Tema 0 - Programación Básica con Python
Clase 4

Curso de Física Computacional

M. en C. Gustavo Contreras Mayén

Contenido

Contenido

Graficación con Python

Una buena parte del trabajo que tendremos que hacer como físicos es utilizar un conjunto de datos que por si solos, no van a darnos información sobre un modelo o un fenómeno, por ello, será necesario usar gráficas.

Pythonincluye un módulo de graficación bastante versátil para generar gráficas y exportarlas a diferentes tipos de archivos.

La librería se llama `matplotlib`. Haremos algunos ejercicios para demostrar su potencia.

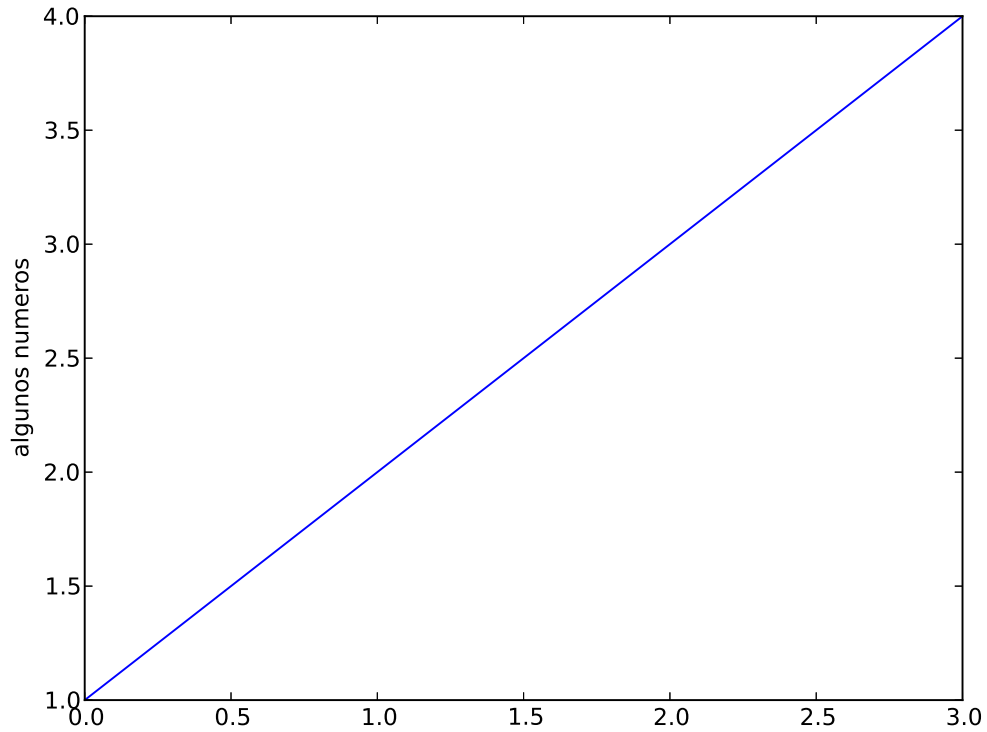
`matplotlib.pyplot` es una colección de funciones de estilo de mando, de tal manera que matplotlib funciona a la manera de MATLAB. Cada instrucción `pyplot` aplica un cambio a una figura: por ejemplo, crear una figura, crear un área de trazado en una figura, trazar algunas líneas en un área de trazado, decorar con etiquetas, etc.

Ejecicio 1

```
1 import matplotlib.pyplot as plt
2 plt.plot([1,2,3,4])
3 plt.ylabel('algunos numeros')
4 plt.show()
```

Ejecicio 1

```
1 import matplotlib.pyplot as plt
2 plt.plot([1,2,3,4])
3 plt.ylabel('algunos numeros')
4 plt.show()
```



Te estarás preguntando por qué tenemos en el eje x el rango $0 - 3$ y en el eje y el rango $1 - 4$.

Te estarás preguntando por qué tenemos en el eje x el rango $0 - 3$ y en el eje y el rango $1 - 4$.

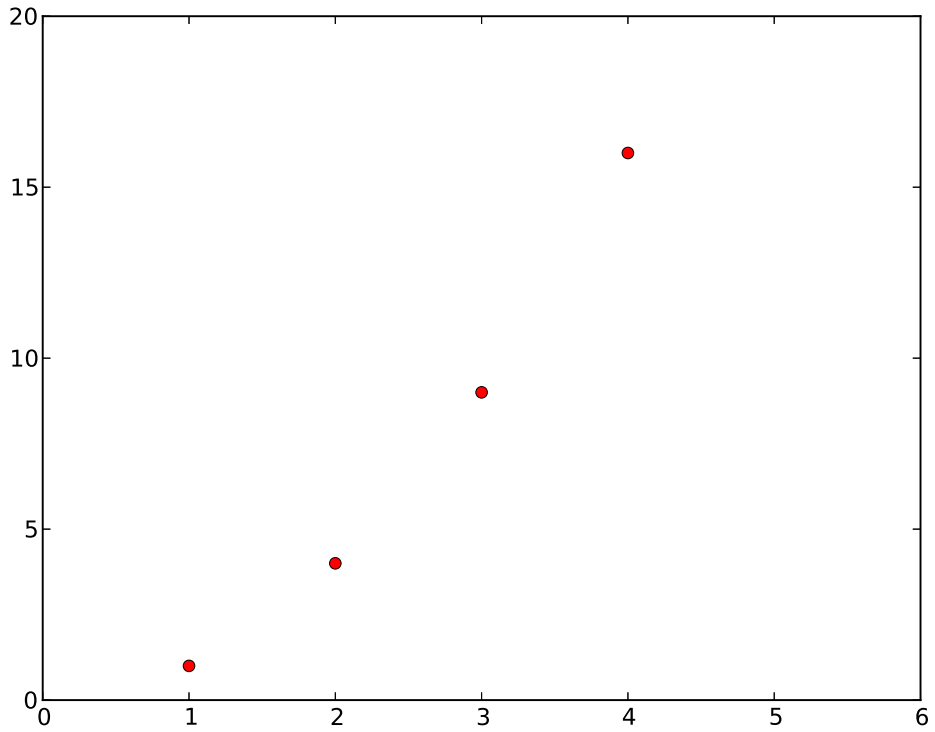
Si proporcionamos una única lista o matriz en el comando `plot`, `matplotlib` asume que es una secuencia de valores de y, por lo que genera automáticamente los valores de x para nosotros. Como los índices en `Python`comienzan en 0, el vector x por defecto tiene la misma longitud que y, pero inicia con 0. De ahí que los datos x son $[0, 1, 2, 3]$.

Ejecicio 2

```
1 import matplotlib.pyplot as plt
2 plt.plot([1,2,3,4], [1,4,9,16], 'ro')
3 plt.axis([0, 6, 0, 20])
4 plt.show()
```

Ejecicio 2

```
1 import matplotlib.pyplot as plt
2 plt.plot([1,2,3,4], [1,4,9,16], 'ro')
3 plt.axis([0, 6, 0, 20])
4 plt.show()
```



Por cada par x, y de argumentos, existe un tercer argumento opcional, que es la cadena de formato que indica el color y tipo de línea.

Las letras y los símbolos de la cadena de formato son como en MATLAB, y concatenar una cadena de color con una cadena estilo de línea.

La cadena de formato por defecto es `'b-'`, que es una línea de color azul.

Tipos de líneas

carácter	descripción
'-'	línea sólida
'--'	línea cortada
'-.'	línea-punto
':'	línea de puntos
'.'	marca de punto
','	marca de pixel
'o'	marca de círculo
'v'	marca de triándulo hacia abajo
'^'	marca de triángulo hacia arriba

Lista de colores

carácter	color
`'b'`	azul
`'g'`	verde
`'r'`	rojo
`'c'`	cyan
`'m'`	magenta
`'y'`	amarillo
`'k'`	negro
`'w'`	blanco

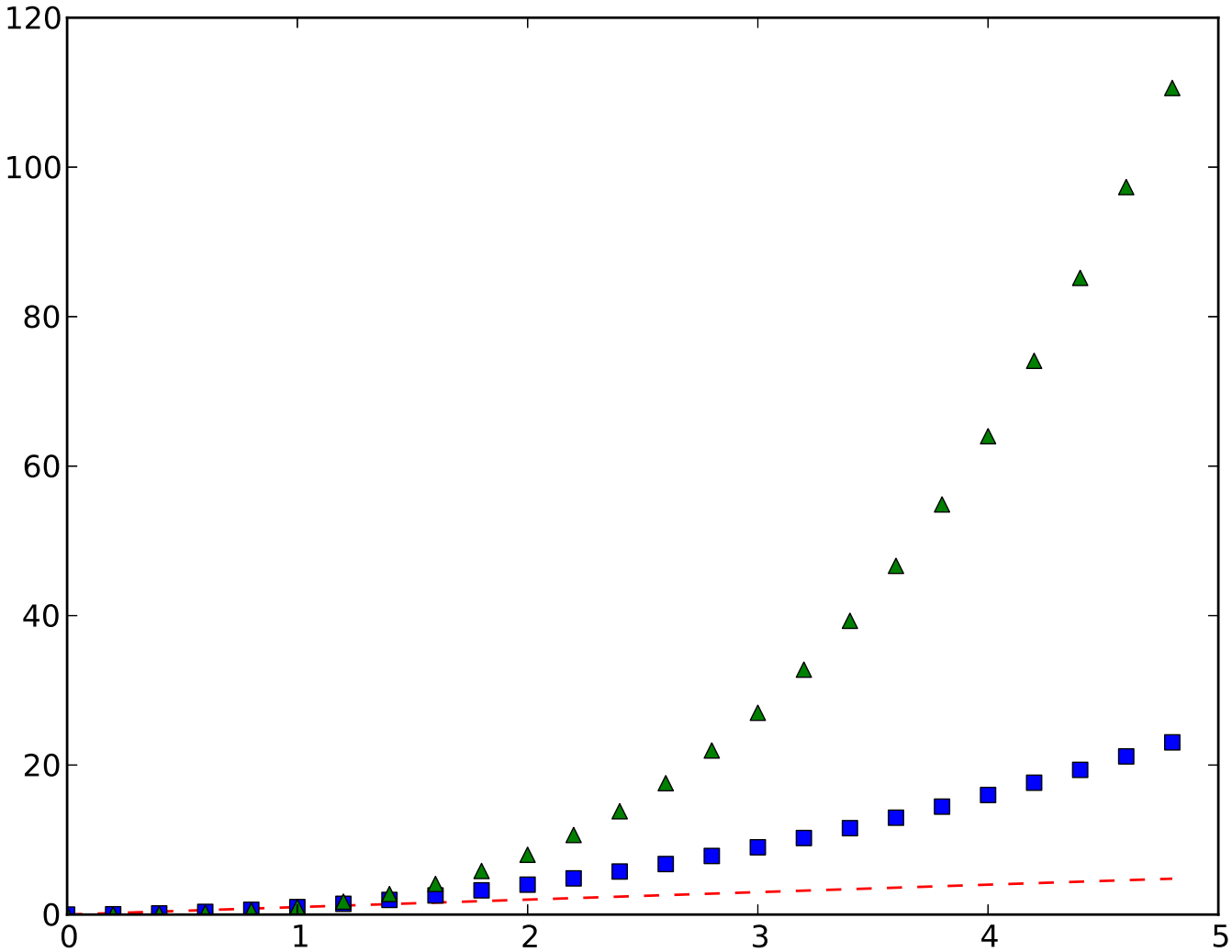
`matplotlib` se limita a trabajar con listas, por lo que sería bastante acotado para el procesamiento y análisis numérico.

Por lo general, se utilizan los arreglos del módulo `numpy`. De hecho, todas las secuencias se convierten en matrices de `numpy` internamente.

El siguiente ejemplo ilustra un trazado de líneas con varios estilos diferentes en una sola instucción utilizando arreglos.

Ejecicio 3

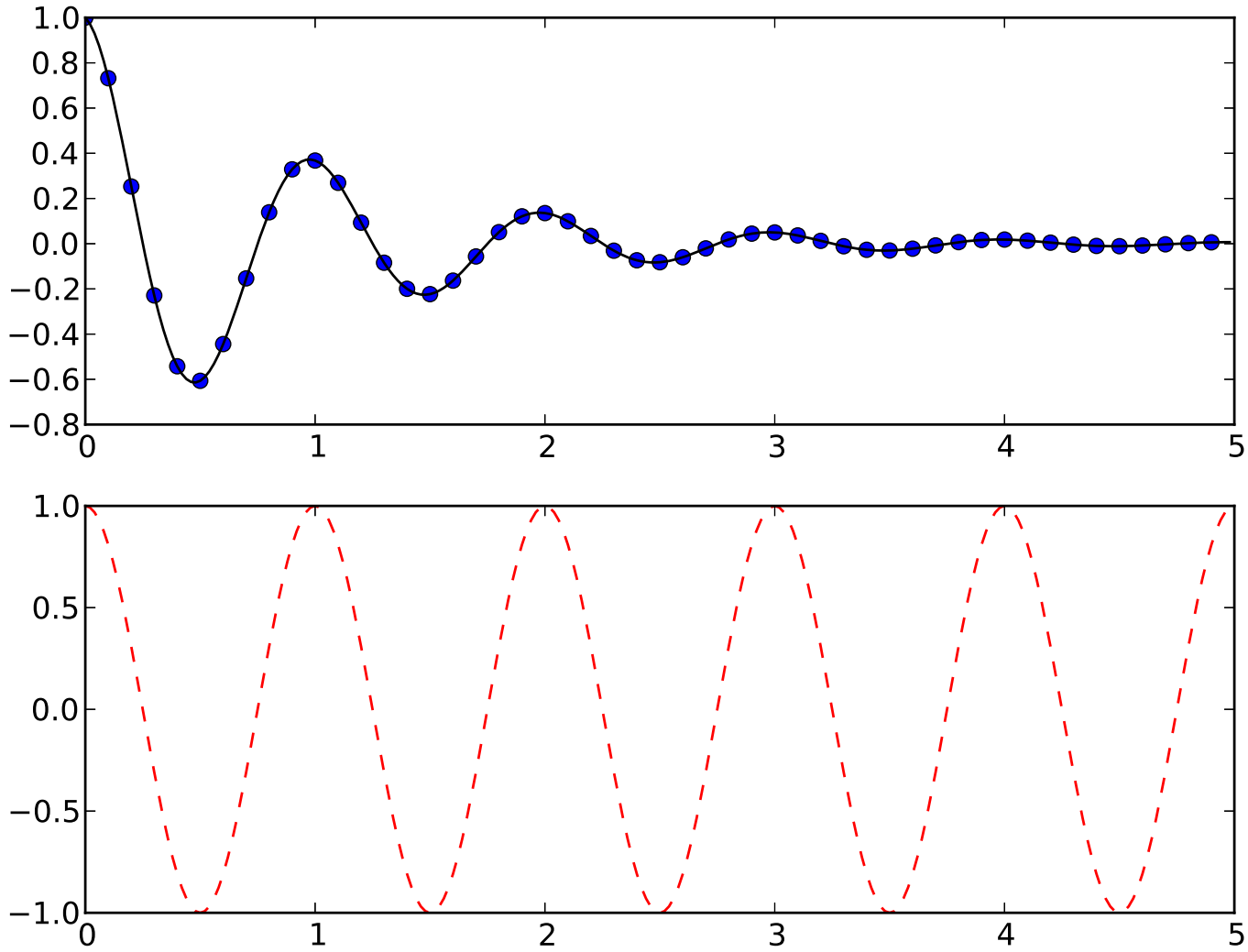
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 t = np.arange(0., 5., 0.2)
5 plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3,
      'g^')
6 plt.show()
```


120
100
80
60
40
20
0
0
1
2
3
4
5

Ejercicio 4

Trabajando con múltiples gráficas

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  def f(t):
5      return np.exp(-t) * np.cos(2*np.pi*t)
6
7  t1 = np.arange(0.0, 5.0, 0.1)
8  t2 = np.arange(0.0, 5.0, 0.02)
9
10 plt.figure(1)
11 plt.subplot(211)
12 plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
13
14 plt.subplot(212)
15 plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```

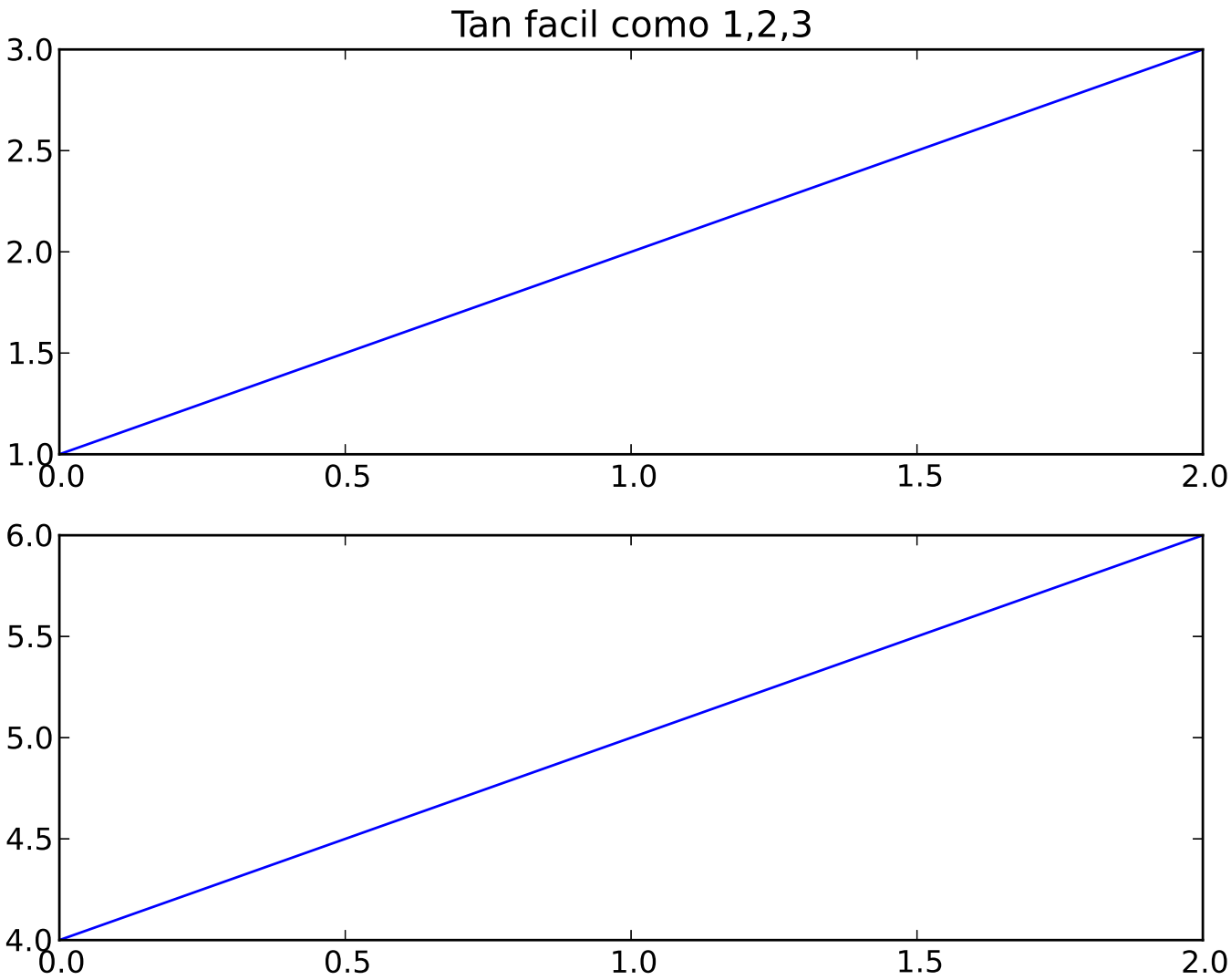

1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
1
2
3
4
5
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5

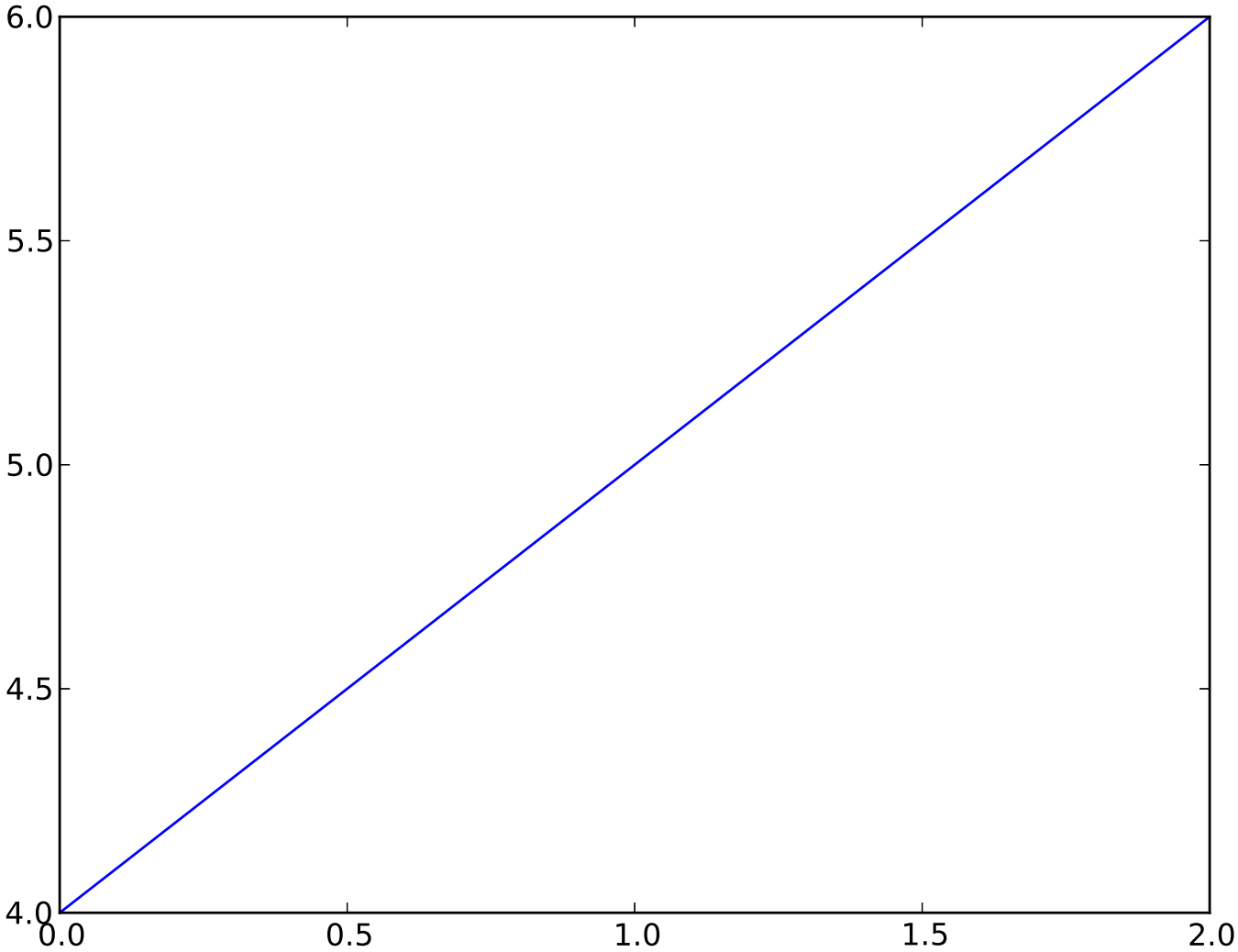
El comando `figure()` aquí es opcional, ya `figure(1)` se crea de forma predeterminada, así mismo `subplot(111)` se crea de forma predeterminada si no se especifica manualmente un eje.

El comando `subplot()` especifica `numrows, numcols, fignum` donde `fignum` varía en rango de 1 a `numrows * numcols`. Las comas en el comando `subplot()` son opcionales si `numrows * numcols` < 10. Por tanto `subplot(211)` es idéntica a la `subplot(2,1,1)`.

Ejercicio 5

```
1  import matplotlib.pyplot as plt
2
3  plt.figure(1)
4  plt.subplot(211)
5  plt.plot([1,2,3])
6  plt.subplot(212)
7  plt.plot([4,5,6])
8
9
10 plt.figure(2)
11 plt.plot([4,5,6])
12
13 plt.figure(1)
14 plt.subplot(211)
15 plt.title('Tan facil como 1,2,3')
16 plt.show()
```


Tan facil como 1,2,3
3.0
2.5
2.0
1.5
1.0
0.0
0.5
1.0
1.5
2.0
6.0
5.5
5.0
4.5
4.0
0.0
0.5
1.0
1.5
2.0


6.0
5.5
5.0
4.5
4.0
0.0
0.5
1.0
1.5
2.0

Más recursos para graficar con Python

Lo que hemos visto es una revisión muy básica y general de cómo generar una gráfica con Python, hay todavía una enorme cantidad de información sobre matplotlib, encontrarás en la página oficial de la librería, bastante documentación, ejemplos y elementos para extender completamente esta herramienta.

Más recursos para graficar con Python

Lo que hemos visto es una revisión muy básica y general de cómo generar una gráfica con Python, hay todavía una enorme cantidad de información sobre matplotlib, encontrarás en la página oficial de la librería, bastante documentación, ejemplos y elementos para extender completamente esta herramienta.

Se les proporcionará una guía breve de graficación, con la intención de que revisen casos prácticos aplicados a la física. Para cada gráfica que usemos más adelante en el curso, tendrán oportunidad de agregar más elementos que ustedes consideren.

Contenido

Primeros problemas de la física

De manera paralela a los conceptos importantes del curso, es necesario iniciar el trabajo de plantear algoritmos de solución a problemas de la física.

La mejor manera de aprender, es sentarse a programar. Debe de tenerse la calma para ello, la idea es ir perfeccionando las propuestas de solución, es importante señalar que la inspiración divina, no se da siempre.

Proceso de programación

El proceso de programación consta de las actividades necesarias para escribir programas que funcionen adecuadamente como solución a un problema particular. [1]

1. Definición del problema.

[1] Amparo López Gaona, *Introducción al desarrollo de programas con Java*, 3a. Ed., La Prensa de Ciencias, México D.F. 2013.

Proceso de programación

El proceso de programación consta de las actividades necesarias para escribir programas que funcionen adecuadamente como solución a un problema particular. [1]

1. Definición del problema.
2. Diseño de la solución.

[1] Amparo López Gaona, *Introducción al desarrollo de programas con Java*, 3a. Ed., La Prensa de Ciencias, México D.F. 2013.

Proceso de programación

El proceso de programación consta de las actividades necesarias para escribir programas que funcionen adecuadamente como solución a un problema particular. [1]

1. Definición del problema.
2. Diseño de la solución.
3. Codificación.

[1] Amparo López Gaona, *Introducción al desarrollo de programas con Java*, 3a. Ed., La Prensa de Ciencias, México D.F. 2013.

Proceso de programación

El proceso de programación consta de las actividades necesarias para escribir programas que funcionen adecuadamente como solución a un problema particular. [1]

1. Definición del problema.
2. Diseño de la solución.
3. Codificación.
4. Depuración.

[1] Amparo López Gaona, *Introducción al desarrollo de programas con Java*, 3a. Ed., La Prensa de Ciencias, México D.F. 2013.

Proceso de programación

El proceso de programación consta de las actividades necesarias para escribir programas que funcionen adecuadamente como solución a un problema particular. [1]

1. Definición del problema.
2. Diseño de la solución.
3. Codificación.
4. Depuración.
5. Mantenimiento.

[1] Amparo López Gaona, *Introducción al desarrollo de programas con Java*, 3a. Ed., La Prensa de Ciencias, México D.F. 2013.

Definición del problema

Aquí se especifica qué es lo que debe de hacer el programa.

Este primer paso puede parecer trivial aunque no lo es. La comprensión exacta de lo que se necesita hacer es requisito indispensable para crear una solución funcional.

En ocasiones, se ignora esta fase y se comienza a escribir un programa sin tener en claro el problema a resolver.

Diseño de la solución

En esta fase se indica una forma de satisfacer, mediante un programa, los requerimientos establecidos en la etapa anterior.

El diseño de un programa es un proceso al que muchas veces no se le da la importancia y de ahí que en las etapas posteriores se tengan muchos problemas.

En el diseño es necesario identificar los principales componentes de la solución y la relación entre ellos.

Codificación

Una vez que se tiene el diseño de la solución, se procede a traducirlo a un lenguaje de programación.

Esta tarea se conoce como codificación o implementación. En muchas ocasiones, uno se centra únicamente en esta etapa aunque, como se puede ver, el proceso de programar es mucho más complejo y creativo.

Es recomendable acostumbrarse desde el inicio a escribir programas que sean fácilmente entendibles por otras personas; podemos apoyarnos con lo siguiente:

1. Los programas deben de tener una estructura clara.

Es recomendable acostumbrarse desde el inicio a escribir programas que sean fácilmente entendibles por otras personas; podemos apoyarnos con lo siguiente:

1. Los programas deben de tener una estructura clara.
2. El código debe estar organizado y presentado de manera que sea fácil su lectura.

Es recomendable acostumbrarse desde el inicio a escribir programas que sean fácilmente entendibles por otras personas; podemos apoyarnos con lo siguiente:

1. Los programas deben de tener una estructura clara.
2. El código debe estar organizado y presentado de manera que sea fácil su lectura.
3. El código debe de estar documentado.

Depuración

El siguiente paso en el desarrollo de un programa es la depuración que consiste en verificar que el algoritmo y el programa sean adecuados. No importa que tan bonito esté el programa, si no produce los resultados deseados, simplemente no sirve.

Depurar implica descubrir, localizar y corregir todos los errores que causen que un programa produzca resultados incorrectos o que no produzca ningún resultado.

Mantenimiento

En los programas y trabajos escolares, la tarea termina en el paso anterior, pero en la vida real no es así. La etapa de mantenimiento consiste en supervisar la operación de un programa, corregir cualquier error encontrado durante su uso continuo o efectuar modificaciones al mismo, con el propósito de que realice más tareas o de manera diferente a las que tenían contempladas originalmente.

Un primer problema de Mecánica

Supongamos que tenemos una partícula de masa m que está confinada a moverse a lo largo del eje x, bajo una fuerza $f(x)$. Sabemos de la ley de Newton que

$$f = ma = m\frac{dv}{dt} \tag{1}$$

donde a es la aceleración y v la velocidad de la partícula respectivamente, t es el tiempo.

Si dividimos el tiempo en pequeños intervalos iguales $\tau = t_{i+1} - t_i$, sabemos que la velocidad en el tiempo t_i, está dada de manera aproximada por el promedio de la velocidad en el intervalo de tiempo $[t_i, t_{i+1}]$

Por lo que

$$v_i \simeq \frac{x_{i+1} - x_i}{t_{i+1} - t_i} = \frac{x_{i+1} - x_i}{\tau} \tag{2}$$

La aceleración de la partícula es aproximadamente el promedio de la aceleración en el mismo intervalo

$$a_i \simeq \frac{v_{i+1} - v_i}{t_{i+1} - t_i} = \frac{v_{i+1} - v_i}{\tau} \tag{3}$$

donde τ es muy pequeño.

El algoritmo más sencillo para determinar la posición y velocidad de la partícula en el tiempo t_{1i+1}, a partir de las cantidades correspondientes al tiempo t_i, se obtiene luego de combinar las ecuaciones (**??**), (**??**) y (**??**), por lo que

$$x_{i+1} = x_i + \tau v_i \tag{4}$$

$$v_{i+1} = v_i + \frac{\tau}{m} f_i \tag{5}$$

donde $f_i = f(x_i)$

Si se proporciona la posición inicial y la velocidad de la partícula y se buscan las cantidades correspondientes en algún momento posterior (problema de valor inicial), podemos obtenerlas de forma recursiva a partir del algoritmo dado en las ecuaciones (**??**) y (**??**).

Ejercicio 1: Oscilador mecánico

Por simplicidad, consideremos que la fuerza es $f(x) = -kx$, donde k es la constante del resorte. Usemos $m = k = 1$. Queremos describir la posición y velocidad de la partícula en un intervalo de tiempo de 100 segundos. Las condiciones iniciales son las siguientes: $x(t = 0) = 0$ y $v(t = 0) = 1$.

Resolviendo el problema con Python

¿Qué es lo que tenemos?

- El intervalo de tiempo de $[0, 100]$ segundos.

Resolviendo el problema con Python

¿Qué es lo que tenemos?

- El intervalo de tiempo de $[0, 100]$ segundos.
- La posición y velocidad inicial.

Resolviendo el problema con Python

¿Qué es lo que tenemos?

- El intervalo de tiempo de $[0, 100]$ segundos.
- La posición y velocidad inicial.
- Las expresiones para calcular los x_{i+1} y v_{i+1}

Resolviendo el problema con Python

¿Qué es lo que tenemos?

- El intervalo de tiempo de $[0, 100]$ segundos.
- La posición y velocidad inicial.
- Las expresiones para calcular los x_{i+1} y v_{i+1}

¿Qué nos falta?

Lo que nos falta

- Calcular para cada segundo en $[0, 100]$ la posición y velocidad.

Lo que nos falta

- Calcular para cada segundo en $[0, 100]$ la posición y velocidad.
- Guardar esos valores en un algún lado.

Lo que nos falta

- Calcular para cada segundo en $[0, 100]$ la posición y velocidad.
- Guardar esos valores en un algún lado.
- Graficar los resultados.

Uso de los módulos para nuestro programa

Para graficar con el módulo `matplotlib` incluido en Python, necesitamos llamar a la librería `pyplot`, para acortar la escritura, usamos un *alias*, en este caso `plt`.

```
1 import matplotlib.pyplot as plt
2 from math import pi
```

Usando la información que nos proporciona el problema

```
1 n = 100
2
3 x = []
4 v = []
5
6 dt = 2* pi/n
7
8 x.append(0)
9 v.append(1)
```

Ciclo de iteración para los nuevos valores

Como la fuerza en este caso es del tipo $f(x) = -kx$, tenemos un cambio de signo en el segundo sumando de vi.

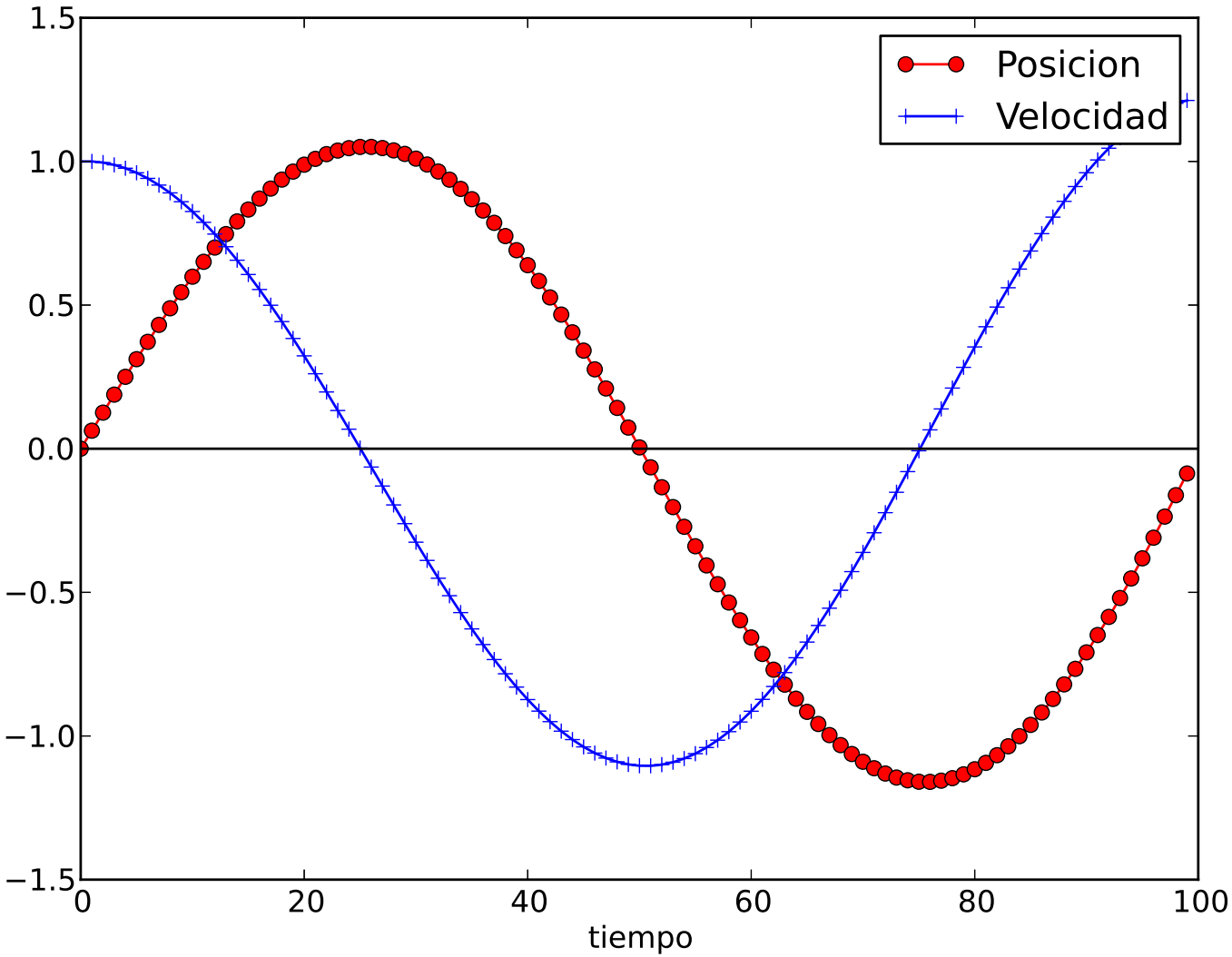
```
1 for i in range(n-1):
2     xi = x[i] + v[i]*dt
3     vi = v[i] - x[i]*dt
4
5     x.append(xi)
6     v.append(vi)
```

Generación de una gráfica

La instrucción `plot`, nos permite crear en una nueva ventana, la gráfica, en este caso, requiere de una variable para poder desplegar los valores contenidos en la lista; el parámetro "ro-" lo usamos para decirle a matplotlib que nos devuelva una línea roja, con el símbolo "o", que estará "unido" entre sí por cada valor de la lista, además le agregamos una etiqueta para identificar nuestra gráfica. Hacemos lo mismo para la otra gráfica que nos representa la velocidad de la masa, ahora con color azul y el símbolo "+".

```
1 plt.plot(x, "ro-", label="Posicion")
2 plt.plot(v, "b+-", label="Velocidad")
3 plt.legend(loc="upper right")
4 plt.xlabel("tiempo")
5 plt.show()
```

Resultado del problema



Ejecicio 2: Efecto de la resistencia del aire

La bicicleta es una forma muy eficiente de transporte, este es un hecho bien conocido por cualquier persona que monta una. Nuestro objetivo en este ejercicio es comprender los factores que determinan la velocidad máxima de una bicicleta y estimar la velocidad de un caso real.

Comenzaremos haciendo caso omiso de la fricción; tendremos que añadirlo al final, por supuesto, pero debemos primero entender cómo lidiar con el caso más simple y sin fricción.

La ecuación de movimiento corresponde a la segunda ley de Newton, que escribimos de la forma

$$\frac{dv}{dt} = \frac{F}{m} \tag{6}$$

donde v es la velocidad, m es la masa de la combinación de la bicicleta-conductor, t es el tiempo, y F es la fuerza en la bicicleta que viene del esfuerzo del conductor (en este caso vamos a suponer que la bicicleta se mueve sobre un terreno plano)

Tratar correctamente a F se complica por la mecánica de la bicicleta, ya que la fuerza ejercida por el ciclista se transmite a las ruedas por medio del plato, engranajes, cadena, etc. Esto hace que sea muy difícil derivar una expresión exacta para F.

Sin embargo, hay otra manera de abordar este problema que evita la necesidad de conocer la fuerza. Este enfoque alternativo implica la formulación del problema en términos de la potencia generada por el ciclista.

Estudios fisiológicos de ciclistas de carreras han demostrado que estos atletas son capaces de producir una potencia de salida de aproximadamente 400 watts durante largos períodos de tiempo (~ 1 h)

Usando las ideas de trabajo-energía podemos reescribir (**??**) como

$$\frac{dE}{dt} = P \tag{7}$$

donde E es la energía total, P es la potencia de salida del ciclista. Para un trayecto plano la energía es totalmente cinética, es decir, $E = \frac{1}{2}mv^2$, y $\frac{dE}{dt} = mv(\frac{dv}{dt})$, usando esto en (**??**), resulta

$$\frac{dv}{dt} = \frac{P}{mv} \tag{8}$$

Si P es una constante, la ecuación (**??**), se puede resolver de manera analítica, rearreglando términos:

$$\int_{v_0}^{v} v' dv' = \int_{0}^{t} \frac{P}{m} dt' \tag{9}$$

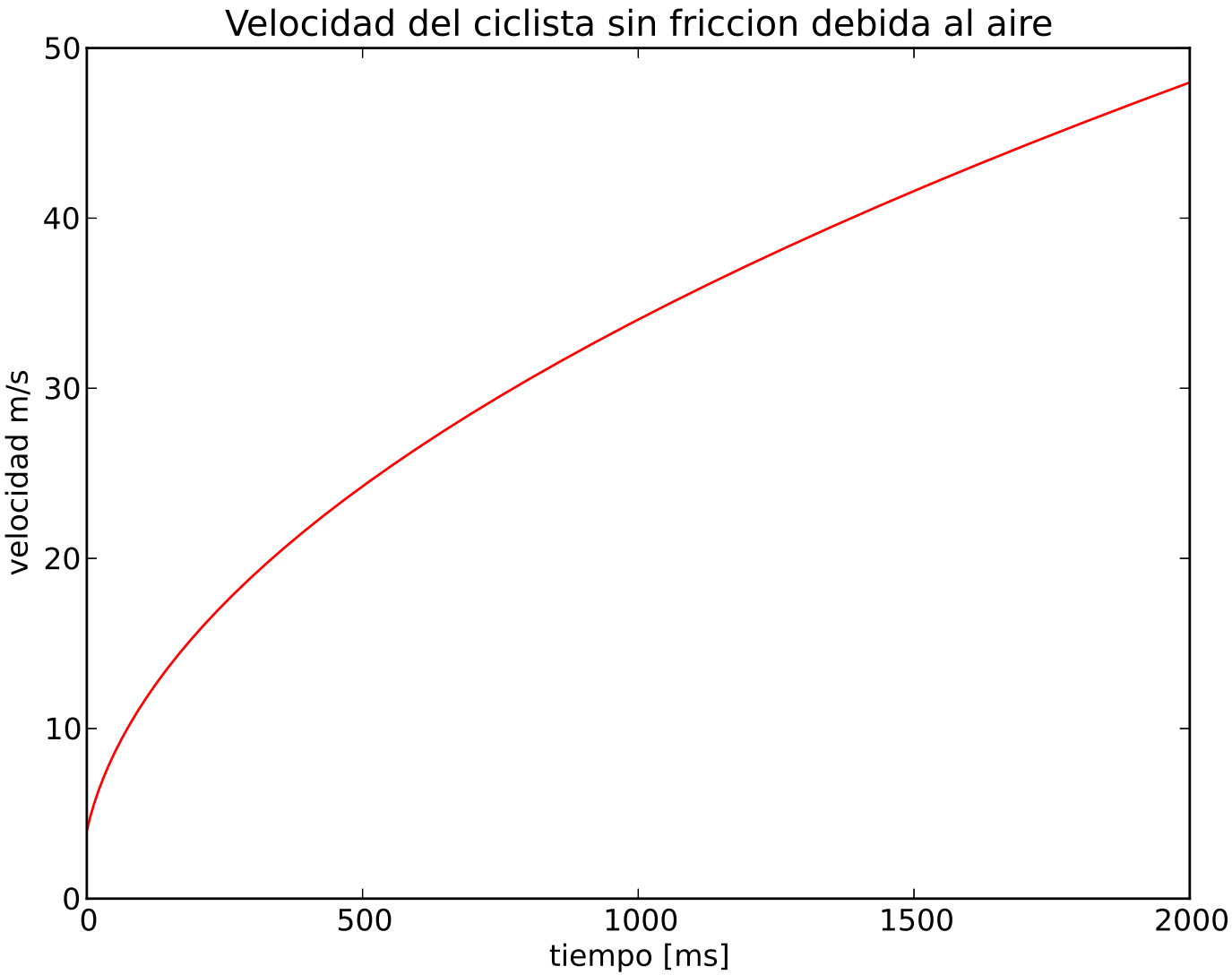
donde v_0 es la velocidad de la bicicleta en $t = 0$. Integrando ambos lados de la ecuación y resolviendo para v, tenemos

$$v = \sqrt{v_0^2 + 2P\frac{t}{m}} \tag{10}$$

Considera lo siguiente

```
t = []
v = []

dt = 1

potencia = 400

masa = 70

tmax = 200

nmax = tmax/dt


t.append(0)
v.append(4)
```

Resultado de la velocidad sin fricción



Si bien esta es la solución correcta de la ecuación de movimiento (**??**), nuestro trabajo no concluye aquí, ya que predice que la velocidad se incrementará sin límite para tiempos muy largos.

Vamos a corregir este resultado, cuando se generaliza el modelo se debe de incluir el efecto de la resistencia del aire. El nuevo término que vamos a añadir a la ecuación de movimiento nos obliga a desarrollar una solución numérica, así que con eso en mente se considera un tratamiento numérico de (**??**)

Comenzamos con la forma de diferencias finitas para la derivada de la velocidad

$$\frac{dv}{dt} \simeq \frac{v_{i+1} - v_i}{\Delta t} \tag{11}$$

donde asumimos que Δt es paso discreto pequeño, y v_i es la velocidad al tiempo $t_i \equiv i\Delta t$, por lo que de la ecuación (**??**)

$$v_{i+1} = v_i + \frac{P}{mv_i}\Delta t \tag{12}$$

Dada la velocidad en un tiempo i (es decir, v_i), podemos usar (**??**) para calcular un valor *aproximado* de la velocidad en el siguiente paso v_{i+1}.

Si conocemos la velocidad inicial v_0, podemos obtener v_1, v_2, y así sucecivamente.

Considerando la fricción del aire

La fuerza debida a la fricción puede aproximarse de manera inicial como

$$F_a \simeq -B_1 v - B_2 v^2 \tag{13}$$

Para velocidades muy bajas, el primer término es el que domina, y el coeficiente B_1 se puede calcular para objetos con formas sencillas.

Para una velocidad razonable v^2 el término domina sobre los demás, pero B_2 no puede calcularse exactamente en objetos sencillos como una pelota de beisbol, menos para una bicicleta.

Podemos aproximar el valor de B_2 como sigue:

Si un objeto se mueve a través de la atmósfera y debe empujar fuera del camino el aire delante de él. La masa de aire movido en el tiempo dt es $m_{aire} \sim \rho A v dt$, donde ρ es la densidad del aire y A el área frontal del objeto. A este aire se le da una velocidad de orden v, y por lo tanto, su energía cinética es $E_{aire} \sim m_{aire} v^2/2$

Este es también el trabajo realizado por la fuerza de arrastre (la fuerza sobre el objeto debido a la resistencia del aire) en el tiempo dt, por lo $F_a v dt = E_{aire}$. Poniendo todo esto junto nos encontramos

$$F_a \simeq -C\rho A v^2$$

Incluyendo este término en la expresión para la velocidad

$$v_{i+1} = v_i + \frac{P}{mv_i}\Delta t - \frac{C\rho A v_i^2}{m}\Delta t \qquad (14)$$

Ahora te toca implementar el código, considerando $C = 0.5$ y $A = 0.33$

Comparando velocidades