Plan (6h)

Cours-TD d'introduction à Python, 2 séances de 3h.

1. Séance 1 : Les bases du langage
2. Séance 2 : Python pour la science des données

Site web :
`http://chercheurs.lille.inria.fr/pgermain/neurones2018/`

Un peu d'histoire...

1994 Python 1.0 : Développé au CWI (Amsterdam, Pays-Bas)

2000 Python 2.0 : Introduction du ramasse-miettes, listes par compréhensions, ...

2008 Python 3.0 : Correction de quelques incohérences (non-rétrocompatible)

2016 Python 3.6

Pourquoi le langage se nomme «Python» ?

Un peu d'histoire...

1994 Python 1.0 : Développé au CWI (Amsterdam, Pays-Bas)

2000 Python 2.0 : Introduction du ramasse-miettes, listes par compréhensions, ...

2008 Python 3.0 : Correction de quelques incohérences (non-rétrocompatible)

2016 Python 3.6

Pourquoi le langage se nomme «Python» ?

En l'honneur du groupe d'humoristes Anglais *Monty Python*.

Pour votre culture générale : `https://www.youtube.com/watch?v=iV2ViNJFZC8`

Quelques infos sur Python avant de pratiquer

- Langage multiparadigme : on peut faire la programmation orientée objet, impérative, fonctionnelle ou procédurale
- Fonctionne sur tous les systèmes d'exploitation courants (Windows, Linux, Mac OS)
- Typage dynamique (contrairement au C++) : les variables ne sont pas déclarées, leur type est détecté lors de l'exécution du code

Quelques infos sur Python avant de pratiquer

- Langage multiparadigme : on peut faire la programmation orientée objet, impérative, fonctionnelle ou procédurale
- Fonctionne sur tous les systèmes d'exploitation courants (Windows, Linux, Mac OS)
- Typage dynamique (contrairement au C++) : les variables ne sont pas déclarées, leur type est détecté lors de l'exécution du code
- Indentation : pas de parenthèses ou accolades pour délimiter des blocs de code Python
- Pas de gestion de la mémoire pour le codeur (ramasse-miette)
- Nombreuses librairies et outils disponibles : ils doivent être importés quand on en a besoin, il faut donc connaître ceux qui nous sont utiles : `NumPy`, `SciPy`, `matplotlib`, `PyTables`, `pandas`, `scikit-learn` ...)

Quelques infos sur Python avant de pratiquer

- Les environnements de développements interactifs (IDE) : IPython (à l'intérieur d'un terminal), Jupyter-notebook (basé sur un navigateur), Spyder, PyCharm, Visual Studio Code ...

- Actuellement le langage de prédilection pour les librairies de *deep learning* :
  - PyTorch (supporté par Facebook),
  - TensorFlow / Keras (supporté par Google)

Section réservée aux étudiants du Master Finance

Python pour la finance

- Python permet de répondre à certains problèmes des institutions financières : traitement des données massives (big data), éco-finance en temps réel

Exemple de code : estimateur de Monte-Carlo pour le prix d'un Call européen

On suppose que le sous-jacent suit le modèle de Black-Scholes-Merton :

$$S_T = S_0 \exp\left\{(r - \frac{\sigma^2}{2})T + \sigma\sqrt{T}Z\right\} \text{ avec } Z \sim \mathcal{N}(0,1).$$

On donne $S_0 = 100$, $\sigma = 0.2$, $T = 1$, $r = 0.05$ et $K = 105$.

Exemple de code : estimateur de Monte-Carlo pour le prix d'un Call européen

On suppose que le sous-jacent suit le modèle de Black-Scholes-Merton :

$$S_T = S_0 \exp\left\{(r - \frac{\sigma^2}{2})T + \sigma\sqrt{T}Z\right\} \text{ avec } Z \sim \mathcal{N}(0,1).$$

On donne $S_0 = 100$, $\sigma = 0.2$, $T = 1$, $r = 0.05$ et $K = 105$.
L'algorithme est le suivant :

1. Simuler n réalisations de Z, notées $z(1), \ldots, z(n)$
2. Calculer les n valeurs de S_T correspondantes, notées $S_T(1), \ldots, S_T(n)$
3. Calculer les n payoffs correspondants, notés $h_T(1), \ldots, h_T(n)$
4. Calculer l'estimation du prix du Call au temps 0 via

$$C_0 \simeq \mathrm{e}^{-rT}\frac{1}{n}\sum_{i=1}^{n} h_T(i).$$

Exemple de code : estimateur de Monte-Carlo pour le prix d'un Call européen

On fixe les paramètres :

```
S0 = 100.
K = 105.
T = 1.0
r = 0.05
sigma = 0.2
```

Exemple de code : estimateur de Monte-Carlo pour le prix d'un Call européen

On fixe les paramètres :

```
S0 = 100.
K = 105.
T = 1.0
r = 0.05
sigma = 0.2
```

On utilise NumPy pour le codage de l'algorithme :

```
import numpy as np
from math import exp, sqrt
n = 100000
z = np.random.standard_normal(n)
ST = S0 * np.exp((r-0.5*sigma**2)*T + sigma*sqrt(T)*z)
hT = np.maximum(ST-K,0)
C0 = exp(-r*T)*np.sum(hT)/n
```

Exemple de code : estimateur de Monte-Carlo pour le prix d'un Call européen

Voici le code (quasi identique) que l'on pourrait enregister dans un fichier .py :

```
# Estimation d'un call européen dans le modèle BSM
import numpy as np
# Valeurs des paramètres
S0 = 100. ; K = 105. ; T = 1.0
r = 0.05 ; sigma = 0.2 ; n = 100000
# Algorithme
z = np.random.standard_normal(n)
ST = S0 * np.exp((r-0.5*sigma**2)*T + sigma*np.sqrt(T)*z)
hT = np.maximum(ST-K,0)
C0 = np.exp(-r*T) * np.sum(hT) / n
# Sortie
print('Valeur du call :', C0)
```

Calculer la volatilité de log-rendements de l'action TESLA

Il faut charger d'abord les librairies nécessaires :

```
import numpy as np
import pandas as pd
import pandas_datareader.data as web
import datetime as dt
```

On récupère ensuite les données :

```
start = dt.datetime(2014,1,1)
end = dt.datetime(2018,1,1)
tsla = web.DataReader('TSLA','yahoo', start, end)
tsla.tail() # affiche la fin du tableau de données
```

On calcule les log return et volatilités annualisées

```
tsla['LR'] = np.log(tsla['Close']/tsla['Close'].shift(1))
tsla['Vol'] = tsla['LR'].rolling(252).std()*np.sqrt(252)
```

Et les graphiques afférents ...

```
tsla[['Close','Vol']].plot(subplots=T,color='blue')
```

On peut spécifier la taille des graphiques à l'aide de `figsize()` en argument optionnel de la commande `plot()`.