Tema 1: Python

Introducción

Computación Avanzada: uso de herramientas numéricas para la resolución de problemas de Física.

- Problemas que no pueden ser resueltos analítica ni experimentalmente.
- Problemas que pueden ser resueltos experimentalmente: comparación de los experimentos con las bases teóricas que los sustentan.

En algunos casos se requiere realizar cálculos analíticos antes de aplicar los métodos numéricos.

Ejemplos de uso de los ordenadores para realizar cálculos científicos:

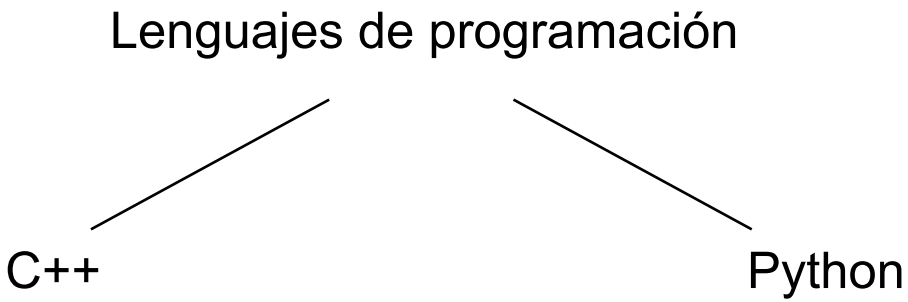
- Resolución de integrales complicadas.
- Resolución de ecuaciones diferenciales no lineales.
- Cálculos con matrices gigantescas.
- Etc.

Nunca hay una única manera de resolver un problema numéricamente.

Experiencia ⟶ estrategia más precisa o eficiente ⟶ “arte de programar”

Tema 1: Python

Introducción



Python

- Lenguaje intérprete: sencillo, de fácil comprensión y aprendizaje. Multiplataforma
- Extensible: existen multitud de paquetes con funciones que amplían las funcionalidades básicas de Python, muchas de ellas orientadas a problemas científicos y matemáticos.
- Gratuito y accesible (www.python.org)

C++

- Lenguaje de compilación: creación de un ejecutable en código máquina
- Más control de la memoria
- Mayor rapidez y rendimiento para cálculos exigentes

Tema 1: Python

Introducción

Lenguajes compilados y lenguajes interpretados

Un programa de ordenador que debe convertir el programa fuente escrito por el programador en un programa ejecutable que ejecuta el ordenador. Puede realizar esa tarea de dos formas distintas:

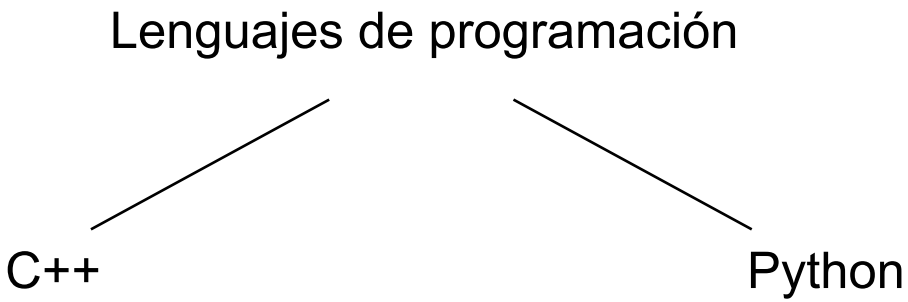
- en un **lenguaje compilado**, la tarea de conversión se realiza una sola vez. El compilador crea un fichero ejecutable a partir del código fuente del programa y a partir de entonces el usuario ejecuta el fichero ejecutable tantas veces como quiera, sin tener que volver a utilizar el código fuente.
- en un **lenguaje interpretado**, la tarea de conversión se realiza cada vez que se quiere ejecutar una instrucción del programa fuente, la convierte en código ejecutable, la ejecuta y pasa a la siguiente. En este caso, el usuario necesita tener el programa fuente para poder ejecutarlo.

Para un mismo programa fuente, la compilación produce programas ejecutables más rápidos que los programas interpretados, aunque la tendencia es a que esas diferencias se reduzcan.

También se debe tener en cuenta que las preferencias por compiladores o intérpretes pueden deberse a motivaciones no técnicas. Por ejemplo, el programador de software comercial preferirá el uso de lenguajes compilados para poder comercializar el programa ejecutable sin necesidad de hacer público el programa fuente, mientras que el programador de software libre no tiene problemas con los lenguajes interpretados puesto que va a proporcionar también el código fuente

Tema 1: Python

Introducción



Python

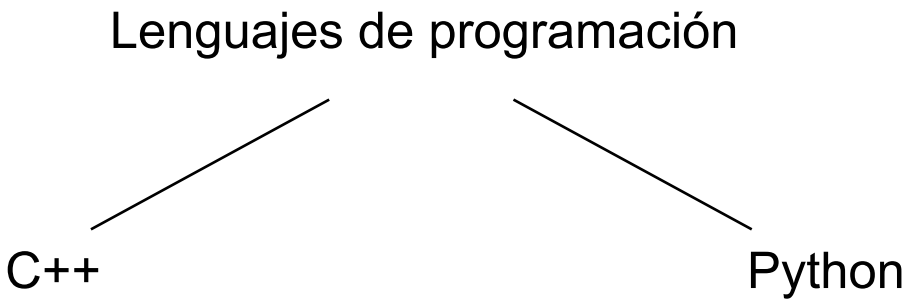
- Lenguaje intérprete: sencillo, de fácil comprensión y aprendizaje. Multiplataforma
- Extensible: existen multitud de paquetes con funciones que amplían las funcionalidades básicas de Python, muchas de ellas orientadas a problemas científicos y matemáticos.
- Gratuito y accesible (www.python.org)

C++

- Lenguaje de compilación: creación de un ejecutable en código máquina
- Más control de la memoria
- Mayor rapidez y rendimiento para cálculos exigentes

Tema 1: Python

Introducción



En definitiva

Rendimiento vs. Sencillez

Usaremos **Python 3**, que coexiste con Python 2. Existen diferencias significativas a nivel de sintaxis entre ambas versiones.

Tema 1: Python

1.1 Fundamentos de Python

- Los archivos con el código en Python deben tener la extensión .py
- Los comentarios van precedidos de “#”
- Para continuar un comando en la línea siguiente se utiliza “\”
- Nombres de variables:
  - Pueden tener cualquier longitud
  - Se forman a partir de letras, números y “_”
  - No pueden comenzar por número
  - Mayúsculas y minúsculas son diferentes
- Tipos:
  - Tipado dinámico: el propio lenguaje asigna los tipos de manera dinámica, durante la ejecución, no es necesario declarar las variables.
  - Una variable puede cambiar de tipo durante la ejecución del programa (no recomendable)
  - Tipado fuerte: intransigente con los tipos de variable.
  - Tipos más comunes: entero (int), real (float) –valores aproximados–, complejo (complex), cadena (str), boolean (bool)

- Existen funciones para cambiar el tipo de una variable: int(), round(), float(), complex(), o string(). type(variable) devuelve el tipo de la variable
- Las variables de texto van con comillas. Cuando se encierra texto entre tres comillas (3 de abrir y 3 de cerrar) se pueden usar saltos de línea.
- Los valores booleanos son con la primera letra mayúscula: True, False

Entrada/salida

- Salida:
  - *print(..., ..., ...).* Puede tener varios argumentos separados por ","
  - *print(..., ..., ..., sep = "x"),* para indicar que se utilice el separador "x" a la salida (por defecto es el espacio).
  - Para eliminar el separador, *sep=""*
- Entrada:
  - *input()* o *input("Introduce un número: ").* El programa espera la entrada
  - Importante: Cualquier entrada es una cadena ("string").
  - Para convertir la cadena al tipo deseado: x = int(input()), la convierte en un número entero, y = float(input()), en un número real.

Tema 1: Python

1.1 Fundamentos de Python

Operadores

- Simples: +, -, *, /. Potencias: **. División entera: //. Resto (modulo): %. Precedencia: como en álgebra.
- // es la división entera y trunca el resultado al menor entero más cercano
- El resultado de la división normal es siempre un número real (float) o complejo, aunque ambos números sean enteros (comportamiento diferente a C++ y otros lenguajes).
- % no es válida para complejos
- El tipo resultante de una operación es siempre el más general de los tipos participantes, excepto para "/" cuyo resultado es flotante o complejo.

Asignación

- "=" Operador asignación, el valor de la derecha se calcula y asigna a la variable de la izquierda; x = a + 5
- En Python es posible la múltiple asignación: x, y = 2, 5 ⟶ x = 2, y = 5. (Caso importante, intercambio de valores: x, y = y, x)
- También a=b=c=4.0. En una sola línea asigna el valor 4.0 a las 3 variables

Tema 1: Python

1.1 Fundamentos de Python

Paquetes

- Colección de funciones y constantes.
- Uno de los paquetes más usados es "math", ya que contiene funciones matemáticas no contenidas en el lenguaje Python, como *log*, *log10*, *exp*, *sin*, *cos*, *tan*, *sqrt*, etc.
- También incluye constantes como los números *pi*, y *e*.
- El paquete "math" viene incluido en la instalación estándar de Python. Otros paquetes hay que instalarlos específicamente.
- Además de "math" usaremos los paquetes "numpy", "matplotlib" y "vpython"
- Para usar una función o una constante de un paquete se usa, por ejemplo:

  from math import log, pi

  Esto carga la función logaritmo natural y la constante "pi" en la memoria.
- Se puede cargar el paquete completo con "*from math import * ",* aunque puede dar problemas si hay una función o constante con el mismo nombre en dos paquetes distintos.

Tema 1: Python

1.1 Fundamentos de Python

Listas

- En Python, una lista permite guardar valores de distinto tipo, no como un array en C++, por ejemplo.
- Otra diferencia es que el tamaño no es fijo, las listas se pueden expandir dinámicamente.

  Nombrelista=[elem1, elem2, elem3,…]

  Ejemplo: r = [1, ..., 1 +2j, ..., “cadena”, [1, 2, 3]]
- Se pueden usar variables siempre que ya tengan valores asignados:

  r = [x, y, z, ...] válido si x, y, z, ... ya tienen valores
- Como en C++, la numeración de los elementos de la lista comienza en 0.

Nombrelista[2]	indica el tercer elemento de la lista
Nombrelista[-2]	indica el penúltimo elemento de la lista. El último sería el [-1]
Nombrelista[:]	indica todo el rango de la lista
Nombrelista[n:m]	subconjunto entre n y m, pero el m no entra.

- En Python todo lo que se crea se considera un objeto, incluso las variables

Listas *(continuación)*

- Para añadir un elemento al final de una lista, se usa

  Nombrelista.append(Nuevo_elemento)
- Si en lugar de eso queremos insertarlo en algún lugar intermedio (n):

  Nombrelista.insert(n,Nuevo_elemento)
- Si queremos añadir varios elementos al final de una lista

  Nombrelista.extend([Nuevos elementos])
- Si queremos saber qué índice tiene un elemento determinado:

  Nombrelista.index(Elemento)

  (En caso de que el elemento esté repetido en varios sitios, nos devuelve el índice de la primera vez que aparece)
- Para averiguar si un elemento está en la lista:

  print(elemento in Nombrelista)

  imprime True si el elemento está en la lista y False si no está.
- Para eliminar un elemento de una lista

  Nombrelista.remove(Elemento)

Tema 1: Python

1.1 Fundamentos de Python

- Para eliminar el último elemento de una lista

  Nombrelista.pop()

- La suma de dos listas produce una concatenación. Y el producto por un número produce la repetición de la lista tantas veces como el número.
- sum(nombrelista): suma todos los elementos de una lista.
- min(nombrelista): da el mínimo valor de los elementos de una lista.
- max(nombrelista): da el máximo valor de los elementos de una lista.
- len(nombrelista): da el número de elementos que contiene la lista.
- Para aplicar una operación o una función a todos los elementos de una lista se utiliza el comando map:

  map(operación, nombrelista)

  por ejemplo, para hacer la raíz cuadrada de todos los elementos de una lista: map(sqrt, nombrelista)
- Para crear una nueva lista con los resultados de la operación se usa list:

  Nombre_nuevalista=list(map(sqrt, nombrelista))
- Se puede crear una lista vacía con “Nombrelista=[]" y luego ir añadiendo elementos con Nombrelista.append.

Tema 1: Python

1.1 Fundamentos de Python

Arrays *(numpy)*

- En python también existen los arrays, que son análogos a los de C++: el número de elementos es fijo, y todos los elementos han de ser del mismo tipo.
- Se necesita el paquete numpy para usar arrays. Su fomato es

  [elem1 elem2 elem3…] (sin comas)

- La manera más directa de crear un array con valores iniciales es a partir de una lista, con la función de numpy "array":

```
import numpy as np
a=[1, 2, 3, 4, 5]
b=np.array(b, dtype=int)          (se puede omitir "dtype")
```

- Otra posibilidad es crear un array con ceros, usando la función zeros de numpy. Por ejemplo, para crear un array de 4 elementos de tipo float con ceros:

```
Nombre_array=np.zeros(4,float)
```

- Posteriormente se pueden dar valores a los elementos que queramos hacer distintos de cero recorriendo los índices correspondientes.

Arrays *(continuación)*

- Además de la función zeros, el paquete numpy también contiene la función "ones", que crea un array con unos.
- Hay una tercera opción para crear un array, con el comando empty:

  Nombre_array=np.empty(4,float)

- Si se crea un array de elementos enteros a partir de una lista de elementos reales, estos elementos se truncan.
- Si se incluyen dos listas, se crea un array bidimensional:

  Nombre_array=np.array([[1,2,3],[4,5,6]],int),

  crea el array
  ```
  [[1  2  3]
   [4  5  6]]
  ```
  esta disposición se conoce como lista de listas, y el array se crea por filas.
- Ventajas de los arrays frente a las listas:

  1) los arrays pueden ser de varias dimensiones, las listas sólo de una.
  2) se pueden hacer operaciones aritméticas con arrays y no con listas.
  3) operar con ellas es mucho más sencillo.

Tema 1: Python

1.1 Fundamentos de Python

Arrays *(continuación)*

- Un elemento individual de un array se expresa: nombre_array[n], como en una lista. Si es de dos dimensiones, el primer índice es la fila y el segundo la columna.
- Se puede transformar un array en una lista usando el comando list:

  Nombre_lista=list(Nombre_array)

- Se puede crear un array a partir de una columna de datos en un fichero. Para ello se utiliza la función de numpy **loadtxt**.

  Nombre_array=np.loadtxt("fichero.dat",float)

- El comando es bastante poderoso: si el fichero contiene m filas con n columnas cada una, al ejecutar el comando anterior se crea un array bidimensional de m filas y n columnas.
- Algunas funciones o métodos que se aplican a los arrays son *ndim*, que da el número de dimensiones, *shape*, que da el número de elementos de cada dimensión, o *size*, que da el número total de elementos del array.
- Una función muy útil es la de transposición: *T*. Por ejemplo, si el array *a1* es una matriz de 3x4 elementos, *a1.T* es su traspuesta, de 4x3 elementos.

Tema 1: Python

1.1 Fundamentos de Python

Arrays *(continuación)*

- Cualquier operación aritmética que se aplica sobre un array, se ejecuta sobre todos los elementos de dicho array.
- Si sumamos dos arrays se suman sus elementos de 2 en 2 (diferente a las listas, que se concatenan). Lo mismo si las multiplicamos. Para obtener el producto escalar se usa la función dot.

  Resultado=dot(array_1,array_2)
- El comando dot es muy poderoso, porque aplicado a matrices (arrays bidimensionales) ejecuta el producto matricial.
- Cuando multiplica un vector (v) por una matriz (m), en el producto v·m considera a v como vector columna, y en el producto m·v lo considera como vector fila.
- Las mismas funciones que vimos para las listas, funcionan para arrays, como sum, max, min, o len. También map, aunque el resultado es una lista y posteriormente hay que convertirla a array. Por ejemplo:

  Nombre_array2=array(list(map(sqrt,Nombre_array1)),float)
- En el caso de arrays bidimensionales es mejor usar size y shape, que len.

Tema 1: Python

1.1 Fundamentos de Python

Arrays *(continuación)*

- Cómo crear un array que es copia de otro array: Esto es un poco contraintuitivo, porque la sentencia

  Nombre_array_2 = Nombre_array_1

  no crea una nueva array que sea copia de Nombre_array_1, sino que crea un segundo nombre de la primera array. Si se modifica algún valor en una de ellas se modifica en la otra.
- Para crear una copia de un array se usa la función de numpy copy:

  np.copy(nombre_array)
- En el ejemplo anterior se pondría:

  Nombre_array_2 = np.copy(Nombre_array_1)

Otras estructuras

- Además de las listas y los arrays existen otras estructuras, como las tuplas o los diccionarios. Las primeras llevan sus elementos encerrados entre paréntesis y separados por comas – (elem1, elem2,…) –. Los diccionarios están formados por pares de elementos encerrados entre llaves, uno de ellos funciona como clave – {elem1:clave1, elem2:clave2, …}

Tema 1: Python

1.1 Fundamentos de Python

Condicionales

```
if expresión_lógica:                    (ojo a los dos puntos!)
        instrucciones
continúa el programa
```

- La expresión lógica va sin paréntesis ni otros elementos.
- Las instrucciones del if se distinguen del resto porque van **indentadas (!)**
- Si usamos un else, y/o un elif:

```
if expresión_lógica:
        instrucciones
elif expresión_lógica:
        instrucciones
else:
        instrucciones
continúa el programa
```

Comparadores, como en C++:

==, <, >, <=, >=, !=

- Se pueden concatenar varios operadores dentro de una misma expresión lógica. Por ejemplo: *if 0<variable<100*, *variable1<variable2<variable3<.....*
- Operadores lógicos "and" y "or": se usan normalmente, sin paréntesis ni nada: *if condicion1 and condicion2*

Tema 1: Python

1.1 Fundamentos de Python

Bucles

While

- En Python no hay bucle do-while, sólo while:

```
while condición:                (notar los dos puntos)
        instrucciones
continúa el programa
```

- Se puede salir de un bucle while mediante el comando *break*.
- Otras maneras de modificar la ejecución de bucles while es usando:
  - *continue*: se salta la ejecución actual y pasa a la siguiente.
  - *else*: las instrucciones que siguen se ejecutan una vez que ha terminado el bucle. Puede ser útil para restablecer variables.

For

```
for variable in elemento a recorrer (lista, tupla, etc):
        instrucciones
continúa el programa
```

- Los elementos de la lista pueden ser de tipos distintos. Si ponemos una palabra, la variable recorre cada letra de la palabra.

Bucles

For

- Por ejemplo:

```
for i in [3.14, "Juan", 21]:
        instrucciones
```

  ejecuta las instrucciones 3 veces.

- En el caso de una palabra:

```
for i in "Pepito":
        instrucciones
```

  ejecuta las instrucciones 6 veces, las letras que tiene Pepito.

- Lo más conveniente es usar range(n), que crea una lista de n elementos, que son números enteros que comienzan en 0 y terminan en n-1. Por ejemplo:

```
for i in range(5):
        instrucciones
```

  ejecuta las instrucciones 5 veces. range(5) toma los valores 0,1,2,3,4

Tema 1: Python

1.1 Fundamentos de Python

Bucles

For

- range(5) no genera una lista, genera un "iterador".
- Se puede generar un array con el comando arange(n) de numpy. En este caso, los elementos del array pueden ser números reales, no como en range() que son enteros.
- También puede usarse linspace(inicio,fin,num_puntos) que divide linealmente el espacio comprendido entre inicio y fin (incluído) en num_puntos puntos de tipo real.
- Al utilizar range() en un bucle for, se puede hacer simple, es decir, range(n) va desde 0 hasta n-1, pero también se puede modificar la forma en que se recorre el rango. Por ejemplo:

  range(5,10) va desde 5 hasta 9 incluidos.

  range(5,50,3) va desde 5 hasta 49 de 3 en 3.

Tema 1: Python

1.1 Fundamentos de Python

Funciones

```
def nombre_funcion([argumentos]):
        Instrucciones
        return (opcional)
```

- Indentación esencial porque python interpreta lo indentado como perteneciente a la función. Para terminar la función simplemente se elimina la indentación.
- Las indentaciones en Python pueden tener cualquier longitud a partir de un espacio. Lo estándar es insertar cuatro espacios en cada indentación.
- Ejecución de una función:

```
nombre_función([argumentos])
```

- En el comando "return" se pueden devolver valores de cualquier tipo, incluyendo listas o arrays

```
def nombre_función(argumentos)
        .....
        nombre_lista=[x,y]
        return nombre_lista
```

Funciones

- También podría devolverse un array:

  return array([x,y],float)

- O incluso un grupo de valores separados por comas para luego usar la asignación múltiple de Python:

  return x,y

  y luego en el programa principal:

  a,b=nombre_función(argumentos)

  que asigna x->a, y->b.

- Puede haber funciones sin argumentos, y también puede haber funciones sin la línea de "return", que funcionan como una subrutina.
- Las funciones pueden estar en cualquier momento del código, pero siempre antes de ser llamadas. Es buena práctica poner todas al principio.
- Una vez definida una función se puede usar en cualquier parte, como dentro de un comando print, o incluso servir como argumento de otras funciones. También se pueden usar como argumento del comando map.

Funciones

- Se pueden incluir las funciones en un fichero externo, por ejemplo "misfunciones.py". Luego para incluir estas funciones en el código hay que poner:

```
from misfunciones import funcion_concreta
```

  o bien

```
from misfunciones import *
```

- Se puede incluso meter una función dentro de sí misma y hacer un código recursivo. Por ejemplo, una forma de calcular el factorial de un número.

```
def factorial(n)
        if n==1:
                return 1
        else:
                return n*factorial(n-1)
```

Tema 1: Python

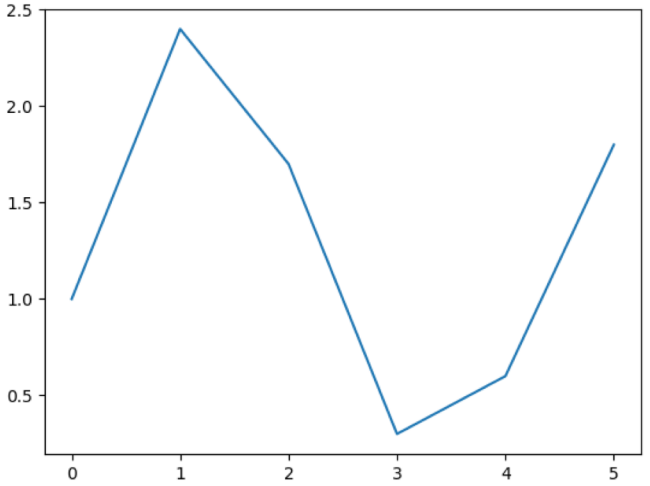
1.2 Gráficos con Python

- Se pueden crear y visualizar gráficos desde Python. Existen varios paquetes, aquí vamos a ver “matplotlib”. Este paquete emula el entorno de matlab para gráficos. Se puede obtener información en *matplotlib.org*
- Dentro del paquete matplotlib usaremos el módulo pyplot, que se carga habitualmente de la siguiente manera:

```
import matplotlib.pyplot as plt
```

- Para crear un gráfico hay que usar al menos dos comandos: plot() y show(). El primero debe encerrar al menos una lista o array, el segundo puede ir vacío.
- Si se incluye sólo un array o lista, ésta se representa frente a una secuencia de números enteros comenzando por cero.

```
import matplotlib.pyplot as plt
y=[1.0, 2.4, 1.7, 0.3, 0.6, 1.8]
plt.plot(y)
plt.show()
```

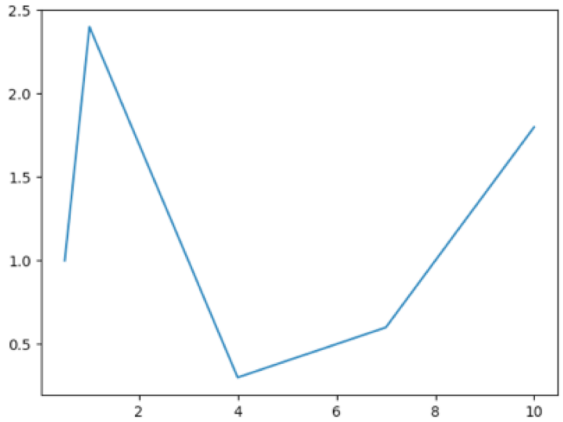


Tema 1: Python

1.2 Gráficos con Python

- Si se incluyen dos arrays o listas se representa la segunda frente a la primera:

```
import matplotlib.pyplot as plt
x=[0.5, 1.0, 2.0, 4.0, 7.0, 10.0]
y=[1.0, 2.4, 1.7, 0.3, 0.6, 1.8]
plt.plot(x,y)
plt.show()
```

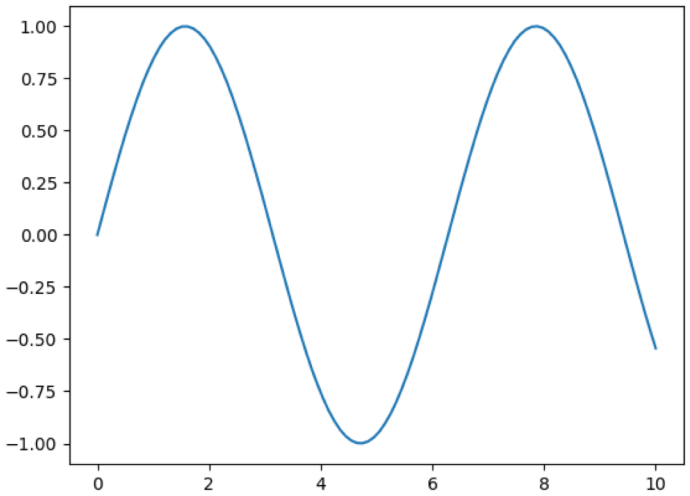


- Resultan útiles algunas funciones del paquete “numpy”. Por ejemplo, para crear un rango de valores de x se puede usar linspace(inicio,fin,nº puntos). También se puede utilizar arange().
- Las funciones de numpy, que incluyen funciones trigonométricas, logarítmicas, etc., tienen la ventaja de que se aplican a todos los elementos de un array, facilitando la creación de funciones para ser representadas.
- El siguiente código representa f(x)=sin(x) entre 0 y 10 radianes usando 100 puntos:

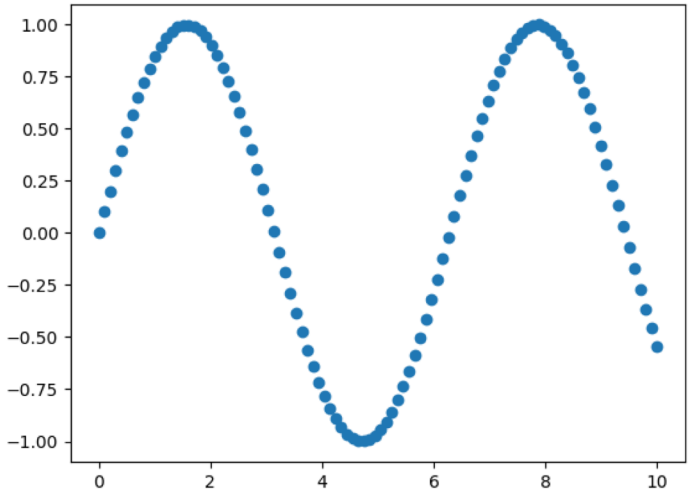
Tema 1: Python

1.2 Gráficos con Python

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(0,10,100)
y=np.sin(x)
plt.plot(x,y)
plt.show()
```



- Si en lugar de plot(x,y) ponemos plot(x,y,'o'), la curva se representa con círculos:



Tema 1: Python

1.2 Gráficos con Python

Representación de datos de un fichero

- El paquete numpy contiene una función que permite importar datos de un fichero: loadtxt. Esta función lee filas de un fichero y crea un array de 2 dimensiones, con n x m elementos, siendo n las filas y m las columnas.
- Por defecto ignora las filas que comienzan con "#".
- Una vez cargados los datos hay que asignar cada columna a una variable, que va a ser un array. Por ejemplo, si la primera columna es la x y la segunda la y:

```
Nombre_array=np.loadtxt(“nombrefichero”, tipo)
x=Nombre_array[:,0]
y=Nombre_array[:,1]
plt.plot(x,y)
plt.show()
```

- En realidad se podría poner directamente

```
Nombre_array=np.loadtxt(“nombrefichero”, tipo)
plt.plot(Nombre_array[:,0],Nombre_arrray[:,1])
plt.show()
```

1.2 Gráficos con Python

- El comando show() detiene el programa. Esto quiere decir que no se continúa la ejecución hasta que se cierra la ventana gráfica. Es un comportamiento parecido al comando input(). Ambas se denominan funciones bloqueantes.

Ejes, títulos y leyendas

- Se pueden hacer invisibles los ejes, con plt.axis(off).
- Para cambiar los rangos de los ejes x e y hay dos opciones. Una opción es utilizar plt.axis, colocando los límites de manera ordenada en forma de lista, primero los del eje x y luego los del eje y: plt.axis([0,2,0,5]). En este caso, el eje X va desde 0 hasta 2, y el eje Y desde 0 hasta 5.
- También se puede utilizar las funciones xlim e ylim. Los límites anteriores se pondrían así:

```
plt.xlim([0,2])
plt.ylim([0,5])
```

- Se puede hacer visible una rejilla en el gráfico mediante la función plt.grid(True).

1.2 Gráficos con Python

- Para añadir títulos en los ejes (labels): se hace con plt.xlabel("texto"), plt.ylabel("texto"). Se puede poner también un título en la parte superior del gráfico con plt.title("Texto").
- Importante: todos los modificadores del gráfico, como xlim, xlabel, o title, deben ir después de plot() y antes de show()

Representar más de una curva en el mismo gráfico

- Para meter varias curvas se usa un comando plot para cada una de ellas. Con ello se van acumulando y en el momento en que se pone el comando show() se muestran todos los comandos plot que se han acumulado. Si no se han asignado colores a las curvas, cada una sale con un color distinto preasignado.
- En estos casos es útil añadir una leyenda para mostrar qué se representa en cada curva. Para ello se añade una opción en el comando plot con el texto que se quiera añadir a la leyenda:

```
plt.plot(x,y, label='datos')
```

  y cuando se han puesto todos los comandos plot se llama a la función plt.legend(), que es la que crea la leyenda.

Tema 1: Python

1.2 Gráficos con Python

Tipos de línea y símbolo

- Para elegir el tipo de línea se añade un tercer argumento al comando plot, de tipo texto.
- Los siguientes caracteres determinan el tipo de línea o símbolo:

  - línea sólida
  -- línea discontinua
  -. línea discontinua raya-punto
  : línea punteada
  . punto (se puede usar también la función scatter(x,y) en lugar de plot())
  , pixel
  o círculo
  s cuadrado

  (hay muchos más símbolos, se pueden mirar en https://matplotlib.org/3.1.1/api/markers_api.html)

- Dicho texto puede también modificar el color con el que se representa la curva: r, g, b, c, m, y, k, w, que corresponden a rojo, verde, azul, cian, magenta, amarillo, negro y blanco.

Tema 1: Python

1.2 Gráficos con Python

Tipos de línea y símbolo

- Se pueden combinar el color y el tipo de línea o símbolo. En este caso, es necesario poner primero el color y después el tipo de letra. Por ejemplo:

```
plt.plot(x, y, 'mo')
plt.plot(x, y+1 , 'g-.')
```

- Todas estas opciones de la función plot se pueden poner por separado, con su campo correspondiente. Por ejemplo:

```
plt.plot(x, y, color="g", linestyle="--", linewidth=1.5, marker='o',
markerfacecolor="b", markeredgecolor="k", markeredgewidth=1.5,
markersize=5)
```

Otras opciones

- Para crear una función y representarla se puede hacer uso de un bucle for implícito:

```
x=np.arange(10)
plt.plot(x, [y**2 for y in x])
```

- Además de poder guardarse la figura que se cree mediante el botón interactivo, se puede guardar mediante plt.savefig('nombre.extensión')

Otras opciones

- El formato en el que se guarde la figura depende de la extensión que se ponga.
- Se puede controlar lo que aparece en los ticks de los ejes. Incluso se puede usar una serie no numérica en lugar de valores numéricos. Para ello hay que usar xticks o yticks

```
x = np.arange(10)
y=x**2
plt.plot(x, y, 'o--')
plt.xticks(range(len(x)), ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'])
plt.yticks(range(0, 100, 10))
plt.show()
```

- La llamada a xticks dice que hay que crear 10 ticks (len(x)) y a continuación los detalla mediante una lista.
- La llamada a yticks dice que los ticks deben ir desde 0 a 100 (no incluido), separados una distancia de 10 unidades.

Tema 1: Python

1.2 Gráficos con Python

- Matplotlib tiene muchas más posibilidades. Por ejemplo, permite trabajar con objetos. A continuación se muestra un ejemplo que muestra otras maneras de usar este potente paquete:

```
import matplotlib.pyplot as plt
import numpy as np
x1, x2 = np.linspace(0.0, 2.0, 20), np.linspace(0.0, 2.0, 200)
y1, y2, y3 = np.exp(-x1), np.exp(-x2), np.sin(2 * np.pi * x2)
y4 = y2 * y3
l1 = plt.plot(x1, y1, "bD--", markersize=5)
l3 = plt.plot(x2, y3, "go-", markersize=5)
l4 = plt.plot(x2, y4, "rs-", markersize=5)
plt.ylim(-1.1, 1.1)
plt.xlabel("Segundos")
plt.ylabel("Voltios")
plt.legend( (l3[0], l4[0]), ("Oscilatorio", "Amortiguado"), shadow = True )
plt.title("Movimiento Oscilatorio Amortiguado")
plt.show()
```

Se pueden crear objetos de tipo "plot" para luego referirse a ellos

Tema 1: Python

1.2 Gráficos con Python

- Dos posibilidades para usar matplotlib:
- Método directo (usando la API pyplot)

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2, 100)
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
```

- Método orientado a objetos

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2, 100)
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend()  # Add a legend.
```

Tema 1: Python

1.2 Gráficos con Python

Gráficos de densidad

- Se usan para representar datos tridimensionales con la tercera dimensión como una escala de color.
- Normalmente se leen los datos de un fichero con loadtxt. El fichero contiene una matriz bidimensional con valores que indican una magnitud.
- La función que crea el gráfico de densidad es imshow(array) y hay que seguirla de show(). Ejemplo

```
import numpy as np
import matplotlib.pyplot as plt
NombreArray=np.loadtxt("NombreFichero",float)
plt.imshow(NombreArray)
plt.show()
```

- Los ejes muestran el índice de la correspondiente fila y columna de la matriz. El origen está arriba a la izquierda. El origen se puede cambiar, por ejemplo al extremo inferior, metiendo un segundo parámetro: imshow(data, origin="lower")

Tema 1: Python

1.2 Gráficos con Python

Gráficos de densidad

- Se puede cambiar la escala de los ejes mediante un tercer parámetro: imshow(data, origin="lower", extent=[LimInfX, LimSupX, LimInfY, LimSupY]). Notar que lo que sigue a extent es una lista.
- Para cambiar el esquema de colores se pueden utilizar varias funciones, como por ejemplo gray(), que usa escalas de grises. Hay otras posibilidades que pueden consultarse en internet.
- La función colorbar() añade una escala de colores con los correspondientes valores numéricos.
- Para cambiar el factor de forma del gráfico (ancho y largo) se puede utilizar el parámetro aspect(factor), que aplica el factor correspondiente al eje-y a la hora de representar los datos.
- Si se desea representar sólo una porción de los datos, se puede usar xlim o ylim, con los extremos que queremos que encierren el gráfico. Estos extremos se ponen en unidades de índices matriciales si no se ha usado extent, y si se ha usado, con la magnitud correspondiente.

Tema 1: Python

1.2 Gráficos con Python

Imágenes tridimensionales

- Se pueden crear objetos y animaciones 3D gracias al paquete "vpython". Este paquete crea objetos 3D y los representa en una ventana del navegador de internet por defecto. Por ejemplo, uno de los objetos es sphere( ). Entre los paréntesis se pueden poner varias opciones, como la posición, el color (como un vector RGB), el radio, etc.
- Para definir coordenadas, el paquete vpython incluye un objeto vector, cuyo formato es vector(coord_x, coord_y, coord_z).

```
from vpython import *
sphere(radius=0.5, pos=vector(0.5, -1.0, 0.0), color=vector(1,1,1))
```

- Se pueden crear varios objetos de tipo esfera y darles a cada uno un nombre. Por ejemplo:

```
s1=sphere();    s2=sphere()
```

- Una vez creado se pueden modificar sus propiedades, incluyendo su posición lo que permite crear movimiento.

```
s1.pos = vector(x,y,z)
s1.radius=valor
```

Tema 1: Python

1.2 Gráficos con Python

Gráficos tridimensionales

- Se pueden crear listas o arrays de estos objetos. Por ejemplo, usando numpy:

```
NombreArray = np.empty(10, sphere)
for n in range(10):
        NombreArray[n] = sphere()
```

  crea un array de 10 esferas, cada una se puede identificar por NombreArray[n].

- Otros objetos que se pueden crear son cajas, cilindros, conos, pirámides y flechas. Cada uno permite ciertos parámetros.
- Se pueden controlar las propiedades de la ventana donde se crean los objetos mediante canvas(). Una propiedad interesante es desactivar la autescala.
- Para crear movimiento está la función rate(valor), que ralentiza la ejecución de cada paso en un intervalo de 1/valor segundos.
- (Cuidado!! numpy tiene también una función rate(). Si se usa "from numpy import *" puede haber problemas con la de vpython.)