Taller de Python

Segundo encuentro: Numérico

FIFA

Federación Interestudiantil de Física Argentina

24 de octubre de 2014

¿Por qué Python y no X?

Python es un *lenguaje de programación*:

- Libre
- Expresivo
- Mucho soporte de la comunidad
- Divertido.

Además, a diferencia de otros lenguajes es *interactivo* (si uno sabe usarlo)

¿Qué necesito para programar Python?

Como todo lenguaje, la computadora y vos necesitan un diccionario, lo que se llama *compilador*. Sobre ese compilador, en general se distribuye parte de programas hecho por otras personas, que en conjuto se llaman *librerías*.

¿Qué necesito para programar Python?

Para eso, podemos instalar Anaconda, que instala todos los programas que vamos a usar (compilador) y librerías. Además, instala algunos programas para ayudarnos a escribir Python, denominados IDE. Nosotros vamos a usar el programa *Spyder*.

¿Qué necesito para programar Python?

Páginas de los programas necesarios

- Anaconda `http://continuum.io/downloads#py34`
- Miniconda (Anaconda sin librerías, ni programas) `http://conda.pydata.org/miniconda.html`
- Spyder `http://goo.gl/VByA2t`

¿Qué necesito para programar Python?

Comandos útiles de teclado de Spyder

- F5: Ejecutar archivo editado
- Shift + Ctrl + E: Ir a editor
- Shift + Ctrl + C: Ir a consola
- Tab: autocompletar (en consola también!!)

Cálculo numérico... ¿y eso?

En esta época donde las computadoras son ubicuas, la ciencia avanza sobre el cálculo numérico:

- Realizar y analizar mediciones antes imposibles (imaginen el LHC sin computadoras!!);
- Prototipando la realidad con simulaciones.

Obviamente que esta el argumento básico, resolver problemas puros numéricos, pero queremos tener una aplicación para aprender a usarla.

Objetivos del cálculo numérico

El objetivo del cálculo numérico:

- Determinar diferentes algoritmos para el mismo problema;
- Determina la cantidad de pasos a realizar, por lo tanto la velocidad al ejecutarse;
- Determinar el error que se comete al efectuar una cantidad determinada de iteraciones.

Problemas del cálculo numérico

Como la computadora *no* tiene infinita memoria, *no existe* forma de representar los números reales.

Para eso existen los números de punto flotante (llamados float), tienen precisión y cota. Prueben sumar 0,1 y 0,2

El ejemplo anterior es claro: hay que tener cuidado con la aritmética de floats, puede dar errores groseros al sumar a un número grande uno muy pequeño, o al dividir por algún número muy pequeño, o el error intrínseco de la representación.

Problemas del cálculo numérico

Como la computadora *no* tiene infinita memoria, *no existe* forma de representar los números reales.

Para eso existen los números de punto flotante (llamados float), tienen precisión y cota. Prueben sumar 0,1 y 0,2

El ejemplo anterior es claro: hay que tener cuidado con la aritmética de floats, puede dar errores groseros al sumar a un número grande uno muy pequeño, o al dividir por algún número muy pequeño, o el error intrínseco de la representación.

Problemas del cálculo numérico

Como la computadora *no* tiene infinita memoria, *no existe* forma de representar los números reales.
Para eso existen los números de punto flotante (llamados float), tienen precisión y cota. Prueben sumar 0,1 y 0,2

El ejemplo anterior es claro: hay que tener cuidado con la aritmética de floats, puede dar errores groseros al sumar a un número grande uno muy pequeño, o al dividir por algún número muy pequeño, o el error intrínseco de la representación.

Problemas del cálculo numérico II

Y en general para comparar floats se debe tomar una cota de error plausible, es decir

```
In [2]: a = 0.1; b = 0.10000001
In [3]: if a - b < 0.00001:
.... print(True)
True
```

Numpy, ¿qué es eso?

Numpy es la librería básica para el análisis numérico. Antes de ver de qué se trata, hagamos una prueba de fuego.
Para eso probemos ingresando lo siguiente en la consola

```
In [1]: import numpy as np
In [2]: A = np.array([[1,2],[3,4]])
In [3]: A
Out [1]: array([1,2]
               [3,4])
In [4]: A[1,1]
Out [2]: 4
```

Ventajas de Numpy

- Implementado sobre lenguajes de bajo nivel (C/C++, Fortran)
- Es *muy* rápido (más que Matlab)
- Tiene casi toda la funcionalidad que se usa en los ambientes científicos y tecnológicos.

Python: Calculadora avanzada

Hagamos algunas cuentas con Numpy

```
In [1]: 5 * 5
Out [1]: 15
In [2]: np.sin(np.pi/2)
Out [2]: 1
In [3]: np.exp(2)
Out [3]: 7.3890560989306504
```

Cálculo con matrices

Prueben lo siguiente

```
In [5]: B = np.array([[3,1,2],[1,4,1],[2,1,5]])
In [6]: import numpy.linalg as lin
In [7]: lin.eig(B)
Out[3]:
(array([ 6.89510652,  1.70759841,  3.39729507]),
 array([[ 0.49725362,  0.86427949,  0.07589338],
        [ 0.43170413, -0.17059871, -0.88573564],
        [ 0.75257583, -0.47319874,  0.45794385]]))
```

Eig de eigenvalues/eigenvectors, autovalores. Todos sabemos que son cosas útiles.

Cálculo con matrices

Solucionemos un sistema lineal $Bx = b$.
Acordate que todavía tenés definida B, así que definamos b:

```
In [8]: b = np.array([2,0,1])
In [9]: lin.solve(B,b)
Out[4]: array([ 0.775, -0.175, -0.075])
```

Para las diapositivas que vamos a ver a continuación tenemos que ejecutar previamente el siguiente comando

```
import numpy as np
import scipy as sp
import scipy.optimize as opt
import matplotlib.pyplot as plt
```

que representan los paquetes que vamos a utilizar.
Y como ya estamos más grandes, trabajemos directamente en el editor, y con F5 ejecutemos el código escrito en consola.
Así se trabaja generalmente

Obtención de raíces

Ahora vamos a encontrar la raíz de una $f(x)$. Primero debemos definirla. Usemos la definición *inline*

```
f = lambda x: np.log(x) - 1
x0 = 3
print(opt.newton(f, x0))
#2.71828182846
```

Ajustes de datos

Primero creamos los datos. *En general* se obtienen de una experiencia, pero como ejemplo usamos un algoritmo aleatorio para obtenerlos.
Para eso definamos una función que determina el modelo

```
def f(p,x):
return p[0]*x/np.sqrt(p[1]*x**2 +
       (x**2 - p[2]**2)**2)
```

Ajustes de datos

Primero creamos los datos. *En general* se obtienen de una experiencia, pero como ejemplo usamos un algoritmo aleatorio para obtenerlos.

```
x = np.linspace(0,5,70)
y = f([2,4,2],x) + np.random.random(x.size)*0.05 - 0.025
np.savetxt("datos.csv",np.array([x,y]).T)
```

De esta forma en el archivo datos.csv está la tabla de datos

Ajustes de datos

Ahora creamos una variable data con los datos guardados en datos.csv. También creamos los errores dx y dy con un algoritmo.

```
data = np.loadtxt("datos.csv")
dx = np.ones_like(data[:,0])*0.02
dy = np.ones_like(data[:,1])*0.05
```

Ajustes de datos

Finalmente ajustamos con la librería scipy.odrpack, que tiene una función especializada en Regresión ortogonal (con errores en ambas variables)

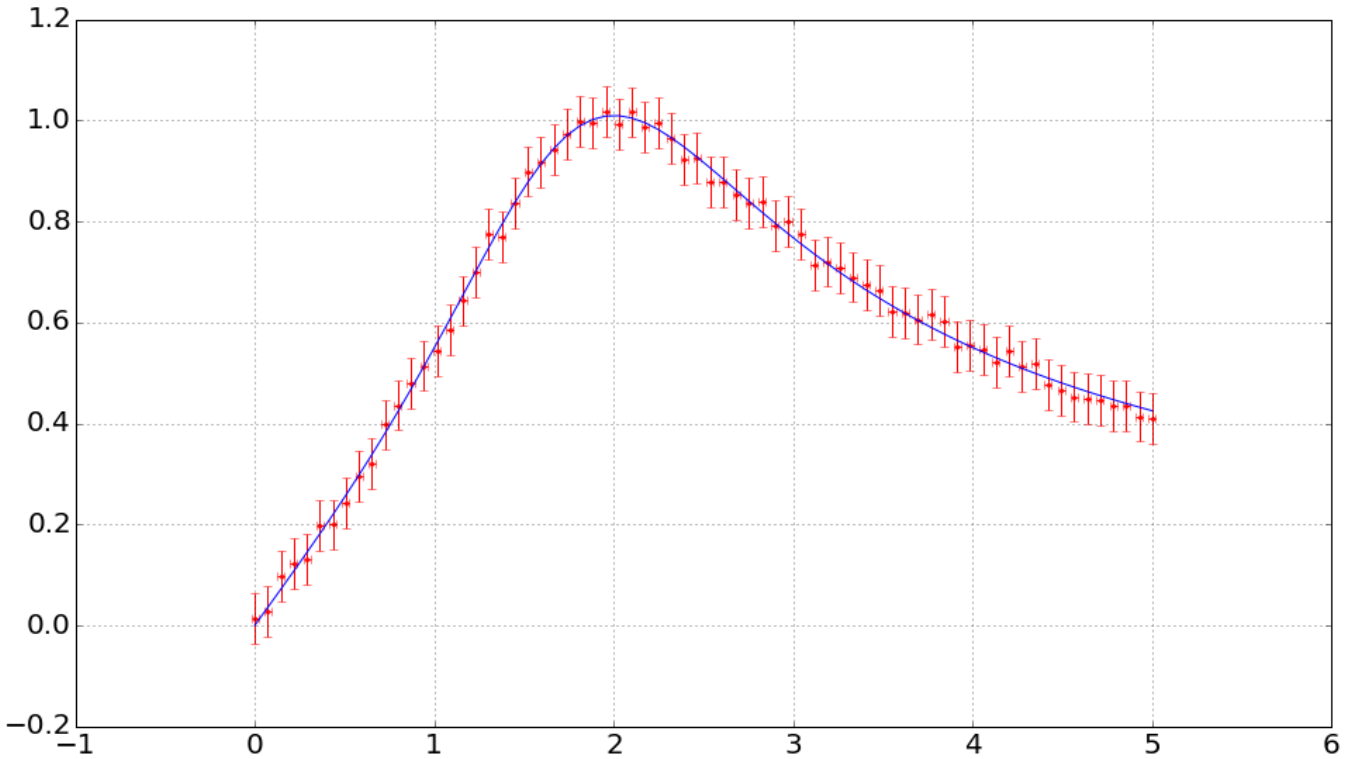
```
beta0 = [1,1,1] #Parametros iniciales
out = sp.odr.odrpack.odr(f, beta0, y, x,
                         full_output = 1)
print(out[0],out[1],out[3]['sum_square'])
```

Visualización de datos

Ahora que tenemos todos los datos analizados, queremos visualizarlos. Para eso usamos la librería matplotlib, con su módulo pyplot. Este módulo nos permite usar funciones simples para imprimir gráficos

```
import matplotlib.pyplot as plt
plt.errorbar(x,y,dy,dx,'.r')
plt.plot(x,f(out[0],x))
plt.show()
```

Visualización de datos



Integración de ODE

Para resolver una ecuación diferencial ordinaria (ODE), primero nos conviene definir la función $f(t, x) = x'$, la expresión funcional de la ODE

```
def van_der_pol(x,t,mu):
#Oscilador de Van Der Pol
    X, Xdot = x
    return [Xdot, mu*(1-X**2)*Xdot - X]
```

Integración de ODE

Con eso podemos pasar a definir la escala de tiempo, los parámetros del problema y las condiciones iniciales

```
x0 = [0,1] #Condiciones iniciales
t = arange(0,50,0.01)
mu = 5 #Parámetro
p = (mu,)  #Si le pasa (mu) no agarra como tupla
```

Integración de ODE

Para que finalmente ejecutemos el integrador, que para Python usa Runge Kutta de orden 4 u 5 dependiendo de los parámetros finos del método

```
sol = sp.integrate.odeint(van_der_pol,x0,t,args=p)
```

Integración de ODE

Si queremos representar el resultado, usamos nuevamente la librería matplotlib.pyplot, pero con el siguiente código

```
plt.figure(1,figsize=(16,9))
plt.subplot(1,2,1)
plt.plot(t,sol[:,0]) #Posición
plt.subplot(1,2,2)
plt.plot(sol[:,0],sol[:,1]) #Diagrama de fase
plt.show()
```

Integración de ODE

Y acá está el resultado