Python para Finanzas

Módulo 3: Funciones y Matemáticas

Francisco A. Ibáñez

Universidad Adolfo Ibáñez

Julio 2021

Contenidos

1 Funciones
- Introducción
- Elementos básicos de una función

2 Matemáticas en Python
- NumPy
- Generación de Números Aleatorios
- Álgebra lineal

3 Aplicación: Analíticas de Portafolio
- Matemática de Portafolios
- Regresiones Lineales

Contenidos

1 Funciones
- Introducción
- Elementos básicos de una función

2 Matemáticas en Python
- NumPy
- Generación de Números Aleatorios
- Álgebra lineal

3 Aplicación: Analíticas de Portafolio
- Matemática de Portafolios
- Regresiones Lineales

Introduciendo funciones

- Una función es un bloque de código reutilizable diseñado para cumplir una tarea específica
- Puede recibir argumentos (o no) para lograr distintos resultados

Un ejemplo sencillo (sin argumentos)

```
1 # Mi primera funcion
2 def saluda_usuario():  # Sin argumentos!
3   print('Hola!')
4
5 saluda_usuario()  # Llamemos a la funcion con parentesis
```

```
Hola!
```

```
6 saluda_usuario  # Y los parentesis?
```

```
<function saluda_usuario at 0x000002C33546E1F8>
```

Un ejemplo sencillo (con argumentos)

```
1 # Mi segunda funcion
2 def saluda_usuario(nombre):  # Ahora definimos argumentos
3   print(f'Hola {nombre.title()}!')
4
5 saluda_usuario(nombre='pepe')  # Le pasamos el argumento a
      la funcion
```

```
Hola Pepe!
```

Contenidos

1 Funciones
- Introducción
- Elementos básicos de una función

2 Matemáticas en Python
- NumPy
- Generación de Números Aleatorios
- Álgebra lineal

3 Aplicación: Analíticas de Portafolio
- Matemática de Portafolios
- Regresiones Lineales

Argumentos posicionales y keywords

```
1  def describir_mascota(especie, nombre):
2    # Muestra informacion sobre una mascota
3    print(f'\nYo tengo un {especie}.')
4    print(f'Mi {especie} se llama {nombre.title()}.')
5
6  describir_mascota('hamster', 'harry')  # Argumentos
       posicionales
```

```
Yo tengo un hamster.
Mi hamster se llama Harry.
```

```
10 describir_mascota(nombre='harry', especie='hamster')  #
       Pasando como keywords
```

```
Yo tengo un hamster.
Mi hamster se llama Harry.
```

Argumentos posicionales y keywords (cont'd)

¿Y si no paso argumentos?

```
11 describir_mascota()
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: describir_animal() missing 2 required positional arguments:
'especie' and 'nombre'
```

Valores por defecto

```
1 def describir_mascota(especie='perro', nombre='cachupin'):
2   # Muestra informacion sobre una mascota
3   print(f'\nYo tengo un {especie}.')
4   print(f'Mi {especie} se llama {nombre.title()}.')
5
6 describir_mascota()
```

```
Yo tengo un perro.
Mi perro se llama Cachupin.
```

```
7 describir_mascota(especie='gato')  # Pasando un solo
    argumento
```

```
Yo tengo un gato.
Mi gato se llama Cachupin.
```

Return

- Una función no siempre tiene que mostrar el resultado directamente, sino que puede procesar datos y devolver (**return**) un resultado
- Return toma un valor dentro de la funcion y lo entrega como *output* de esta para que pueda ser utilizada por otros segmentos del programa
- Una función puede devolver (y aceptar como argumentos) cualquier tipo de variable, incluyendo estructuras más complicadas como diccionarios y listas

Return (cont'd)

```
1 def potencia(numero, n):
2   resultado = numero ** n
3   return resultado
4
5 valor_1 = potencia(2, 2)
6 valor_2 = potencia(2, 3)
7 print(valor_1 + valor_2)
```

```
12
```

Usando funciones con loops

```
1  def formatea_nombre(nombre, apellido):
2    nombre_completo = f'{nombre} {apellido}'
3    return nombre_completo.title()
4
5  while True:
6      print('\nPor favor dime tu nombre...')
7      print("(entra 'salir' para terminar)")
8
9      nom = input('Nombre: ')
10     if nom == 'salir':
11         break
12
13     ape = input('Apellido: ')
14     if ape == 'salir':
15         break
16
17     nombre = formatea_nombre(nombre=nom, apellido=ape)
18     print(f'\nHola , {nombre}!')
```

Contenidos

1 Funciones
- Introducción
- Elementos básicos de una función

2 Matemáticas en Python
- NumPy
- Generación de Números Aleatorios
- Álgebra lineal

3 Aplicación: Analíticas de Portafolio
- Matemática de Portafolios
- Regresiones Lineales

El siguiente paso

- De manera nativa, Python puede manejar operaciones matemáticas elementales (i.e. aritmética)
- Si queremos dar el siguiente paso y realizar operaciones más avanzadas, necesitamos utilizar un paquete externo
- **NumPy** es una librería ampliamente utilizada que nos permite realizar computación científica en Python

Usos de NumPy

NumPy tiene múltiples usos en finanzas, incluyendo:

- Algebra lineal $\rightarrow$ Portfolio management, análisis factorial
- Muestreo aleatorio $\rightarrow$ Simulaciones de Monte Carlo
- Interpolación polinómica $\rightarrow$ Estructura de tasas de interés
- Transformadas de Fourier $\rightarrow$ Valorización de opciones

Importando un paquete externo

- Podemos utilizar funciones que son partes de paquetes desarrollados por la comunidad por medio del comando **import**
- Esto asume que el paquete/librería ya está instalada en nuestro ambiente de Python
- El programa podrá hacer uso de las funciones dentro del paquete por medio de **paquete.funcion()**
- También podemos definir un alias (más corto) para el paquete por medio de **import** *paquete* **as** *alias*

Mis primeros pasos con NumPy

- Los **ndarray** son la piedra angular de NumPy
- Son un arreglo de datos sobre el cual se pueden realizar operaciones matemáticas
- Son creados usando la función **array()**

```
1 import numpy as np
2
3 # Los ndarray pueden ser creados pasando listas o tuples
4 v1 = np.array([1, 2, 3])
5
6 type(v1)
```

```
<class 'numpy.ndarray'>
```

Dimensionalidad

Los array pueden tener cualquier número de dimnesiones.

```
1  import numpy as np
2
3  escalar = np.array(1)
4  vector = np.array([1, 2, 3])
5  matrix_2d = np.array([[1, 2, 3], [4, 5, 6]])
6  matrix_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3],
       [4, 5, 6]]])
7
8  print(escalar.ndim)
9  print(vector.ndim)
10 print(matrix_2d.ndim)
11 print(matrix_3d.ndim)
```

```
0
1
2
3
```

Indexación

Se puede acceder, utilizar, y modificar las entradas de los array usando la misma indexación que listas y tuples.

```
1 import numpy as np
2
3 v1 = np.array([1, 2, 3, 4, 5])
4
5 print(v1[4] + v1[3])
```

```
9
```

```
6 print(v1[-1] + v1[-2])
```

```
9
```

Indexación (cont'd)

En el caso de matrices, se sigue la misma convención que en álgebra lineal (fila, columna).

```
1 import numpy as np
2
3 m1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
4
5 print(m1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
6 m1[2, -1]  # Accediendo entrada (3, 3) de la matriz
```

```
9
```

Slicing

Al igual que en listas y tuples, se puede realizar slicing sobre arrays para acceder a segmentos dentro de éstos.

```
1 import numpy as np
2
3 v1 = np.array(range(10))
4
5 print(v1[-3:])  # Desde la antepenultima entrada en adelante
```

```
[7 8 9]
```

```
6 print(v1[2:6])
```

```
[2 3 4 5]
```

Contenidos

1 Funciones
- Introducción
- Elementos básicos de una función

2 Matemáticas en Python
- NumPy
- Generación de Números Aleatorios
- Álgebra lineal

3 Aplicación: Analíticas de Portafolio
- Matemática de Portafolios
- Regresiones Lineales

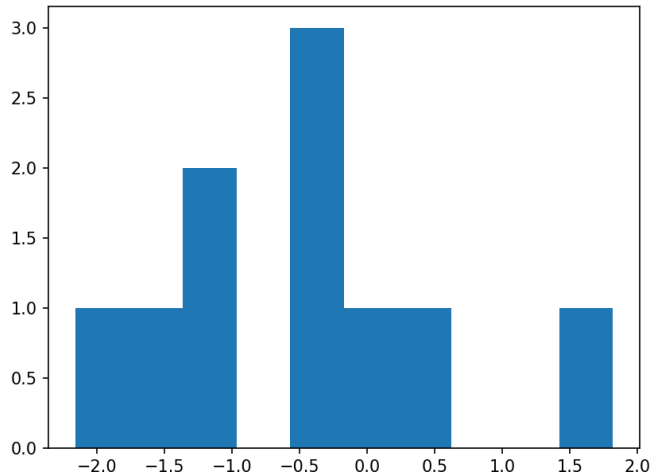
Muestreo aleatorio

NumPy contiene un amplio módulo de generación de números pseudo-aleatorios (**numpy.random**), que incluye funciones como:

- **numpy.random.rand()**: devuelve **float** aleatorios provenientes de una distribución uniforme [0, 1)
- **numpy.random.randint()**: devuleve **int** aleatorios entre dos límites
- **numpy.random.randn()**: devuelve **float** aleatorios provenientes de una distribución normal estándar ($\sim \mathcal{N}(0, 1)$)

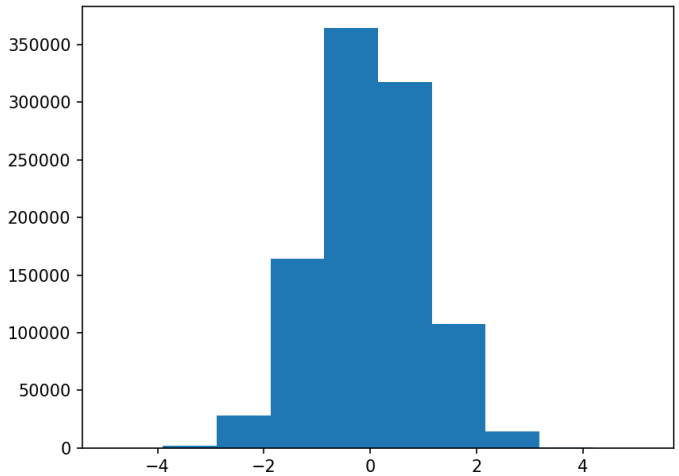
Ejemplo de muestreo normal

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Creemos una pequena muestra
5 muestra = np.random.randn(10)
6 plt.hist(muestra)  # Creamos un histograma
7 plt.show()
```



Ejemplo de muestreo normal (cont'd)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Creemos una muestra mas grande
5 muestra = np.random.randn(1000000)  # <--------
6 plt.hist(muestra)  # Creamos un histograma
7 plt.show()
```



Contenidos

1 Funciones
- Introducción
- Elementos básicos de una función

2 Matemáticas en Python
- NumPy
- Generación de Números Aleatorios
- Álgebra lineal

3 Aplicación: Analíticas de Portafolio
- Matemática de Portafolios
- Regresiones Lineales

Antes de empezar

Al trabajar con vectores, primero nos tenemos que asegurar que su forma es de $N \times 1$:

```
1 import numpy as np
2
3 v1 = np.array([1, 2, 3]).reshape(-1, 1)  # -1: wildcard (N)
4 print(v1)
```

```
[[1]
 [2]
 [3]]
```

```
5 print(v1.shape)
```

```
(3, 1)
```

Transposición

```
6 v1_T = v1.T  # Transponemos con .T
7 print(v1_T)
```

```
[[1 2 3]]
```

```
8 m1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
9 print(m1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
10 m1_T = m1.T
11 print(m1_T)
```

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Productos

En NumPy, las multiplicaciones de matrices se llevan a cabo con el operador **@**.

```
12 print(v1.T @ v1)
```

```
[[14]]
```

```
13 print(m1.T @ m1)
```

```
[[66  78  90]
 [78  93 108]
 [90 108 126]]
```

```
14 print(m1 @ v1)  # La forma de las matrices debe coincidir
```

```
[[14]
 [32]
 [50]]
```

Contenidos

1 Funciones
- Introducción
- Elementos básicos de una función

2 Matemáticas en Python
- NumPy
- Generación de Números Aleatorios
- Álgebra lineal

3 Aplicación: Analíticas de Portafolio
- Matemática de Portafolios
- Regresiones Lineales

Combinaciones lineales

- El desempeño (retorno) de un portafolio de inversiones es una **combinación lineal** del desempeño de los activos que lo componen
- Por ejemplo, en el caso de dos activos (A, B), el retorno (μ) de un portafolio (P) va a estar dado por:

$$\mu_P = w_A \mu_A + w_B \mu_B$$

y su varianza (σ^2) por:

$$\sigma_P^2 = w_A^2 \sigma_A^2 + w_B^2 \sigma_B^2 + 2 w_A w_B \sigma_{A,B}$$

- ¿Y qué pasa en el caso de 3 o más activos?

Generalizar es bueno (en algunos casos)

La verdad es que es mejor ni intentarlo, ya que lo anterior se puede generalizar con matrices, donde:

$$w = \begin{bmatrix} w_A \\ w_B \end{bmatrix}, \quad \mu = \begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_A^2 & \sigma_{A,B} \\ \sigma_{B,A} & \sigma_B^2 \end{bmatrix}$$

De esta forma, ahora el retorno y varianza de un portafolio pueden ser expresados como:

$$\mu P = w^T \mu$$

$$\sigma_P^2 = w^T \Sigma w$$

Un ejemplo

```python
import numpy as np

r = np.array([.15, .05]).reshape(-1, 1)  # Retornos
w = np.array([.3, .7]).reshape(-1, 1)  # Pesos
S = np.array([[.015, .002], [.002, .0225]])  # Covarianza

ret_p = w.T @ r
std_p = np.sqrt(w.T @ S @ w)

# Formateamos los resultados
ret_p_fmt = np.round(ret_p.item() * 100, 2)
std_p_fmt = np.round(std_p.item() * 100, 2)

print('El retorno del portafolio es {}%'.format(ret_p_fmt))
print('Y su volatilidad {}%'.format(std_p_fmt))
```

```
El retorno del portafolio es 8.0%
Y su volatilidad 11.5%
```

Contenidos

1 Funciones
- Introducción
- Elementos básicos de una función

2 Matemáticas en Python
- NumPy
- Generación de Números Aleatorios
- Álgebra lineal

3 Aplicación: Analíticas de Portafolio
- Matemática de Portafolios
- Regresiones Lineales

Intro

- Las regresiones lineales son usadas para estudiar la relación entre una variable dependiente (endógena) y una o más variables independientes (exógenas)
- La forma genérica de una regresión lineal con k variables independientes es:

$$y = x_1\beta_1 + x_2\beta_2 + ... + x_k\beta_k + \varepsilon$$

Forma matricial

El modelo anterior puede ser reorganizado de la siguiente manera:

$$y = Xb + e$$

donde

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,k} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,k} \end{bmatrix}, \quad e = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

Encontrando a **b**

En términos matriciales elijimos **b** de manera tal que:

$$\min_b e^T e = (y - Xb)^T (y - Xb)$$

Al expandir esto, tenemos:

$$e^T e = y^T y - 2y^T Xb + bX^T Xb$$

Y la condición de primer orden:

$$\frac{\partial e^T e}{\partial b} = -2X^T y + 2X^T Xb = 0$$

$$b = (X^T X)^{-1} X^T y$$

Beta en una función

```python
import numpy as np

# Simulemos 100 retornos para probar nuestra funcion
np.random.seed(0)  # Importante!
ret_i = np.random.randn(100, 1) * .2 + .08
ret_mercado = np.random.randn(100, 1) * .15 + .06

def beta(y, X):
  b = np.linalg.inv(X.T @ X) @ X.T @ y

  return b

b = beta(y=ret_i, X=ret_mercado)

print(b)
```

```
[[0.346053]]
```

Beta en una función (ahora sí)

```
8  def beta(y, X, const=True):
9    if const:
10     n = X.shape[0]  # Numero de observaciones
11     X = np.concatenate([np.ones((n, 1)), X], axis=1)
12
13   b = np.linalg.inv(X.T @ X) @ X.T @ y
14
15   return b.flatten()
16
17 b = beta(y=ret_i, X=ret_mercado, const=True)
18
19 print(b)
```

```
[0.08146945 0.14511577]
```