MP/PC-PC*

Informatique

Devoir (auto)surveillé N°1

Le barème est fourni à titre indicatif.

Dans ce sujet, on suppose que toutes les fonctions Python demandées sont écrites dans un script comportant la ligne :

```
import numpy as np
```

Si vous êtes amenés à utiliser tout ou parties de certains modules Python, penser à ajouter les instructions d'importation appropriées.

Par ailleurs, vos fonctions ne chercheront pas à valider la correction des arguments passés.

Si vous êtes amenés à introduire des variables, utilisez des noms explicites et/ou ajouter des commentaires permettant de les identifier facilement. De façon générale, pensez à limiter le nombre de variables utilisées dans vos codes.

Exercice I : autour des entiers de Gauss (15 points)

Un complexe est dit « entier de Gauss » s'il est de la forme : $a+ib$ avec $(a,b)\in\mathbb{Z}^2$.

Pour manipuler de tels nombres sous Python, on a défini un nouveau type d'objets, les `GaussInt`, en écrivant une nouvelle classe dont voici les premières lignes (méthodes `__init__` et `__repr__`) :

```
class GaussInt:
	"""Classe permettant de manipuler des objets du type 'entiers de Gauss',
	c'est à dire des complexes de la forme a+ib où a et b sont deux entiers
	relatifs.
	"""
	def __init__(self,a,b):
		if type(a) != int or type(b) != int:
			raise TypeError("Vous devez fournir des entiers en arguments !")
		else:
			self.re = a
			self.im = b

	def __repr__(self):
		return("("+str(self.re)+","+str(self.im)+")")
```

[Q.1 / 1 point] Dans une console, on saisit les deux instructions :

```
n = GaussInt(4,-7)
n
```

Décrire l'affichage obtenu.

[Q.2 / 2 points] Surcharger la méthode `__neg__` de façon à pouvoir obtenir l'opposé d'un entier de Gauss en utilisant la syntaxe :

```
-n
```

On a surchargé les méthodes `__add__` et `__radd__` :

```
def __add__(self,n):
	if type(n) != GaussInt and type(n) != int:
		raise TypeError("Vous ne pouvez ajouter à un entier de Gauss qu'un
		entier ou un autre entier de Gauss !")
	elif type(n) == GaussInt:
		return(GaussInt(self.re + n.re,self.im + n.im))
	else:
		return(GaussInt(self.re + n,self.im))

def __radd__(self,n):
	if type(n) != int:
		raise TypeError("Vous ne pouvez ajouter un entier de Gauss qu'à un
		entier ou… un autre entier de Gauss ! (cf. méthode __add__)")
	else:
		return(self + n)
```

[Q.3 / 6 points] Dans la classe `GaussInt`, surcharger les méthodes `__sub__`, `__rsub__`, `__mul__` et `__rmul__`.

A l'entier de Gauss $n = a + ib$, on peut associer sa norme $N(n) = a^2 + b^2 = |n|^2$.

[Q.4 / 2 point] Compléter la classe `GaussInt` en écrivant la méthode `norme` renvoyant la norme de l'entier de Gauss considéré.

La norme précédente, bien qu'il ne s'agisse pas d'une norme euclidienne, permet de définir une division euclidienne sur l'ensemble des entiers de Gauss. En effet, pour tous entiers de Gauss *n* et *m*, il existe deux entiers de Gauss *q* et *r* tels que :

$$n = mq + r \text{ avec } N(r) < N(m)$$

Le couple (q, r) n'est cependant pas unique.

Si on écrit $\frac{n}{m} = \alpha + i\beta$, les valeurs possibles de l'entier de Gauss *q* sont à chercher parmi les quatre valeurs suivantes :

$$\mathrm{E}(\alpha) + i \times \mathrm{E}(\beta),\ (\mathrm{E}(\alpha)+1) + i \times \mathrm{E}(\beta),\ \mathrm{E}(\alpha) + i \times (\mathrm{E}(\beta)+1) \text{ et } (\mathrm{E}(\alpha)+1) + i \times (\mathrm{E}(\beta)+1)$$

Par exemple avec $n = 27 - 15i$ et $m = -3 - 4i$, on a :

$$\frac{n}{m} = \frac{27-15i}{-3-4i} = \frac{(27-15i)(-3+4i)}{(-3-4i)(-3+4i)} = \frac{(-81+60)+(108+45)i}{9+16} = \frac{-21+153i}{25}$$

On a alors : $\frac{n}{m} = \alpha + i\beta = \frac{-21}{25} + \frac{153}{25}i$. D'où $\mathrm{E}(\alpha) = \mathrm{E}\left(\frac{-21}{25}\right) = -1$ et $\mathrm{E}(\beta) = \mathrm{E}\left(\frac{153}{25}\right) = 6$.
Les valeurs possibles de l'entier de Gauss *q* sont donc à chercher parmi :

$$-1+6i,\ 6i,\ -1+7i \text{ et } 7i$$

La condition $N(r) < N(m)$, c'est-à-dire $N(n - mq) < N(m)$, conduit alors à ne conserver que les trois valeurs suivantes :

$$-1+6i,\ 6i \text{ et } -1+7i$$

[Q.5 / 4 points] Compléter la classe `GaussInt` en surchargeant l'opérateur // (méthode `__floordiv__`) afin qu'il renvoie une liste de tuples correspondant à la division euclidienne de deux entiers de Gauss.
Par exemple avec `a=GaussInt(27,-15)` et `b=GaussInt(-3,-4)`, la syntaxe `a//b` devra renvoyer la liste :

```
[(-1,6),(0,6),(-1,7)]
```

Exercice II : matrices triangulaires (25 points)

Dans cet exercice, on s'intéresse, sauf mention contraire (dernière question), aux matrices triangulaires.

[Q.1 / 3 points] Ecrire une fonction Python `TriMat_gen` qui reçoit en argument un entier n renvoie un tableau `numpy` correspondant à la matrice suivante :

$$M = \begin{pmatrix} 1 & 2 & 3 & \dots & n \\ 0 & n+1 & n+2 & \dots & 2n-1 \\ 0 & 0 & 2n & \dots & 3n-3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \frac{n(n+1)}{2} \end{pmatrix}$$

[Q.2 / 4 points] Ecrire une fonction `TriMat_test` qui reçoit en argument un tableau numpy `T` et renvoie :

- Le booléen `False` si le tableau `T` ne correspond pas à une matrice carrée ou à une matrice triangulaire.
- La chaîne de caractère `TriSup` (respectivement `TriInf`) si le tableau `T` correspond à une matrice triangulaire supérieure (respectivement inférieure).

[Q.3 / 2 points] Ecrire une fonction Python `TriMat_det` qui reçoit en argument un tableau numpy `T` correspondant à une matrice triangulaire et renvoie son déterminant (on rappelle que le déterminant d'une matrice triangulaire est égal au produit de ses éléments diagonaux).

Pour la question Q.3, on n'utilisera bien sûr pas la fonction `det` du module `linalg`… ☺

[Q.4 / 4 points] Ecrire une fonction Python `TriMat_transpose` qui reçoit en argument un tableau numpy `T` correspondant à une matrice triangulaire et renvoie un autre tableau numpy correspondant à sa transposée.

Pour la question Q.4, on n'utilisera bien sûr pas la méthode `transpose` des tableaux `numpy`… ☺ (again…)

[Q.5 / 5+2 points] Ecrire une fonction `TriMat_inverse` qui reçoit en argument un tableau numpy `T` correspondant à une matrice triangulaire et :

- teste l'inversibilité de `T`.
- renvoie le booléen `True` et la matrice T^{-1} ou seulement le booléen `False` suivant que `T` correspond ou non à une matrice inversible.

Evaluer la complexité (nombre d'opérations) du calcul de T^{-1}.

On souhaite désormais calculer, pour toute matrice diagonale D, une matrice diagonale approchant $e^{\mathrm{D}} = \sum_{k=0}^{+\infty} \frac{1}{k!} \mathrm{D}^k$.

On rappelle que si on note $\mathrm{D} = \left(d_{ij}\right)_{(i,j)\in[\![1,n]\!]^2}$ alors les coefficients diagonaux de e^{D} seront les $\exp\left(d_{ii}\right)$ avec $i \in [\![1,n]\!]$.

On suppose également que l'<u>on ne dispose pas d'une fonction exponentielle</u> mais que l'on a : $\forall x \in \mathbb{R},\ \lim_{p\to+\infty} \left(1+\frac{x}{p}\right)^p = e^x$. On montre alors que, pour tout réel *x* de l'intervalle $[0,1]$ et tout entier naturel *n* non nul, si on approche e^x par $\left(1+\frac{x}{2^n}\right)^{2^n}$ on commet une erreur $r_n(x) < \frac{e}{2^{n+1}}$ avec $e^x = \left(1+\frac{x}{2^n}\right)^{2^n}\left(1+r_n(x)\right)$ et *e* la base des logarithmes népériens.

On suppose que l'on dispose d'une constante Python : `E=2.718281828459045`. On s'en servira comme base des logarithmes népériens.

[Q.6 / 5 points] Ecrire une fonction `Diag_exp` qui reçoit en argument un tableau numpy `D` correspondant à une matrice diagonale ne comportant sur la diagonale que des coefficients dans l'intervalle $[0,1]$ et un flottant `epsilon` correspondant à un réel strictement positif ε et qui renvoie un tableau numpy dont le *i*-ème élément diagonal sera une approximation de $\exp\left(d_{ii}\right)$ avec une erreur $r_n\left(\exp\left(d_{ii}\right)\right)$ inférieure à ε.

ANNEXE

Quelques fonctions utiles du module `math` de Python :

`floor(x)` : renvoie la partie entière de `x`. Le résultat est un entier.
Par exemple `floor(-4.7)` renvoie `-5`.

`log(x,y)` : renvoie le logarithme de base `y` de `x`. Le résultat est un flottant.
Par exemple `log(8,2)` renvoie `3.0`.

Quelques fonctions utiles de la bibliothèque `numpy` :

`zeros((n,p))` : renvoie un tableau correspondant à la matrice nulle comportant `n` lignes et `p` colonnes.

`reshape(L,(n,p))` : renvoie un tableau construit à partir de la liste `L` et comportant `n` lignes et `p` colonnes. Par exemple, avec `L=[1,2,3,4,5,6]` et `A=np.reshape(L,(2,3))`, on obtiendra :

```
A=array([[1,2,3],
         [4,5,6]])
```

`dot(A,B)` : renvoie un tableau numpy correspondant au produit matriciel des tableaux A et B.

Fin du sujet