

¿Cómo calculamos
movimientos con
la compu?
(Integración numérica)

1) Un poco de programación



Con el botón de "play" ejecuto todo el programa

Shift+Enter
ejecuto un
bloque

F9 ejecuto
una selección

Name	Type	Size	Value
dt	float	1	0.01
E_lista	list	3000	[0.5075, 0.5300022500012657, 0.5299295100839757, 0.5298568233992222, 0 ...
f_x	float64	1	0.2734477216863839
f_x_lista	list	2999	[2.0, -1.0000011249981016, -0.9998545001384029, -0.9995601488655005, - ...
f_y	float64	1	-0.0016624604893925186
f_y_lista	list	2999	[0.0, -0.0015000016874971525, -0.002999675992609943, -0.00449858302471 ...
frames	int	1	10
fuerza	list	3000	[-1.0, -1.0000022499987344, -0.9998589998110115, -0.9995702718915023, ...

El explorador de variables muestra el valor de variables que voy generando



Acá el código o script, la secuencia de órdenes que le doy a la compu

Esta es la consola donde también puedo ejecutar órdenes de a una

1) Un poco de programación

The image shows the Spyder Python IDE interface. The main window displays a Python script with the following code:

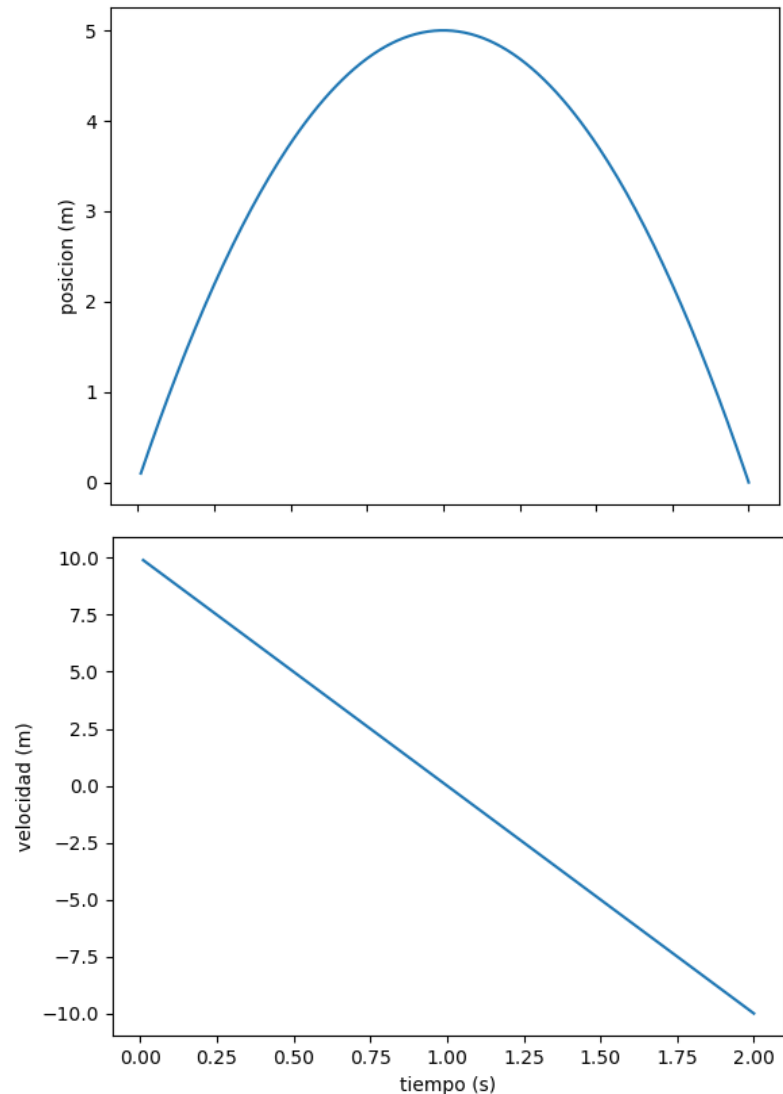
```
1  %% importamos las librerías necesarias
2  import matplotlib.pyplot as plt #librería para
3  import numpy as np #librería para hacer cuentas
4
5  %% definimos variables para ir guardando valores
6  tiempos=[] # definimos lista para los tiempos
7  posiciones=[] # definimos lista para las posiciones
8  velocidades=[] # definimos lista para las velocidades
9
10 dt=0.01 #definimos el paso del tiempo
11 numero_pasos=200 #definimos la cantidad total de pasos
12 g=-10 #defino la gravedad
13 t0=0 #definimos el tiempo inicial
14 x0=0 #definimos la posición inicial
15 v0=10 #definimos la velocidad inicial
16
17 %% generamos los valores de tiempos, posiciones y velocidades de un MRUV
18 t=t0
19 v=v0
20 for i in range(0, numero_pasos):
21     t=t+dt #avanzamos el tiempo en cada paso
22     x=x0+v0*t+g/2*t**2 #tomamos donde estaba el paso anterior
23     v=v0+g*t
24
25     tiempos.append(t) #guardamos el tiempo nuevo en la lista de tiempos
26     posiciones.append(x) #guardamos la posición nueva en la lista
27     velocidades.append(v) #guardamos la velocidad nueva en la lista
28
```

The Preferences dialog box is open, showing the 'Graphics' tab. The 'Support for graphics (Matplotlib)' section has 'Activate support' checked. The 'Graphics backend' section has 'Backend' set to 'Automatic'. The 'Inline backend' section has 'Format' set to 'PNG' and 'Resolution' set to '72,0 dpi'. The 'Reset to defaults' button is visible at the bottom left of the dialog box.

194 de 200

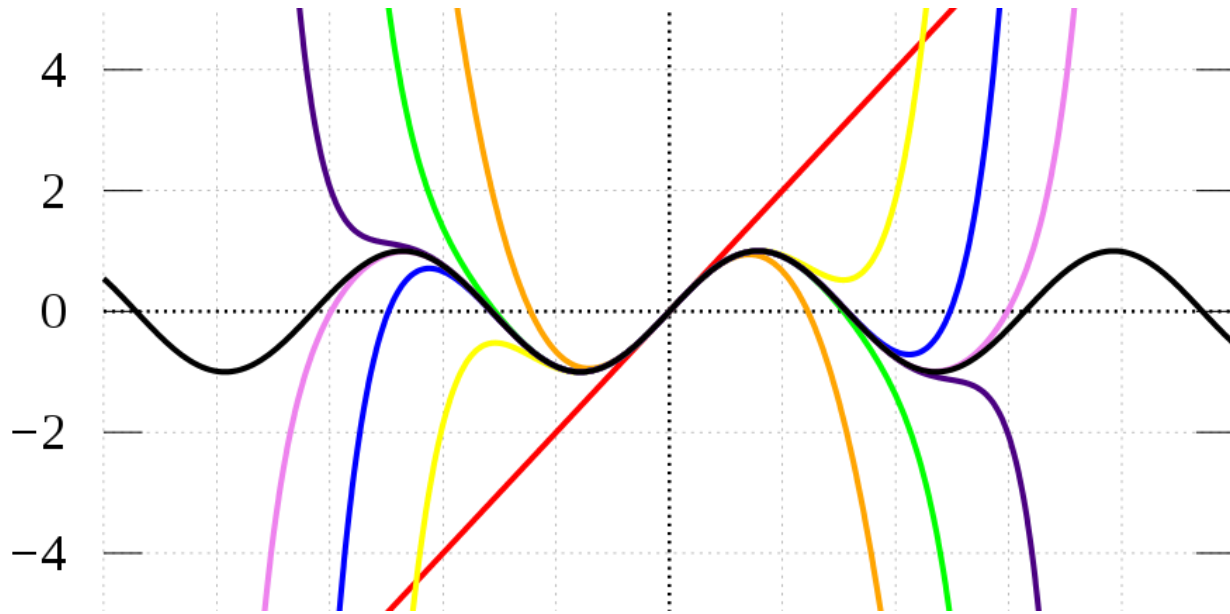
1) Un poco de programación

```
1  %% importamos las librerias necesarias
2  import matplotlib.pyplot as plt #libreria para graficar
3  import numpy as np #libreria para hacer cuentas
4
5  %% definimos variables para ir guardando valores
6  tiempos=[] # definimos lista para los tiempos
7  posiciones=[] # definimos lista para las posiciones
8  velocidades=[] # definimos lista para las velocidades
9
10 dt=0.01 #definimos el paso del tiempo
11 numero_pasos=200 #definimos la cantidad total de pasos
12 g=-10 #defino la gravedad
13 t0=0 #definimos el tiempo inicial
14 x0=0 #definimos la posicion inicial
15 v0=10 #definimos la velocidad inicial
16
17 %% generamos los valores de tiempos, posiciones y velocidades de un MRUV
18 t=t0
19 v=v0
20 for i in range(0, numero_pasos):
21     t=t+dt #avanzamos el tiempo en cada paso
22     x=x0+v0*t+g/2*t**2 #tomamos donde estaba el paso anterior
23     v=v0+g*t
24
25     tiempos.append(t) #guardamos el tiempo nuevo en la lista de tiempos
26     posiciones.append(x) #guardamos la posicion nueva en la lista
27     velocidades.append(v) #guardamos la velocidad nueva en la lista
28
29     print(str(i)+' de '+str(numero_pasos)) #hacemos que imprima el valor de i en pantalla
30
31 %% graficamos
32 #Grafico la posición vs tiempo
33 plt.figure() #abrimos una nueva figura para graficar
34 plt.plot(tiempos, posiciones) #hago el grafico
35 plt.xlabel('tiempo (s)') #nombre del eje X
36 plt.ylabel('posicion (m)') #nombre del eje Y
37 plt.show() #muestra la figura
38
39 #Grafico la posición vs tiempo
40 plt.figure() #abrimos una nueva figura para graficar
41 plt.plot(tiempos, velocidades) #hago el grafico
42 plt.xlabel('tiempo (s)') #nombre del eje X
43 plt.ylabel('velocidad (m)') #nombre del eje Y
44 plt.show() #muestra la figura
```



2) Un poco de matemática

Teorema de Taylor: puedo aproximar $f(x)$ alrededor de un x_0 mediante un polinomio



$$r(t + dt) = r(t) + \dot{r}(t)dt + \frac{\ddot{r}(t)}{2!} dt^2 + \frac{\dddot{r}(t)}{3!} dt^3 + \dots$$

$$r(t + dt) \approx r(t) + \dot{r}(t)dt + \frac{\ddot{r}(t)}{2} dt^2 + \frac{\ddot{r}(t)}{3!} dt^3$$
$$r(t - dt) \approx r(t) - \dot{r}(t)dt + \frac{\ddot{r}(t)}{2} dt^2 - \frac{\ddot{r}(t)}{3!} dt^3$$

} +

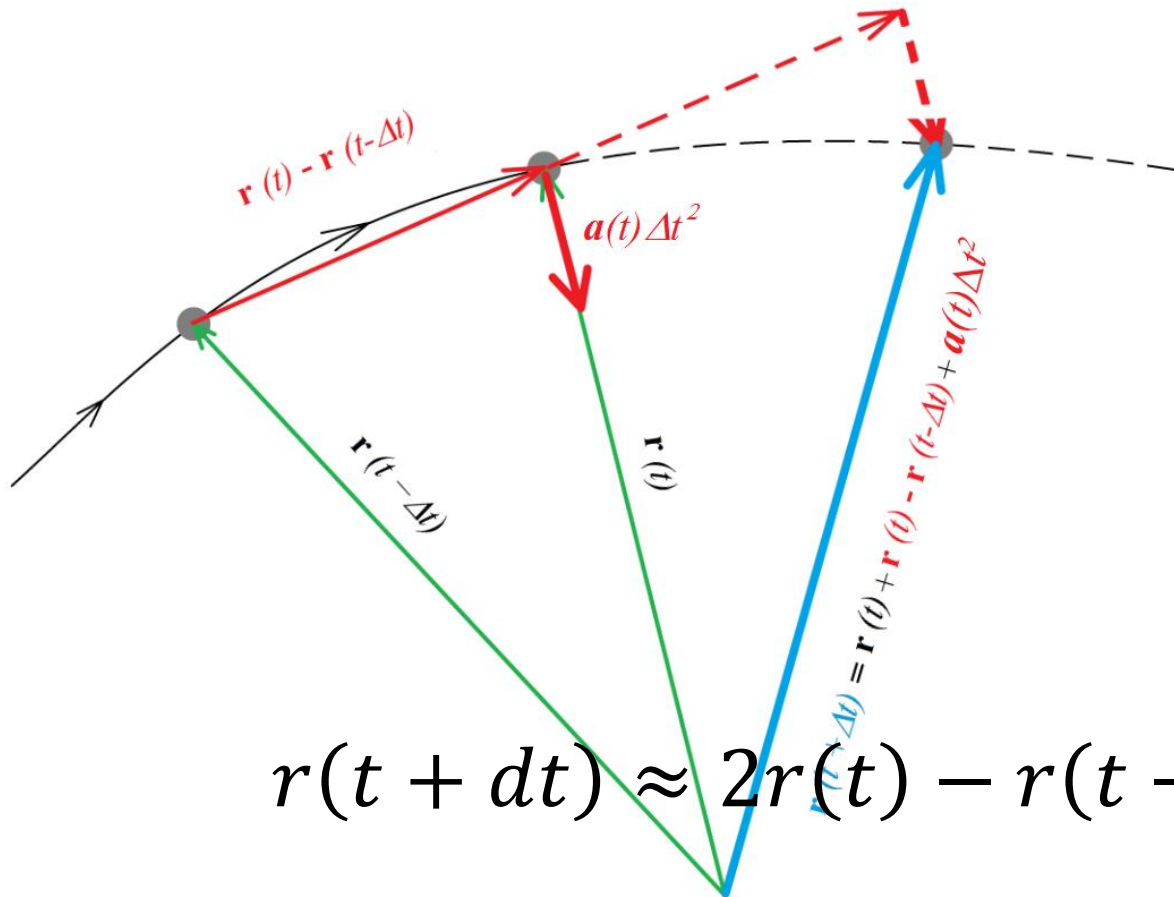
$$r(t + dt) + r(t - dt) \approx 2r(t) + \ddot{r}(t)dt^2$$

$$r(t + dt) \approx 2r(t) - r(t - dt) + \ddot{r}(t)dt^2$$

$$r(t + dt) \approx 2r(t) - r(t - dt) + \frac{\sum F(r(t), t)}{m} dt^2$$

2) Un poco de matemática

$$r(t + dt) \approx 2r(t) - r(t - dt) + \frac{\sum F(r(t), t)}{m} dt^2$$



Sabiendo dónde está el sistema ahora $r(t)$, de dónde vino $r(t - dt)$ y las fuerzas que recibe $\sum F(r(t), t)$, podemos predecir dónde estará $r(t + dt)$

La hipótesis es que el dt es lo suficientemente chico como para asumir que $\sum F(r(t), t)$ es cte.

$$r(t + dt) \approx 2r(t) - r(t - dt) + \frac{\sum F(r(t), t)}{m} dt^2$$

3) Y que se haga la física

$$r(t + dt) \approx 2r(t) - r(t - dt) + \frac{\sum F(r(t), t)}{m} dt^2$$

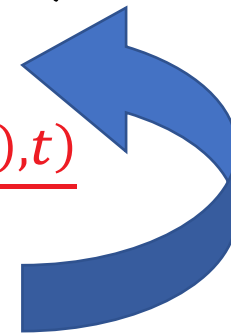
Algoritmo de Verlet:

1) Elegimos una posición $r(t)$

2) Calculamos la fuerza y la aceleración $\frac{\sum F(r(t), t)}{m}$

3) Calculamos la posición siguiente $r(t + dt)$

Loop o ciclo



$t \oplus t + dt$

es asignación,
no igualdad

La idea es tomar un dt pequeño donde valga la aproximación de Taylor y que $F(r(t), t) = cte$

Algoritmo de Verlet:

1) Elegimos una posición $r(t)$

2) Calculamos la fuerza y la
aceleración $\frac{\sum F(r(t),t)}{m}$

3) Calculamos la posición
siguiente $r(t + dt)$

$$r(t + dt) \approx 2r(t) - r(t - dt) + \frac{\sum F(r(t),t)}{m} dt^2$$

```
110 #%% Algoritmo de Verlet
111 pasos = 1000 #número de pasos de la simulación
112 dt=0.01
113 t0=0
114
115 tita=[]
116 tita.append(posicion_inicial)
117 tita.append(posicion_inicial+velocidad_inicial*dt)
118
119 tiempo = [t0-dt, t0]
120 velocidad_tangencial = [(tita[1]-tita[0])/dt]
121 aceleracion=[-g/L*np.sin(tita[0]),-g/L*np.sin(tita[1])-viscosidad]
122 tension=[g*np.cos(tita[0])+L*velocidad_tangencial[0]**2]
123
124 tiempo_actual=tiempo[1]
125 for i in range(1, pasos - 1):
126     #actualizo las posiciones actual y previa
127     tita_actual = tita[i]
128     tita_prev = tita[i-1]
129     #actualizo la velocidad actual
130     velocidad_tangencial.append((tita_actual-tita_prev)/dt)
131     #calculo la aceleración actual y la guardo
132     a_tita = -g/L*np.sin(tita[i])-viscosidad*velocidad_tangencial[i]
133     aceleracion.append(a_tita)
134     #calculo la posición nueva y la guardo
135     tita_nueva = 2 * tita_actual - tita_prev + a_tita * dt**2
136     tita.append(tita_nueva)
137     #calculo la tension y la guardo
138     tension.append(g*np.cos(tita_actual)+L*velocidad_tangencial[i]**2)
139     #actualizo el tiempo y lo guardo
140     tiempo_actual=tiempo_actual+dt
141     tiempo.append(tiempo_actual)
142
```


¿Qué hacemos?

Hay 3 programas:

1) Resorte2masitas

2) Péndulo (barra)

3) Kepler (órbita de la tierra)

Jueguen

Resuelvan la guía

Resuelvan la guía

¿Qué hacemos?

Hay 3 programas:

1) Resorte2masitas

2) Péndulo (barra)

3) Kepler (órbita de la tierra)

Jueguen

Resuelvan la guía

Resuelvan la guía

A jugar