro=4
        end if
    else
        if(abs(x(i))<L/2) then
            ro=2
        else
            if(x(i)>=L/2)ro=3
            if(x(i)<=−L/2))ro=4
        end if
    end if

end if

return
end subroutine hit

subroutine reflect(i)
    ! calc. the abscissa of the reflection point, x(i+1), here called xr

```

```

integer,intent(in) :: i

if(px(i)==0.0) then
  xr(ro)=x(i)
else
  if((ro==1.0).or.(ro==2.0)) then
    xr(ro)=((-1)**(ro+1) - q)/m
  end if
  if((ro==3.0).or.(ro==4.0)) then
    b=(1.0+m**2)*(q**2+(L**2)/4.0 -1.0)
    a=((m*q+((-1)**ro)*L/2)**2)-b
    xr1=(-(m*q+((-1)**ro)*L/2)+sqrt(a))/(1+m**2)
    xr2=(-(m*q+((-1)**ro)*L/2)-sqrt(a))/(1+m**2)
    xrmin=min(xr1,xr2)
    xrmax=max(xr1,xr2)
  end if
  if(ro==3.0) then
    if((m*py(i))<0.0) then
      xr(ro)=xrmin
    else
      xr(ro)=xrmax
    end if
  end if
  if(ro==4.0) then
    if((m*py(i))<=0.0) then
      xr(ro)=xrmin
    else
      xr(ro)=xrmax
    end if
  end if
end if

return
end subroutine reflect

subroutine move(i)
!   determine the position of reflection and momenta after reflection
integer,intent(in) :: i

x(i+1)=xr(ro)
if((ro==1.0).or.(ro==2.0)) then
  px(i+1)=px(i)
  py(i+1)= - py(i)
  y(i+1)= (-1)**(ro+1.0)
end if

```

```

        if((ro==3.0).or.(ro==4.0)) then
            if(px(i)/=0.0) then
                y(i+1)= m*x(i+1)+q
            else
                y(i+1)=sign(1.,py(i))*sqrt(1-(x(i+1)+((-1)**ro)*L/2)**2)
            end if
            a=(y(i+1)**2-(x(i+1)+((-1)**ro)*L/2)**2)*px(i)
            b=2*(x(i+1)+((-1)**ro)*L/2)*y(i+1)*py(i)
            px(i+1)=a-b
            a=-2*(x(i+1)+((-1)**ro)*L/2)*y(i+1)*px(i)
            b=((x(i+1)+((-1)**ro)*L/2)**2-y(i+1)**2)*py(i)
            py(i+1)=a+b
        end if

        return
    end subroutine move

    subroutine output(i)
        !      write positions and momenta on a file
        integer,intent(in) :: i
        write(unit=3,fmt=*)i,x(i),y(i),px(i),py(i)
        return
    end subroutine output

end module bill

program billiard
    use bill
    integer::j
    call start()
    open(unit=3,file="dati",status="replace",action="write")

    do j=1,n-1
        call hit(j)
        call reflect(j)
        call move(j)
        call output(j)
    end do

    deallocate (x)
    deallocate (y)
    deallocate (px)
    deallocate (py)
    close(3)
end program billiard

```