```
Ejercicios Unidad 5
-------------------

Dia 1
-----------------------------

La ecuacion del movimiento de Newton es
   m*d2x/dt2 = f(x)
Esta ecuacion diferencial de segundo orden para x(t) puede
reescribirse como una ecuacion de primer orden con dos
componentes, x(t) y v(t):
   dx/dt = v
   dv/dt = a = f(x)/m
El metodo de Euler consiste en iterar
   x(t+dt) =~ x(t) + v(t)*dt   (donde =~ significa aqui
                                'aprox. igual')
   v(t+dt) =~ v(t) + a(t)*dt,  con a(t)=f(x(t))/m
comenzando con x(0)=x0 y v(0)=v0.
Aunque muy sencillo, este metodo es demasiado inexacto
en la practica.

El metodo de Verlet aproxima directamente la ecuacion de
segundo orden:
   d2x/dt2 = dv/dt =~ (v(t+dt/2) - v(t-dt/2)) / dt =~
     ( (x(t+dt)-x(t))/dt - (x(t)-x(t-dt))/dt ) / dt =
     ( x(t+dt)-2*x(t)+x(t-dt)) / dt^2 = a(t)
donde
   a(t) = f(x(t))/m                                    (1)
despejando x(t+dt),
   x(t+dt) =~ 2*x(t) - x(t-dt) + a(t)*dt^2.            (2)
La iteracion de (1) y (2) es el metodo de Verlet 'normal',
que nos da la
posicion en cada paso. Pero a menudo nos interesa tambien
la velocidad:
   v(t) =~ (x(t+dt)-x(t-dt))/(2*dt)
Sustituyendo x(t+dt) de (2) obtenemos
   v(t) =~ (x(t)-x(t-dt))/dt + a(t)*dt/2               (3)
A su vez, despejando x(t-dt) en (3) y sustituyendo en (2)
obtenemos
   x(t+dt) =~ x(t) + v(t)*dt + (1/2)*a(t)*dt^2
O bien, atrasando un paso la ecuacion,
   x(t) =~ x(t-dt) + v(t-dt)*dt + (1/2)*a(t-dt)*dt^2  (4)
La iteracion de (4), (1) y (3) (por ese orden) se llama
velocity-Verlet y nos da x(t), v(t) y a(t) a partir de
x(t-dt), v(t-dt) y a(t-dt). Este metodo es sencillo y
preciso pero solo puede aplicarse para fuerzas conservativas,
en que la fuerza no dependa de la velocidad.

El metodo de Runge-Kutta es mas complicado, pero puede
aplicarse a fuerzas no conservativas. Sus bases se estudiaran
en Computacion II. En este curso lo usaremos como 'caja negra'
preprogramada en la funcion rungeKutta.

x, v, a, y f pueden en general ser vectores, y todas las
ecuaciones anteriores seran entonces vectoriales. Cuando haya
n particulas moviendose en tres dimensiones, los vectores x,
etc, tendran dimension 3*n

-----------------------------

Para una masa m unida a un muelle de constante k, la fuerza
es f(x)=-k*x y el movimiento resultante es
   x(t) = A*cos(w*t-phi) = x0*cos(w*t) + (v0/w)*sin(w*t).
donde w=sqrt(k/m), w2=w^2.

-----------------------------

Escribir una funcion con la siguiente intefaz

function x = exactOscillator( w, x0, v0, dt, tmax )
% Returns the exact movement of a harmonic oscillator
% Input:
%   w    : oscillator's angular frequency, in s^-1
%   x0   : initial position, in m
%   v0   : initial velocity, in m/s
%   dt   : integration time interval, in s
%   tmax : maximum time of solutions, in s
% Output:
```

```
%   x    : oscillator positions at times t=(0:dt:tmax)

-----------------------------

Escribir una funcion con la siguiente interfaz

function x = eulerOscillator( w, x0, v0, dt, tmax )
% Finds the motion of a harmonic oscillator with Euler's method
% Input:
%   w    : oscillator's angular frequency, in s^-1
%   x0   : initial position, in m
%   v0   : initial velocity, in m/s
%   dt   : integration time interval, in s
%   tmax : maximum time of solutions, in s
% Output:
%   x    : oscillator positions at times t=(0:dt:tmax)

-----------------------------

Escribir una funcion verletOscillator.m, con la misma interfaz
que las anteriores, que resuelva la ecuacion del oscilador
armonico por el metodo de Verlet 'normal'

-----------------------------

Escribir un programa plotOscillator.m que de valores a w, x0,
v0, dt, y tmax, llame a las funciones anteriores, y dibuje
las tres trayectorias.

-----------------------------
Dia 2
-----------------------------

Escribir una funcion con la siguiente interfaz

function f = spring( x )
% Finds the force of a harmonic oscillator, with m=1 Kg and
% w=2*pi s^-1 (so that period=1 s)
%
% Input:
%   x    : position of the particle, in m
% Output:
%   f    : force in N

-----------------------------

Escribir una funcion con la siguiente interfaz

function x = verlet1( force, mass, x0, v0, dt, tmax )
% Solves Newton's equation, using Verlet's method, for
% one particle in 1D.
% Input:
%   force : function to calculate the force as f=force(x), in N
%   mass  : particle mass in kg
%   x0    : intial position, in m
%   v0    : initial velocity in m/s
%   dt    : integration time interval in s
%   tmax  : final integration time in s
% Output:
%   x     : position at each time (0:dt:tmax), in m

-----------------------------

Modficar plotOscillator para que recalcule la trayectoria llamando
a verlet1 y a newtonRK, y dibuje las trayectorias. La devuelta
por verlet1 deberia ser igual a la obtenida con verletOscilator.

-----------------------------

Cambiar el intervalo de integracion dt en plotOscillator y
observar su efecto sobre la precision de las distintas trayectorias.
Comprobar que:
- El metodo de Euler es mucho peor que los de Verlet y Runge-Kutta.
- El metodo de Runge-Kutta es mejor que el de Verlet para dt
  pequeño, pero ocurre lo contrario para dt grande.
- El metodo de Verlet siempre conserva la amplitud de oscilacion,
  y por tanto la energia.
Sugerencia: aumentar tambien el tiempo tmax y usar la herramienta
```

lupa.

Modificando spring.m, escribir una funcion dampedSpring.m que
acepte la velocidad como argumento adicional, e introduzca una
fuerza de friccion de modo que f = -kx - b*v, cuya solucion es
x(t) = (x0*cos(w*t) + (v0+c*x0)/w*sin(w*t)) * exp(-c*t),
donde w=sqrt(w0^2-c^2), w0=sqrt(k/m), c=b/(2*m).
La interfaz sera:

```
function f = dampedSpring( x, v )
% Finds the force of a damped oscillator, with m=1 Kg and
% w0=2*pi rad/s (period=1 s), and a dumping constant b (an
% internal parameter), so that f = -m*w0^2*x - b*v
%
% Input:
%   x    : position of the particle, in m
%   v    : velocity of the particle, in m/s
% Output:
%   f    : force in N
```

Escribir tambien una funcion con la interfaz

```
function x = exactDampedOscillator( w, c, x0, v0, dt, tmax )
% Returns the exact movement of a damped harmonic oscillator, of
% the form x(t) = (x0*cos(w*t) + (v0+c*x0)/w*sin(w*t)) * exp(-c*t),
% Input:
%   w    : oscillator's angular frequency, in s^-1
%   c    : damping coefficient, in s^-1
%   x0   : initial position, in m
%   v0   : initial velocity, in m/s
%   dt   : integration time interval, in s
%   tmax : maximum time of solutions, in s
% Output:
%   x    : oscillator positions at times t=(0:dt:tmax)
```

Escribir un programa plotDampedOscillator.m que llame a las
funciones newtonRK y newton45 y compare las soluciones obtenidas
con la exacta.

Dia 3

Estudiar detenidamente las funciones earthGravity.m, verlet.m y
oneShot.

Modificando earthGravity.m, escribir una funcion
earthGravityAndFriction.m, que añada una fuerza de friccion
proporcional al vector velocidad, -b*v. Escribir una funcion
twoShots.m que llame a newtonRK en vez de a verlet y añada una
segunda particula en tres dimensiones. Estudiar las trayectorias
obtenidas en funcion del parametro b de friccion.

Escribir una funcion earthGravityAndWind.m con una fuerza
proporcional a la velocidad del proyectil relativa al aire,
-b*(v-vWind), donde vWind es la velocidad del viento (horizontal).
Estudiar las trayectorias obtenidas para distintos valores de vWind.

Escribir una funcion con la siguiente interfaz

```
function f = gravity( m, r )
% Finds the gravitational force between several bodies (particles)
% Input:
%   m(np)    : mass of the np particles, in kg
%   r(nd,np) : position of np particles in nd dimensions, in m
% Output:
%   f(nd,np) : force on the np particles in each of the
%              nd dimensions, in N
```

```
Estudiar detenidamente las funciones solarSystem.m y
plotSolarSystem.m

-----------------------------

Añadir Venus y la Luna a solarSystem.m

-----------------------------
Dia 4
-----------------------------

En un collar de n masas m unidas por cables de longitud dx con
tension T, la fuerza transversal sobre la masa i es
f(i) = T/dx*(z(i+1)+z(i-1)-2*z(i)), donde z(i) es el desplazamiento
transversal, y se supone que z<<dx. Es util usar "condiciones
periodicas", como si el collar formara una circunferencia, de modo
que z(n+1)=z(1), y z(0)=z(n).

-----------------------------

Escribir una funcion con la interfaz

function f = stringForces(z,T,dx)
% Forces on a linear string, given by f(i)=k*(z(i+1)-2*z(i)+z(i-1)),
% using periodic boundary conditions (z(n+1)=z(1) and z(0)=z(n))
% Input:
%   z(1,n) : transversal displacements of n beads (m)
%   T      : string tension (N)
%   dx     : distance between beads (m)
% Output:
%   f(1,n) : forces on n beads (N)

Sugerencia: usar la funcion circshift para implementar las
condiciones periodicas.

-----------------------------

Escribir un programa stringWave.m que de valores a n, m, T, dx,
y que llame a verlet para obtener los desplazamientos en funcion
del tiempo. Representar dichos desplazamientos mediante plots
sucesivos, separados por pause. Los desplazamientos y velocidades
iniciales, z0 y v0, pueden ser los de un paquete de ondas de la
forma z(t)=exp(-0.5*(x-x0-c*t)^2/a^2)*cos(k*x-w*t) y su derivada
temporal, en t=0, siendo k=2*pi/lambda y w=c*k, donde c=sqrt(T*dx/m)
es la velocidad de grupo de las ondas. Tomar dx<<lambda<<a.
x0 y a son la posicion inicial y la anchura del paquete.

-----------------------------

Hacer la mitad de las masas diferente y comprobar como el paquete
se divide en una parte transmitida y otra reflejada en la frontera.

-----------------------------

Escribir una funcion con la interfaz

function f = membraneForces(z,nx,ny,T,dx)
% Forces on a 2D membrane of beads, given by
%   f(i,j) = k*( z(i+1,j) + z(i-1,j) + z(i,j+1) + z(i,j-1)
%                - 4*z(i,j) ),
% using periodic boundary conditions (z(nx+1,j)=z(1,j), and
% z(i,ny+1)=z(i,1))
% Input:
%   z(1,nx*ny) : vertical displacements of nx*ny beads (m)
%   nx         : number of beads in x direction
%   ny         : number of beads in y direction
%   T      : string tension (N)
%   dx     : distance between beads (m)
% Output:
%   f(1,nx*ny) : forces on nx*ny beads (N)

-----------------------------

Escribir un programa membraneWave.m, similar a stringWave, que
inicialice las variables necesarias y llame a verlet para simular
la propagacion de un paquete de ondas en una membrana.
```