NumPy et LaTeX

Dans ce notebook Jupyter, nous allons présenter

- la mise en page d'expressions mathématique avec **LaTeX**
- la bibliothèque pour le calcul numérique **NumPy**
- l'utilisation des arrays numpy dans **OpenCV**

Le notebook Jupyter est un outil puissant pour développer, tester et documenter des algorithmes, des applications mathématiques ou tout simplement du code informatique. Il permet de combiner dans un seul document:

- documentation (avec Markdown)
- code et son résultat
- images
- formules mathématiques (avec LaTex)

LaTeX

Sources et références:

- http://data-blog.udacity.com/posts/2016/10/latex-primer/ (http://data-blog.udacity.com/posts/2016/10/latex-primer/)
- https://fr.wikibooks.org/wiki/LaTeX/Écrire_des_mathématiques (https://fr.wikibooks.org/wiki/LaTeX/Écrire_des_mathématiques)

LaTeX est un outil puissant pour mettre en forme des formules mathématiques. Il permet de produire des documents scientifiques de grande qualité typographique, principalement des livres et des articles scientifiques. LaTeX est largement utilisé dans le monde scientifique pour les publications dans les domaines de

- mathématiques
- physique
- informatique
- chimie

Contrairement aux traitements de texte habituels, ce n'est pas un système WYSIWIG (what you see is what you get). La mise en forme se fait par des balises. **LaTeX** entre donc dans la catégorie des langages à balises comme le **HTML** pour la descriptions des pages web, ou sa version simplifié **Markdown** pour les pages README.md sur GitHub.

Formules en ligne et formules centrées

Les formules en ligne sont entourées par un simple symbol dollar `$` . Par exemple la formule de Pythagore s'écrit en LaTex `$c = \sqrt{a^2 + b^2}$` et donne le résultat $c = \sqrt{a^2 + b^2}$ intégré dans le texte en ligne.

Pour faire apparaitre une formule seul sur une ligne et centré, il faut entourer la formule LaTeX par un double symbole dollar `$$` . L'expression LaTeX `$$c = \sqrt{a^2 + b^2}$` donne le résultat

$$c = \sqrt{a^2 + b^2}$$

Le placement des formules est différent selon le mode en ligne ou en mode centré. L'expression `$e^x = \sum_{n=0}^\infty \frac{x^n}{n!}$` donne ceci en mode en ligne $e^x = \sum_{n=0}^\infty \frac{x^n}{n!}$ et ceci en mode centré

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Puissances et indices

- le symbole chapeau `^` est utilisé pour exprimer une puissance. Par exemple `$x^2$` donne x^2
- le symbole tiret bas `_` est utilisé pour exprimer un indice. Par exemple `$x_1$` donne x_1
- les accolades `{...}` sont utilisées pour groupes des expressions. Par exeemple `$x^{2+n}$` donne x^{2+n}
- puissance et indice peuvent être combinées. Par exemple `$x_{ij} ^{4j}$` donne x_{ij}^{4j}

Fractions

Pour écrire une fraction, il faut utiliser la balise `\frac`. En fait, la plupart des balises spéciales de LaTeX utilisent la barre oblique en arrière `\`. Donc par exemple, `$\frac x 2` donne $\frac{x}{2}$

Toutefois, il est plus sûr d'utiliser des accolades pour grouper le numérateur et le dénominateur. Par exemple `$\frac{12-z}{x-4}$` donne

$$\frac{12-z}{x-4}$$

Symboles mathématiques

Quelques symboles utiles :

- `\cdot` (center dot) : $\cdot$
- `\times` (multiplication) : $\times$
- `\alpha` (lettre grèque): α
- `\beta` : β
- `\theta` : θ
- `\infty` (infinité) : ∞
- `\approx` : $\approx$
- `\leq` (less or equal) : $\leq$
- `\geq` (greater or equal) : $\geq$
- `\neq` (not equal) : $\neq$

Trigonomètrie

LaTeX peut être utiliser pour écrire des fonctions trigonomètriques, avec les balises `\sin`, `\cos`, `\tan`, et `\cot`. Exemples :

- `\sin(4x)` donne $\sin(4x)$
- `\cos(z^2)` donne $\cos(z^2)$
- `\tan\alpha` donne $\tan\alpha$

Notons que les parenthèses ne sont pas nécessaires.

Matrices

Alors, pour ce qui est des matrices, cela se complique un peu. On va utiliser les balises `\begin {type}` et `\end {type}` en indiquant quel est le type. Les types possibles sont entre-autre :

- `matrix`
- `pmatrix` (parenthesis)
- `bmatrix` (brackets)
- `vmatrix` (
- `Vmatrix`

Nous verrons plus tard les différences. L'interieur d'une matrice s'écrit comme suit : `a & b \ c & d'.

- l'esperluette `&` indique un changement de colonne
- la barre oblique en arrière double `\\` indique un changement de ligne

Donc, pour créer un matrice, on va écrire `\begin{matrix} a & b \\ c & d \end{matrix` , ce qui donne

$$\begin{matrix} a & b \\ c & d \end{matrix}$$

Les différents types de matrice sont donc :

- matrix : $\begin{matrix} a & b \\ c & d \end{matrix}$
- pmatrix : $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$
- bmatrix : $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$
- vmatrix : $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$
- Vmatrix : $\begin{Vmatrix} a & b \\ c & d \end{Vmatrix}$

On peut faire toute sorte de matrices, 2×2, 3×3, 1×2, 2×3, etc.

Il suffit de jouer avec les `&` et les `\\` :

$$\begin{bmatrix} 1 & 3 \\ 5 & 9 \end{bmatrix} \quad \begin{pmatrix} 1 & 37 & 12 \\ 5 & 1 & 6 \end{pmatrix} \quad \begin{vmatrix} 1 & 37 & 12 \\ 5 & 1 & 6 \\ 13 & 3 & 8 \end{vmatrix}$$

Vecteurs

Les vecteurs sont juste une forme de matrice. Pour écrire un vecteur 3D, il sufffit d'ecrire une matrice de type pmatrix et de dimension 1x3 :

`\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}` donne $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$

Pour la notation algèbrique, c'est comme ceci : `\vec{v}` donne $\vec{v}$

NumPy

NumPy est une extenseion du language de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérants sur ces tableaux.

Sources et références:

- https://fr.wikipedia.org/wiki/NumPy (https://fr.wikipedia.org/wiki/NumPy)
- https://docs.scipy.org/doc/numpy/user/quickstart.html (https://docs.scipy.org/doc/numpy/user/quickstart.html)
- https://docs.scipy.org/doc/numpy/numpy-ref-1.14.2.pdf (https://docs.scipy.org/doc/numpy/numpy-ref-1.14.2.pdf) (1333 pages)
- https://docs.scipy.org/doc/numpy/numpy-user-1.14.2.pdf (https://docs.scipy.org/doc/numpy/numpy-user-1.14.2.pdf) (143 pages)

Création de tableau

Il existe plusieurs manière de les créer, la première étant de transformer une liste Pyhton en tableau numpy. La convention est d'importer NumPy sous le nom de `np`.

In [1]:

```
import numpy as np
x = np.array([1, 2, 3])
x
```

Out[1]:

```
array([1, 2, 3])
```

C'est également possible de produire des plages séquentielles.

In [3]:

```
y = np.arange(10)
y
```

Out[3]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Le type de base de NumPy est `ndarray`

In [4]:

```
type(x)
```

Out[4]:

```
numpy.ndarray
```

Type de données

Contrairement à une liste Python, dans un tableau NumPy tous les éléments ont le même type.

Les types les plus utiles sont :

- `float_` : Nombre à virgule
- `int8` : Nombre entier 8 bits (-128...127)
- `uint8` : Nombre entier 8 bits non-signe (unsigned) (0...255) utilisé dans OpenCV pour représenter les pixels RGB

La liste complète: https://docs.scipy.org/doc/numpy-1.10.1/user/basics.types.html (https://docs.scipy.org/doc/numpy-1.10.1/user/basics.types.html)

In [5]:

```
x = np.array([1, 2, 3], dtype=np.uint8)
x
```

Out[5]:

```
array([1, 2, 3], dtype=uint8)
```

Matrice nulle

Une matrice nulle est une matrice dont tout les coefficients sonts nulles. La fonction suivante produit une tel matrice nulle `np.zeros(dims, dtype = "type"`

- dims : Dimensions de la matrice, elle doit être sous la forme d'un tuple.
- dtype : Type de valeur contenue dans la matrice

In [6]:

```
M = np.zeros((3, 3), dtype="int_")
print(M)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

On peut ajouter autant de dimensions que l'on veut. Si l'on voulait faire un matrice **OpenCV** pour contenir une image icône de 64x64 pixels, avec les 3 octets RVB, on pourait le faire comme cela :

In [7]:

```
M = np.zeros((64, 64, 3), dtype="uint8")
```

La Matrice M est en 3 dimensions et contient que des zéros, donc représente une image ou tout les 64x64 pixels sont noirs.

On peut utiliser la méthode np.ones qui fonctionne de la même manière mais qui remplit la matrice de 1.

In [8]:

```
M = np.ones((3, 3), dtype="int_")
print(M)
```

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

On peut aussi créer une matrice à partir des valeurs que l'on veut mettre dedans. Si l'on veut une matrice comme celle-ci :

$$\begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 4 & 1 & 0 \end{pmatrix}$$

Il faut faire cela :

In [9]:

```
M = np.int_([[1, 0, 3],
             [0, 1, 4],
             [4, 1, 0]])
```

le int_ permet de définir le type, voir ci-dessus pour avoir une liste des types.

In [10]:

```
M = np.ones((3, 3), dtype = "string_")
print(M)
```

```
[['1' '1' '1']
 ['1' '1' '1']
 ['1' '1' '1']]
```

Opérations matricielles

NumPy permet d'aditionner et de soustraire simplement des matrices, il suffit d'utiliser les signes habituels.

In [11]:

```
M1 = np.int_([[1, 0],
              [3, 4]])
M2 = np.int_([[2, 4],
              [3, 5]])
print(M1 + M2)
```

```
[[3 4]
 [6 9]]
```

Ce qui représente l'opération d'addition de matrices:

$$\begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 2 & 4 \\ 3 & 5 \end{pmatrix} = \begin{pmatrix} 3 & 4 \\ 6 & 9 \end{pmatrix}$$

In [12]:

```
print(M1 - M2)
```

```
[[-1 -4]
 [ 0 -1]]
```

Ce qui représtante la soustraction de matrices:

$$\begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix} - \begin{pmatrix} 2 & 4 \\ 3 & 5 \end{pmatrix} = \begin{pmatrix} -1 & -4 \\ 0 & -1 \end{pmatrix}$$

NumPy permet aussi de faire des opérations plus complèxes, comme le produit de deux matrices (dot) :

In [13]:

```
print(M1.dot(M2))
```

```
[[ 2  4]
 [18 32]]
```

$$\begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 2 & 4 \\ 3 & 5 \end{pmatrix} = \begin{pmatrix} 1 \cdot 2 + 0 \cdot 3 & 1 \cdot 4 + 0 \cdot 5 \\ 3 \cdot 2 + 4 \cdot 3 & 3 \cdot 4 + 4 \cdot 5 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 18 & 32 \end{pmatrix}$$

Le module d'algèbre linéaire de NumPy `linalg` apporte quelques outils interessants, comme un moyen de calculer le déterminant d'une matrice:

In [14]:

```
print(np.linalg.det(M1))
```

```
4.0
```

ou de calculer l'inverse d'une matrice avec l'opérateur `inv` :

In [15]:

```
Minv = np.linalg.inv(M2)
print(Minv)
print(M2.dot(Minv).astype("int_"))
```

```
[[-2.5  2. ]
 [ 1.5 -1. ]]
[[0 0]
 [0 1]]
```

La matrice inverse de $\begin{pmatrix} 2 & 4 \\ 3 & 5 \end{pmatrix}$ est donc $\begin{pmatrix} -2,5 & 2 \\ 1,5 & -1 \end{pmatrix}$ on peut le vérifier en multipliant les deux matrices entre-elles. Cela donne bien la matrice unité.

(J'ai changé le format de la matrice en int_ car l'opération pour trouver l'inverse crée des nombres à virgules et par exmple à la place d'obtenir zéro, on obtient $\approx 4,4 \times 10^{-16}$ ce qui revient à zéro. Mais pour plus de lisibilité, j'ai changé le format.)

Utilisation de NumPy dans OpenCV

L'une des raisons principales pour lesquels nous utilisons **NumPy**, c'est pour son utilisté dans **OpenCV**. Les images, chez **OpenCV**, sont sous forme de matrices. Donc **NumPy** permet de créer des images, de travailler dessus, des les transformer, etc.

In [2]:

```
import cv2
import matplotlib.pyplot as plt

def pltShow(img):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
```

Créations d'images vièrges

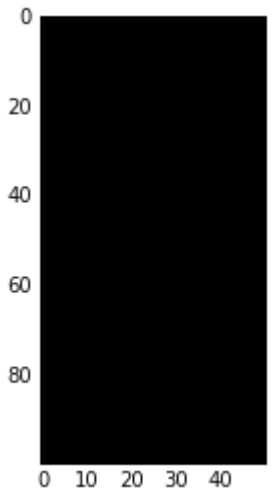
Les images sont des matrices 3d sous la forme largeur x hauteur x cannaux. La largeur et la hauteur définisse la taille de l'image et donc le nombre de pixel qu'elle contient, et les cannaux sont le nombre de couleur contenu dans un pixel. Une image en RGB a 3 cannaux, un pour le rouge, un pour le vert et un pour le bleu.

L'intentité d'une couleur ne s'entend que de 0-255. C'est pourquoi on utilise le type uint8.

Imaginons que l'on veut créer une image BGR noire 100x50, on peut le faire comme cela :

In [3]:

```
N = np.zeros([100, 50, 3], dtype="uint8")
pltShow(N)
```



Transformations

NumPy permet de créer des matrices de transformation affines, comme la translation ou la rotation. A l'aide de la fonction `warpAffine()` de **cv2**.

Source : https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html (https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html)

Translation ¶

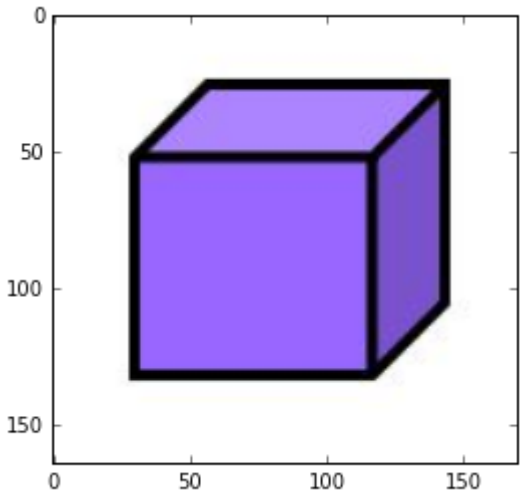
La matrice de translation dans **OpenCV** s'écrit comme cela :

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & ty \end{pmatrix}$$

t_x et t_y sont les valeurs de la transaltion respectivement en x et en y. Donc si l'on veut créer une matrice de translation qui déplace l'image de 50 pixel en x et 25 pixel en y, il faut que la matrice soit comme ceci :

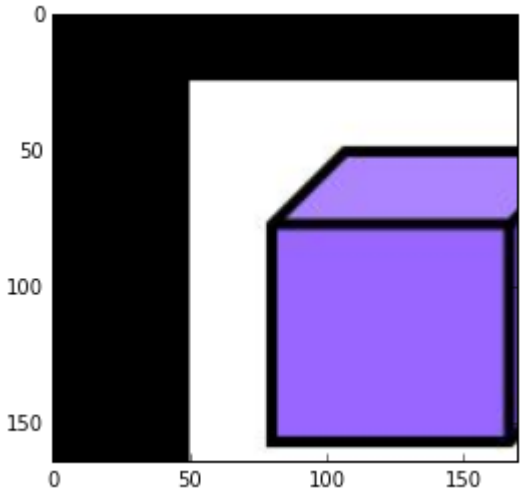
In [4]:

```
T = np.float32([[1, 0, 50],
                [0, 1, 25]])
image = cv2.imread("./img_gabriel/cube.jpg")

(h, w) = image.shape[:2]

pltShow(image)
```



In [5]:

```
imageT = cv2.warpAffine(image, T, (w, h))
pltShow(imageT)
```



Rotation et echelle

La matrice dde rotation est habituellement sous cette forme :

$$\begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

Toutefois, dans **OpenCV**, la matrice de rotation prends aussi en compte le point de rotation et l'echelle de l'image. Cela donne donc,

$$\begin{pmatrix} h\cos\theta & h\sin\theta & (1-h\cos\theta)\cdot a - h\sin\theta \cdot b \\ -h\sin\theta & h\cos\theta & h\sin\theta \cdot a + (1-h\cos\theta)\cdot b \end{pmatrix}$$

avec :

- (a, b) : le point de rotation
- θ : l'angle de rotation (Dans le sens contraire des aiguilles d'une montre
- h : L'echelle de l'image

Ce qui est un peu complexe. C'est pourquoi cv2 offre une méthode pour générer une matrice de rotation simplement :

```
cv2.getRotationMatrix2D((a, b), θ, h)
```

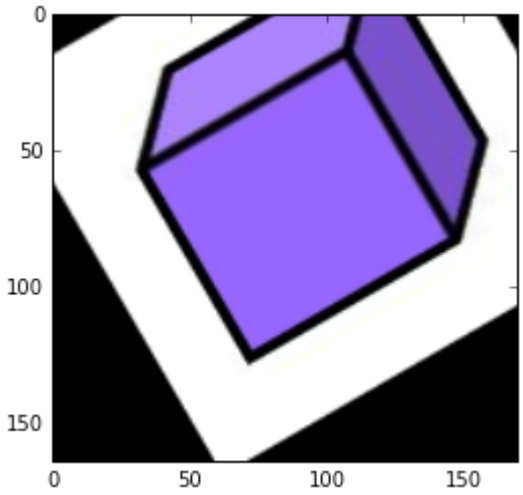
Si l'on veut tourner notre image de 30° autours des points $(40, 50)$, voila comment l'on doit créer la matrice de rotation :

In [6]:

```
R = cv2.getRotationMatrix2D((40, 50), 30, 1)
```

In [7]:

```
imageR = cv2.warpAffine(image, R, (w, h))
pltShow(imageR)
```



NumPy permet aussi de coller des images les unes à coté des autres, pour cela il faut utiliser la méthode `np.hstack([img1, img2])`

In [8]:

```
stacked = np.hstack([image, imageT, imageR])
pltShow(stacked)
```