Correction du DS 3

Informatique pour tous, première année

Julien Reichert

Exercice 1

Voir cours, on autorise les méthodes des rectangles et des trapèzes. Il y a une certaine tolérance pour les méthodes d'ordre supérieur (Simpson, etc.) ou pour des méthodes de résolution d'équations différentielles appliquées à une équation de la forme $y'(x) = f(x)$.

Exercice 2

La complexité de la méthode du pivot pour la matrice d'un système (n, n) auquel on a éventuellement accolé k colonnes de n lignes est un $\mathcal{O}(n^2(n+k))$. Le fait de faire une recherche systématique du meilleur pivot possible (« pivot partiel ») ne change rien à la complexité asymptotique, car on remplace dans le calcul un $\mathcal{O}(n)$ (valant souvent 1) par un $\Theta(n)$ qui est de toute façon négligeable devant le $\mathcal{O}(n(n+k))$ auquel il est finalement additionné.

Exercice 3

L'erreur est naturellement à la ligne 8, car le déclenchement d'exception ne saurait être systématique (`j` est forcément inférieur ou égal à `n`), il faut tester si `j == n`.

Exercice 4

```
def aplatir(listedelistes):
    reponse = []
    for liste in listedelistes:
        reponse.extend(liste) # ou append avec une boucle parcourant la liste
```

À ce sujet, pour ne pas s'encombrer l'esprit, on peut transiter par la structure de matrice et l'aplatir par la force des choses, mais le coût est doublé et surtout le changement de structure est presque toujours sale. L'exception tolérée jusqu'ici était le passage d'un nombre à une chaîne de caractères.

Bref, un regard noir à ceux qui inventeraient le code suivant :

```
from numpy import array, append
def aplatircommeunfragile(listedelistes):
    return list(append(array(listedelistes),42)[:-1])
```

Exercice 5

Pour simuler `arange(deb,fin,pas)` avec un `linspace`, on a besoin de calculer le nombre d'éléments, qui est exactement $\lceil \frac{\texttt{fin-deb}}{\texttt{pas}} \rceil$, et le dernier élément, qui est exactement `deb` plus le nombre d'éléments moins un (soit le nombre d'intervalles, en fait).

Pour simuler `linspace(deb,fin,nombre)` avec un `arange`, on a besoin de calculer le pas, qui est exactement $\frac{\texttt{fin-deb}}{\texttt{nombre-1}}$, et une nouvelle fin exclue, qui doit être strictement supérieure à `fin` mais inférieure ou égale à `fin + pas`. On peut prendre par exemple `fin + pas/42` (mais la division par `42` n'est pas optimisée...).

```
from math import ceil

def arange_to_linspace(deb,fin,pas):
    n = ceil((fin-deb)/pas)
    from numpy import linspace # importation ici afin que la fonction ne dépende pas
        # d'un changement ailleurs dans la façon d'importer
    return linspace(deb,deb+(n-1)*pas,n)

def linspace_to_arange(deb,fin,n):
    pas = (fin-deb)/(n-1)
    from numpy import arange
    return arange(deb,fin+pas/2,pas)
# le / garantit que l'un des paramètres sera un flottant,
# donc le résultat sera un tableau de flottants
```