

TP 3 : Utiliser Matplotlib

2 BCPST 2 - TP Informatique

CONSEIL : à chaque nouvelle question faites une nouvelle cellule en tapant "#%%" .

Vous pourrez ainsi traiter toutes les questions dans un même fichier.

Objectif du TP :

- Réviser le module matplotlib.pyplot - Savoir tracer des droites, des segments représenter des points.

Dans ce TP On revoit les bases du tracé de graphiques sur Python.

Le module `matplotlib` contient des sous-modules. Celui qu'on utilisera est `pyplot`. qu'on importe ainsi :

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

On rappelle les fonctions principales de ce module.

`plt.plot(X,Y,s,...) :`

Prend en entrée `X` une liste d'abscisses, `Y` une liste d'ordonnées **de même taille** que `X`. et `s` une indication de mise en forme.

La fonction va effectuer le tracé reliant par défaut les points dont les abscisses et ordonnées sont contenues dans les listes `X` et `Y`.

Pour `s`, on peut écrire avoir par exemple `s = "o-b"` La chaîne de caractère `"o-b"` contient trois informations sur la manière de tracer la courbe :

- La forme des points (o x . , v * _)
- La forme des lignes (- - - - . :)
- La couleur (r g b c m y k w)

On peut ajouter des informations complémentaires dans `plt.plot` du type `color = "pink"` si on veut une couleur précise, ou encore `linewidth = 0.5` si on veut changer l'épaisseur de trait de la courbe.

`plt.show() :`

Indispensable dès que vous voulez afficher un graphique. Prenez l'habitude de le mettre à la fin de vos codes. C'est requis aux concours. **N'oubliez pas les parenthèses** .

`plt.axis("equal") :`

Permet de rendre les axes orthonormés.

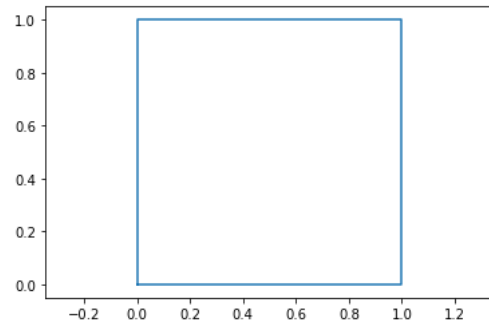
`plt.xlim(xmin, xmax)/ plt.ylim(ymin, ymax) :`

Permet de changer les limites du repère tracé. `xmin` est l'abscisse minimale affichée, `xmax` est l'abscisse maximale.

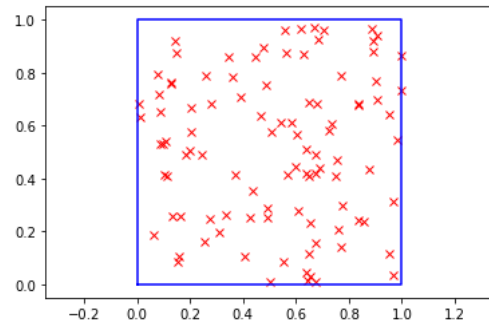
I Exercice 1 : Tracer un carré

Q.1. Niveau 0 : On propose de tracer un carré en utilisant le module `matplotlib`.

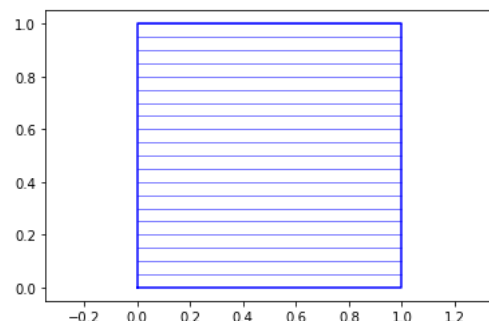
Astuce : N'oubliez pas de taper `plt.axis("equal")` pour avoir un graphe orthonormé et donc afficher un carré régulier.



Q.2. Niveau 1 : En utilisant le module `random` et la fonction `rd.random()` qui retourne un nombre aléatoire entre 0 et 1, afficher une trentaine de points de coordonnées aléatoires dans ce carré.

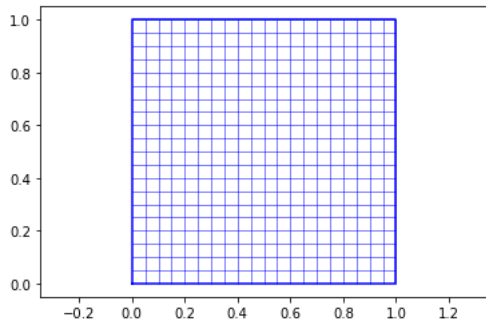


Q.3. Niveau 2 : On propose maintenant de hachurer verticalement le carré. Pour cela on pourra effectuer une boucle qui effectue un par un les segments qui constituent les hachures. Voir l'exemple ci-dessous :

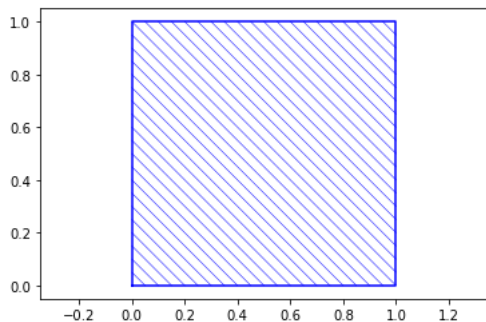


Astuce : Vous pouvez jouer avec la commande `linewidth = ..` en paramètre du plot pour rendre les hachures plus fines.

Q.4. Niveau 2.5 : On propose maintenant de quadriller le carré. Comme sur l'exemple ci dessous :



Q.5. Niveau 3 : On propose maintenant de hachurer le carré avec des hachure diagonales. Comme sur l'exemple ci dessous :



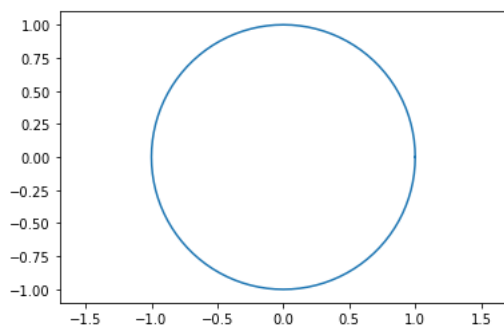
II Exercice 2 : Tracer un cercle

Pour cet exercice, on va utiliser le module numpy qui contient les fonctions `cos` et `sin` et aussi le nombre `pi`. Pour tracer le cercle de centre 0 et de rayon 1, il suffit de remarquer que c'est le cercle trigonométrique. C'est à dire que les points de ce cercle sont ceux de coordonnée $(\cos \theta, \sin \theta)$ pour θ qui varie de 0 à 2π .

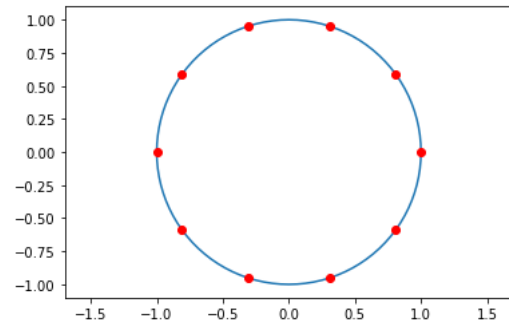
Il faut donc N angles $(\theta_i)_{0 \leq i \leq N}$ qui sont équitablement répartis entre 0 et 2π .

Ensuite on crée la liste d'abscisses `X` contenant les cosinus de ces angles, la liste de `Y` contenant les sinus.

Q.6. Afficher un cercle en utilisant la méthode expliquée ci-dessus :



Q.7. On propose ensuite sur ce cercle d'afficher n points régulièrement espacés, en utilisant la même méthode que celle expliquée ci-dessus, mais avec moins de points. Par exemple le résultat obtenu pour $n = 10$:



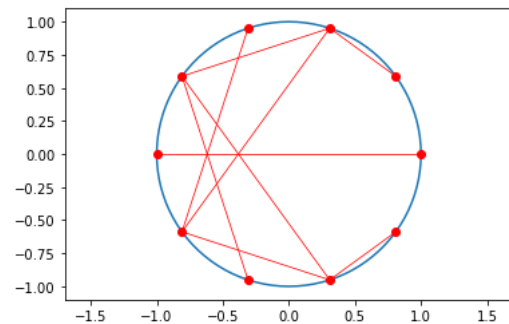
III Exercice 3 : La face cachée des tables de multiplications

Dans cet exercice, on part de la figure obtenue à la question 7. On va l'utiliser pour représenter une table de multiplication. L'idée est la suivante : chaque point rouge représente un nombre entre 0 et $n - 1$. Pour représenter la table de deux, on va simplement relier le point numéro i avec le point numéro $2i$ en traçant un segment. Ainsi le point numéro 1 sera relié au point numéro 2, le point numéro 2 sera relié au point numéro 4 etc...

Quand on dépasse la limite, cela fonctionne comme une horloge, c'est à dire que le point numéro 6 doit être relié au 12, mais 12 n'existe pas, donc on relie le point numéro 6 au point numéro 2.

Astuce : Cela revient à faire le reste dans la division euclidienne par 10 qui s'obtient avec " `12 % 10` "

Q.8. Effectuer le graphique représentant la table de deux en suivant la méthode ci dessus. Par exemple, pour $n = 10$, vous devez obtenir ce graphique :



Q.9. Augmenter la valeur de n , et cesser d'afficher les points en rouge sur le cercle (il y en aura trop) afin d'observer la belle figure que vous donne la table de deux. Puis faites de même avec la table de 3, de 4...

IV Exercice 4 : Des étoiles

On dit qu'un nombre complexe z est une racine $n^{\text{ième}}$ de l'unité si et seulement si $z^n = 1$.

On pose $n \in \mathbb{N}^*$ un entier fixé et on note $z_k = e^{i \frac{2k\pi}{n}}$.

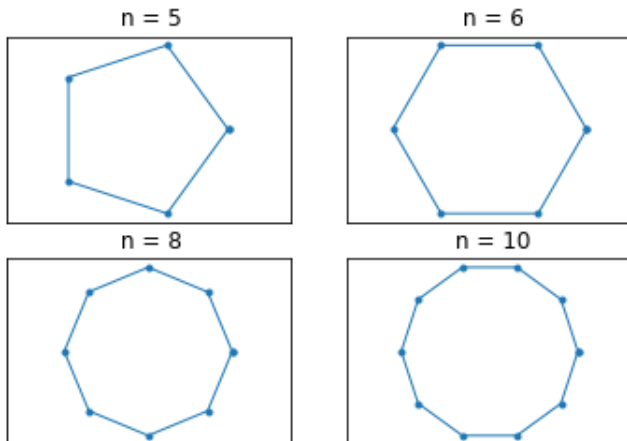
Q.10. On propose de coder un algorithme permettant de représenter graphiquement un tétraèdre régulier :

10.1) **(Python)** Ecrire une fonction python `abscisses(n)` qui retourne la liste `X` des abscisses des points dont les affixes sont les racines $n^{\text{ième}}$ de l'unité ($z_0, z_1, \dots, z_n$)

De même écrire une fonction `ordonnées(n)` qui retourne la liste `Y` des ordonnées des racines $n^{\text{ième}}$ de l'unité

Q.11. Coder une fonction python `polygone_regulier(n)` qui prend en entrée un entier `n` et qui trace un polygone régulier à `n` côtés en utilisant le module `matplotlib.pyplot`.

Q.12. Ecrire un code permettant d'afficher le graphique ci dessous à l'aide de la commande `subplot` :



On souhaite maintenant aller plus loin et utiliser ces fonctions pour représenter graphiquement des étoiles :

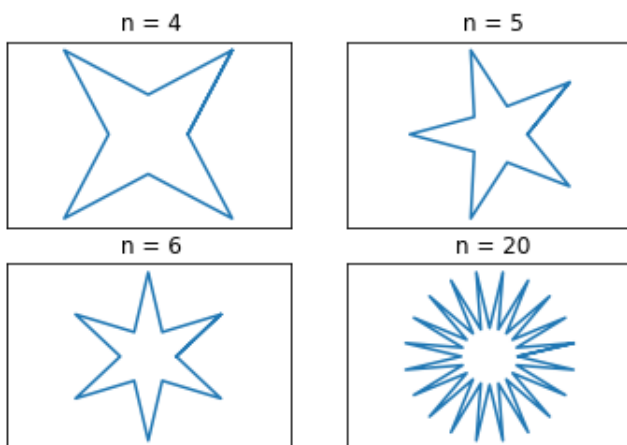


Figure 1

Pour ce faire, on va décortiquer un peu la construction d'une étoile. Il s'agit de remarquer simplement, comme ci-dessous que les points "centraux" forment le même motif que les points extérieurs. Il y a en plus une rotation et une augmentation de taille.

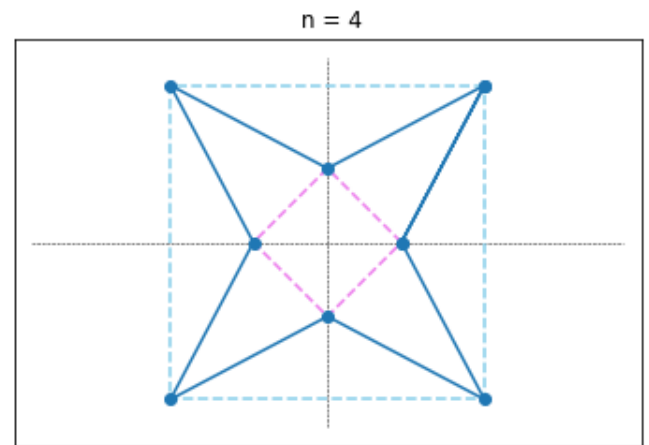


Figure 2

Pour $n = 4$, le carré central est le carré obtenu par la fonction `polygone_regulier(4)`, le carré extérieur est obtenu en faisant une rotation d'angle $\frac{\pi}{4}$ et en augmentant le rayon de distance des points par rapport à l'origine (ici le rayon a été triplé).

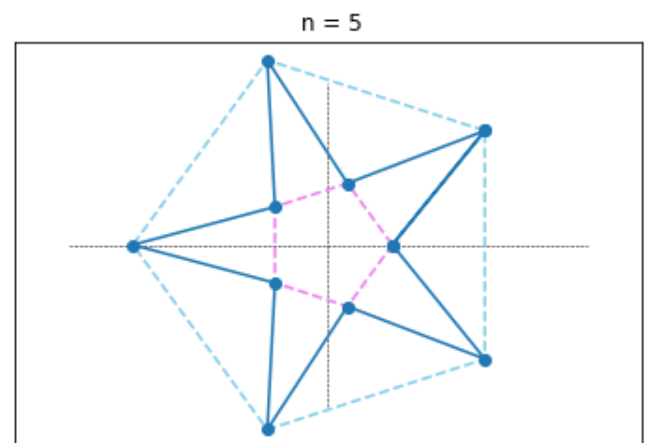


Figure 4

Pour $n = 5$, le pentagone central est obtenu par la fonction `polygone_regulier(5)`, le pentagone extérieur est obtenu en faisant une rotation d'angle $\frac{\pi}{5}$ et en augmentant le module des nombre complexes (ici le module a été triplé).

Q.13. On note $z = a + ib$ et $f(z) = 3e^{\frac{i\pi}{n}} z$.

13.1) Justifier que l'application f associe à z le nombre complexe $f(z)$ obtenu après avoir fait une rotation d'angle $\frac{\pi}{n}$ et un triplage du module.

13.2) Calculer les coordonnées de $f(z)$ sous forme algébrique.

13.3) **(Python)** Coder une fonction python `abcsisses_rot(n)` qui prend en entrée l'entier `n` et qui retourne la liste `Xr` des parties réelles de $f(z_0), f(z_1), \dots, f(z_n)$.

De même coder la fonction python `ordonnees_rot(n)` qui prend en entrée l'entier `n` et qui retourne la liste `Yr` des parties imaginaires de $f(z_0), f(z_1), \dots, f(z_n)$

Q.14. On va maintenant construire les étoiles :

14.1) **(Python)** Coder une fonction `insertion(L1,L2)` qui prend en entrée deux listes de même taille `L1` et `L2` et qui retourne une nouvelle liste qui contient en alternance les données de `L1` et `L2`. Exemple :

```
L1 = [1,1,1,3] ; L2 = [2,2,2,5]
print(insere(L1,L2))
-----
-> [1,2,1,2,1,2,3,5]
```

14.2) **(Python)** On coder deux fonctions `abcsisses_etoile(n)` et `ordonnees_etoile(n)` qui respectivement retournent la liste `Xe` des abscisses et `Ye` des ordonnées des figures en forme d'étoiles.

14.3) **(Python)** Coder une fonction `etoile(n)` qui permet de représenter graphiquement une étoile à n branches. Ecrire ensuite un code permettant d'afficher plusieurs étoiles comme sur la Figure 2.