Module C1

Calcul scientifique avec Python 3

Timothée Duruisseau

Department de génie mécanique
École Polytechnique Montréal
Canada
27 août 2021

Objectifs du module

À la fin du module, il sera important de savoir calculer des fonctions, trouver les intersections d'une fonction et faire des dérivations et des intégrations numériques. De plus, il est primordial que le traçage de courbes 2D soit bien compris car les résultats devront souvent être affichés adéquatement.

Table des matières

Chapitre 1

Chapitre 1 : Calculer une fonction

1.1 Introduction

La première chose à savoir pour faire du calcul scientifique est comment calculer des fonctions. Comme dans tous langages de programmation, il faut distinguer entre une fonction mathématique, qui établie une relation entre des ensembles de variables, et une fonction de progammation. À partir de maintenant, les fonctions mathématiques seront explicitées. Les fonctions de programmation seront adressées simplement comme des fonctions ou des méthodes, selon le cas.

Selon le style de programmation et la taille des projets fait, les fonctions mathématiques peuvent être calculées directement dans une variable ou peuvent être calculer à l'intérieur d'une fonction créé par l'usager. Le principe reste le même, la différence étant surtout le nombre de fois que la fonction mathématique doit être calculer. Aussi, bien qu'il soit possible de faire le calcul de fonctions mathématiques en utilisant des listes et des tuples, cela demande l'utilisation de boucles itératives. Dans un souci d'alléger les codes et de simplifier le traitement de données, les vecteurs et matrices seront préférés, permettant de faire les calculs sur l'ensemble des données en une seule commande.

1.2 Calculer une fonction mathématique simplement

La première manière de calculer une fonction mathématique est de simplement écrire l'expression dans une variable. Par exemple, la fonction mathématique $f(x) = 3x + 2$ peut tout simplement être écrite de cette manière : `f = 3*x+2`, avec `x` préalablement définie comme étant un nombre, un vecteur ou une matrice. La variable dépendante `f` sera généralement du même type que la variable indépendante `x`. Un changement peut être automatique fait dans le cas où `x` est un entier et la fonction mathématique génère un nombre flottant ou un nombre complexe.

Cette manière de calculer est très souvent utilisé lorsque la fonction mathématique n'a besoin d'être calculer qu'une seule fois, par exemple dans une console. C'est un peu l'équivalent d'utiliser une calculatrice. Lorsque la fonction mathématique doit être utilisée plusieurs fois, cela devient lassant à écrire et difficile à maintenir.

Le script 1 est un exemple de calcul d'une fonction mathématique simple. La figure 1.1

SCRIPT 1
Démonstration de calcul d'une fonction mathématique

```
# Importation de modules
import numpy as np
import matplotlib.pyplot as plt

# Génération des données indépendante
x = np.linspace(0,9,num=10)

# Calcul de f(x) = 3*x**1.5
f = 3*x**1.5

# Affichage du graphique
fig, ax = plt.subplots()
ax.plot(x,f,'-or')
ax.set_xlabel('axe x')
ax.set_ylabel('axe y')
ax.set_title('Démonstration graphique\nf(x) = 3*x**1.5')

plt.show()
```

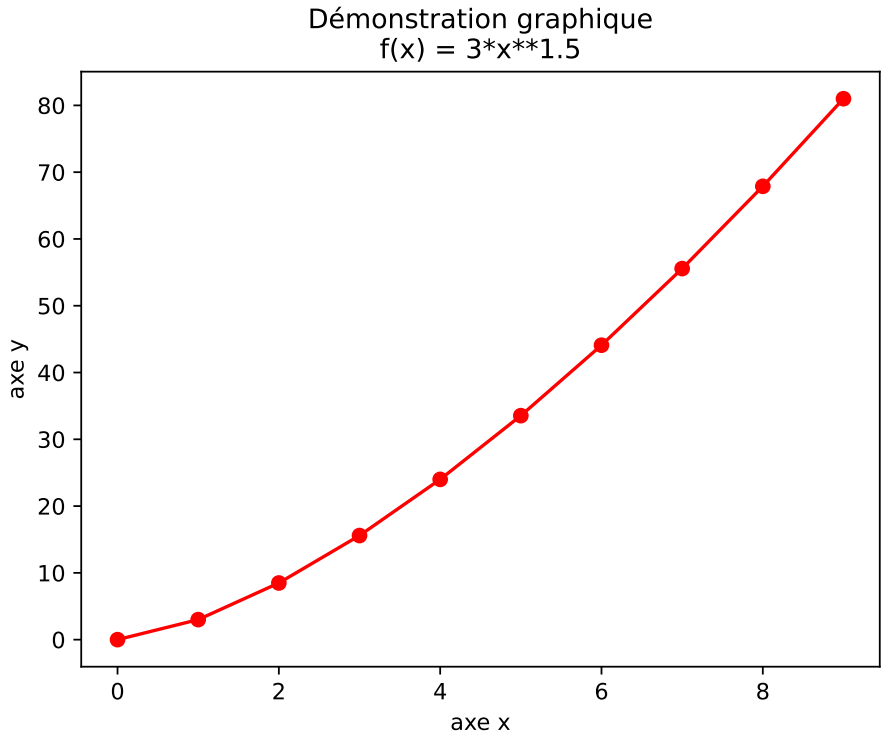

FIGURE 1.1 – Tracé de la fonction mathématique $f(x) = 3x^{1.5}$, résultat du script 1

1.3 Créer une fonction pour calculer

Tel que discuté dans la section précédente, réécrire plusieurs fois la même fonction n'est pas pratique pour coder rapidement et faire la maintenance du code. La manière adéquate est donc de créer une fonction, avec le mot-clé `def`, qui sera ensuite appelée pour faire le calcul. Ainsi, la fonction mathématique elle-même n'est écrite qu'une seule fois et donc si un changement doit être fait, il n'y a qu'une seule ligne à changer. De plus, selon la longueur de l'expression et du nom de la fonction, il peut être beaucoup plus rapide d'écrire le nom que l'expression elle-même.

La fonction peut être écrite directement dans le script principal ou peut être écrite dans un fichier à part, qui sera ensuite importé dans le script principal. Pour importer un fichier de fonctions dans un script, il faut utiliser la même manière que l'importation d'un module. De plus, pour la simplification de l'importation, il faut s'assurer que le fichier de fonctions et le script se trouvent dans le même dossier.

La création d'une fonction de programmation se fait ainsi : `def nom_fonction(entrees):`. Chaque ligne suivante faisant partie de la fonction doit être indentée. La fin d'une fonction se fait par le mot-clé `return`. Ce qui est écrit à la suite de ce mot-clé est ce qui sera retourné dans l'appel. Dans le cas d'une expression mathématique, il est très courant que l'expression soit entièrement inscrite à la suite du mot-clé, par exemple `def fonc(y) : return 2*y/9`. Dans cet exemple, `y` est la variable indépendante en entrée, et sa valeur est directement référencée dans l'expression mathématique.

Il est également possible de transporter des paramètres dans les fonctions, en les ajoutant en entrées. Le transport de paramètres peut être fait de plusieurs manières différentes et dépendra de l'objectif du script.

Le script 2 démontre la définition d'une fonction pour faire le calcul d'une fonction mathématique. Le mot-clé utilisé pour la définition d'une fonction est `def`, le nom de la fonction est `func` et son argument d'entrée est `x`. L'argument qui sera mis en entrée doit être défini avant l'appel et peut avoir n'importe quel nom. Le nom utilisé dans la définition est pour l'utilisation interne de la fonction et donc n'impacte pas le reste du code. La figure 1.2 représente le résultat du calcul sous une forme graphique.

SCRIPT 2
Démonstration de la création d'une fonction et son utilisation

```
# Importation de modules
import numpy as np
import matplotlib.pyplot as plt

# Définition d'une fonction
def func(x):
    return 3*np.exp(x)/np.log(x+2)

# Génération des données indépendante
y = np.linspace(0,9,num=10)

# Appel de la fonction
f2 = func(y)

# Affichage du graphique
fig, ax = plt.subplots()
ax.plot(x,f2,'-or')
ax.set_xlabel('axe x')
ax.set_ylabel('axe y')
ax.set_title('Démonstration graphique\n f(x) = 3*e^x/log(x+2)')

plt.show()
```

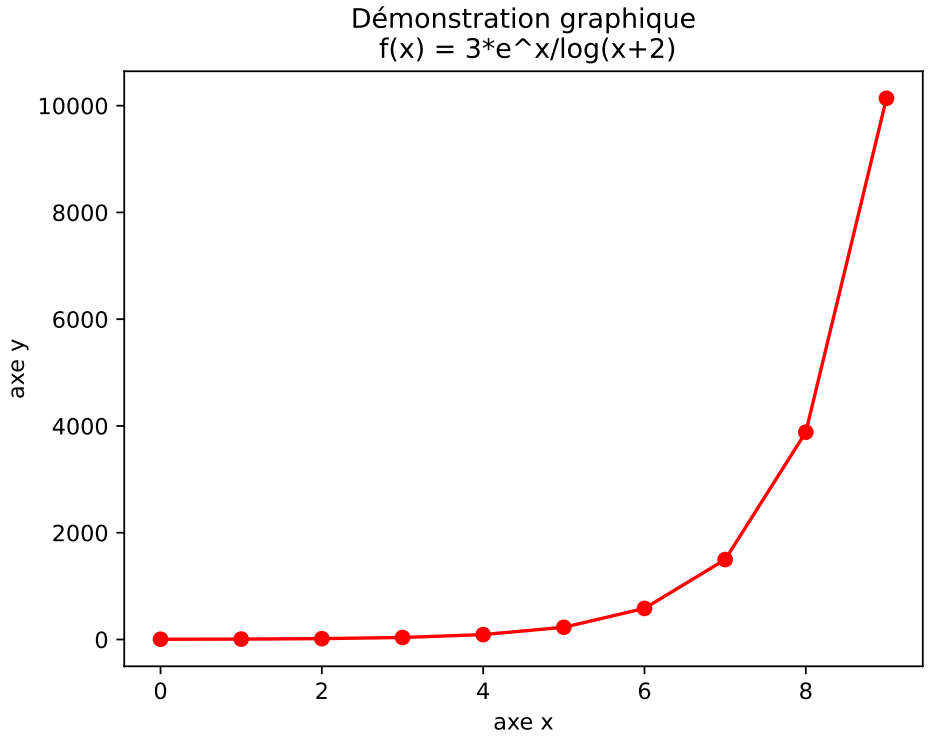

FIGURE 1.2 – Tracé de la fonction mathématique $f(x) = 3\frac{e^x}{\log(x+2)}$, résultats du script 2

Chapitre 2

Chapitre 2 : Trouver les intersections

2.1 Introduction

Il est souvent nécessaire de trouver les intersections d'une courbe, que ce soit les zéros ou les intersections avec une ou plusieurs autres courbes. Pour trouver ces intersections, il existe plusieurs algorithmes. L'algorithme qui sera utilisé ici est la méthode de la bissection. C'est une méthode simple, qui permet de trouver exactement 1 zéro, dont la position approximative est connue. La bissection peut être utilisée pour une fonction mathématique contenant plusieurs zéros, mais l'intervalle sur lequel la méthode sera appliqué ne doit pas contenir plus de 1 zéro.

2.2 Trouver un zéro

Commençons par trouver l'intersection d'une fonction mathématique quelconque avec l'axe horizontale avec la méthode de la bissection. Le principe de base de la bissection est d'évaluer si il y a un changement de signe dans un intervalle. Ainsi, si 2 extrémités d'un intervalle possède le même signe, le résultat de leur multiplication sera toujours positif, alors que si les extrémités sont de signes oposés, le résultat sera toujours négatif. Cette propriété mathématique peut donc être exploitée pour trouver l'intersection, à la condition que l'intervalle choisi ne contient qu'une seule intersection.

Une fois l'intervalle et la précision choisis, l'algorithme consiste en 4 opérations : déterminer le point milieu de l'intervalle, évaluer dans lequel des 2 sous-intervalles se trouvent le changement de signes, calculer l'étendue du sous-intervalle ayant le changement de signe et recommencer les opérations sur ce nouvel intervalle jusqu'à temps que l'étendue de l'intervalle soit inférieur à la précision choisie.

Pour cet algorithme, il est pertinent d'utiliser la boucle itérative `while` et la structure de contrôle `if`. La boucle `while` permet de vérifier si une évaluation supplémentaire est nécessaire, alors que la structure de contrôle `if` permet d'évaluer dans quel sous-intervalle se trouve le changement de signe.

2.2.1 Exemple

Le script 3 démontre un exemple de bissection.

SCRIPT 3
Exemple de bissection

```python
# Importation des modules
import numpy as np
import matplotlib.pyplot as plt

# Fonction mathématique
def fonc(x):
    return (x-3)**2 - 5

# Fonction de bissection
def bissection(x1,x2,tol):
    """ Fonction calculant un zéro dans un intervalle
    Entrées:
        - x1 : limite inférieure
        - x2 : limite supérieure
        - tol : critère d'arrêt

    Sortie:
        - Valeur approximative du zéro
    """
    etendu = 1

    # Boucle while pour itérer jusqu'à l'obtention du zéro
    while etendu > tol:
        pointMilieu = (x1 + x2)/2
        y1 = fonc(x1)
        y2 = fonc(x2)
        ymilieu = fonc(pointMilieu)
        if y1*ymilieu < 0:
            x2 = pointMilieu
        elif y2*ymilieu < 0:
            x1 = pointMilieu
        etendu = abs(x2-x1)
    return pointMilieu

x = bissection(0,5,1e-06)
print("Un zéro se trouve à la position x = %.04f" % x)
```

```
Un zéro se trouve à la position x = 0.7639
```

2.3 Intersections entre 2 courbes

2.3.1 Courbes quelconques

Pour deux courbes quelconques, la méthode de la bissection peut être utilisée. Cela dit, une modification s'impose. En effet, puisque l'intersection entre les 2 courbes n'est pas nécessairement sur l'axe horizontale, le changement de signe n'a plus de signification. Une fonction mathématique composée doit alors être utilisée pour que le changement de signe retrouve la signification recherchée.

Ainsi, soit deux courbes quelconques $h(x)$ et $g(x)$. L'intersection de ces deux courbes répond par définition à la condition $h(x) = g(x)$, ce qui revient à dire que $h(x) - g(x) = 0$. La fonction mathématique composée sur laquelle la méthode de la bissection sera faite est donc $f(x) = h(x) - g(x)$. Le reste des opérations demeurent les mêmes.

2.3.2 Courbes polynômiales

Pour certains problèmes, une solution analytique est possible. C'est le cas notamment d'un polynôme de degré 3 et moins et d'une droite. Supposons d'abord un polynôme représentant une droite,

$$P_1(x) = a_1 x + a_0$$

Pour un polynôme de degré 2, tel que

$$P_2(x) = b_2 x^2 + b_1 x + b_0$$

la résolution se fait avec la formule quadratique. Pour ce faire, on commence par soustraire les deux équations pour obtenir une seule expression,

$$P(x) = P_2(x) - P_1(x)$$

$$P(x) = b_2 x^2 + (b_1 - a_1)\, x + (b_0 - a_0)$$

$$P(x) = \alpha x^2 + \beta x + \gamma$$

Maintenant on applique la formule quadratique

$$r_{1,2} = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha}$$

Pour un polynôme de degré 3, tel que

$$P_3(x) = c_3 x^3 + c_2 x^2 + c_1 x + c_0$$

la résolution peut se faire avec la formule cubique. Comme pour le degré 2, il faut commencer par soustraire les deux équations,

$$P(x) = P_3(x) - P_1(x)$$

$$P(x) = c_3 x^3 + c_2 x^2 + (c_1 - a_1)\, x + (c_0 - a_0)$$

$$P(x) = \alpha x^3 + \beta x^2 + \gamma x + \epsilon$$

Il est pratique ensuite de calculer le déterminant Δ afin de savoir la nature des intersections. Un déterminant inférieur à 0 indique la présence d'une seule racine réelle et de 2 racines complexes. Un déterminant supérieur à 0 indique trois racines réelles distinctes. Un déterminant égale à 0 indique trois racines réelles, dont deux identiques. Le déterminant se calcul avec cette expression

$$\Delta = 18\alpha\beta\gamma\epsilon - 4\beta^3\epsilon + \beta^2\gamma^2 - 4\alpha\gamma^3 - 27\alpha^2\epsilon^2$$

La formule cubique à utiliser est

$$r_{1,2,3} = \sqrt[3]{q + \sqrt{q^2 + (r - p^2)^3}} + \sqrt[3]{q - \sqrt{q^2 + (r - p^2)^3}} + p$$

avec

$$p = \frac{-\beta}{3\alpha}$$

$$q = p^3 + \frac{\beta\gamma - 3\alpha\epsilon}{6\alpha^2}$$

$$r = \frac{\gamma}{3\alpha}$$

Chapitre 3

Chapitre 3 : Calculer une dérivée

3.1 Introduction

Dans le calcul scientifique, il est important de savoir dériver. Pour les cas où la fonction mathématique existe et est connue, il est triviale d'insérer l'expression dans le script. Cependant, ces cas demeurent rares et il est beaucoup plus courant d'avoir uniquement des ensembles de données. Il faut donc avoir recourt à des techniques numériques. La dérivation numérique se base sur le développement de Taylor d'une expression mathématique autour d'un point d'intérêt.

3.2 Dérivée première

La dérivée première est la dérivée décrivant la pente d'une courbe. Ainsi, la dérivée première se calcule en faisant le quotient de la différence de hauteur par la différence de largeur. Selon ce qui est disponible comme informations, il est possible de dérivée par en avant, par en arrière ou d'à partir du centre. L'ordre du calcul influence la précision des résultats.

3.2.1 Ordre de précision 1

La dérivation première d'ordre 1 peut être avant ou arrière. La dérivée avant permet de déterminer la pente à un point x avec le point en $x + \Delta x$, alors que la dérivée arrière permet de déterminer la pente à un point x avec le point en $x - \Delta x$.

Formule de dérivation avant d'ordre 1 :

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Formule de dérivation arrière d'ordre 1 :

$$f'(x) = \frac{f(x) - f(x - \Delta x)}{\Delta x}$$

3.2.2 Ordre de précision 2

La dérivation d'ordre 2 peut être fait par en avant, en arrière ou centrée.

Formule de dérivation avant d'ordre 2 :

$$f'(x) = \frac{-f(x + 2\Delta x) + 4f(x + \Delta x) - 3f(x)}{2\Delta x}$$

Formule de dérivation arrière d'ordre 2 :

$$f'(x) = \frac{f(x-2\Delta x) - 4f(x-\Delta x) + 3f(x)}{2\Delta x}$$

Formule de dérivation centrée d'ordre 2 :

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$$

3.3 Dérivée seconde

Alors que la dérivée première décrit la pente d'une courbe, la dérivée seconde décrit la concavité d'une courbe. Pour un ensemble de données de la position selon le temps, la dérivée seconde sera donc l'accélération.

Formule de dérivation seconde arrière d'ordre 1 :

$$f''(x) = \frac{f(x) - 2f(x-\Delta x) + f(x-2\Delta x)}{\Delta x^2}$$

Formule de dérivation seconde avant d'ordre 1 :

$$f''(x) = \frac{f(x) - 2f(x+\Delta x) + f(x+2\Delta x)}{\Delta x^2}$$

Formule de dérivation seconde centrée d'ordre 2 :

$$f''(x) = \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{\Delta x^2}$$

Chapitre 4

Chapitre 4 : Calculer une intégrale

4.1 Introduction

Tout comme la dérivation numérique, l'intégration numérique est une technique importante dans le calcul scientifique. En effet, il est très fréquent d'avoir uniquement des ensembles de données ou des expressions mathématiques dont la solution analytique est difficile ou carrément impossible à faire. Ainsi, il existe plusieurs techniques d'intégrations numériques. Afin de garder les choses simples, deux techniques seront présentées ici : les sommes de Riemann et la méthode des trapèzes composés.

4.2 Les sommes de Riemann

La méthode des sommes de Riemann fait partie des explications de base pour l'explication d'une intégrale. Elle consiste à faire la somme de rectangles dont les dimensions sont dictées par les points disponibles. La formule de base des sommes de Riemann est

$$\int_a^b f(x)dx = \lim_{N\to\infty} \sum_{i=1}^{N} f(x_i^*)(x_i - x_{i-1}), \quad x_i^* \in [x_{i-1}, x_i]$$

Le point x_i^* peut être choisi selon trois possibilités : à gauche, à droite ou centrée. La méthode à gauche prend la hauteur du rectangle au premier point de l'intervalle. La méthode à droite prend la hauteur du rectangle au deuxième point de l'intervalle. La méthode centrée prend la moyenne des hauteurs des 2 points de l'intervalle. La largeur de l'intervalle est strictement la distance horizontale entre les 2 points. La figure 4.1 montre la différence entre chaque méthode.

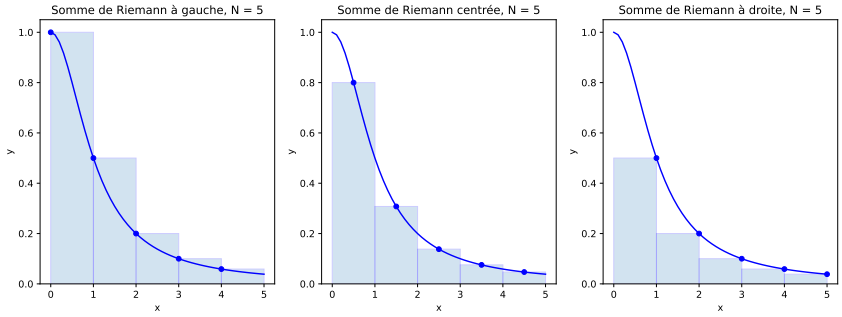

FIGURE 4.1 – Différences entre les méthodes de sommes de Riemann

Le paramètre N contrôle la précision de l'implémentation. Plus N est grand, plus le résultat de la somme de Riemann s'approchera du résultat analytique de l'intégrale. Ce paramètre est est égale au nombre d'intervalles. La figure 4.2 démontre la différence entre différent N pour une même fonction mathématique quelconque.

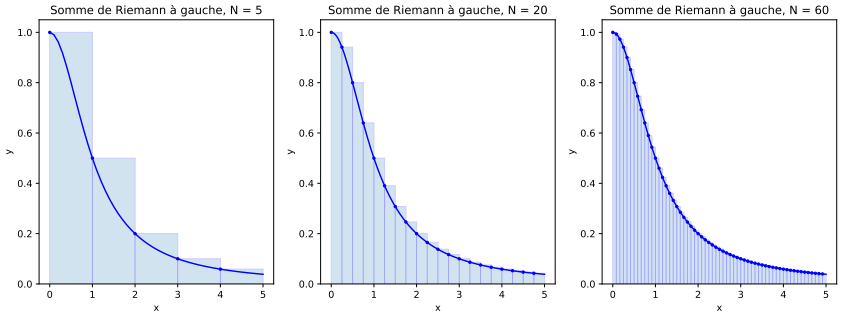

FIGURE 4.2 – Différences entre les méthodes de sommes de Riemann

4.2.1 Exemple

Le script 4 contient un exemple de sommes de Riemann à gauche.

SCRIPT 4
Exemple de code calculant une intégrale avec la méthode des sommes de Riemann à gauche

```python
# Importation des modules
import numpy as np

# Définition de la fonction de sommes de Riemann à gauche
def riemann_gauche(x,y):
    """Fonction calculant une somme Riemann gauche
    Entrées :
        - x : Valeurs indépendantes
        - y : Valeurs dépendantes

    Sortie :
        - Résultat de la somme
    """
    somme = 0
    for i in range(len(x)-1):
        somme += (x[i+1] - x[i])*y[i]

    return somme

# Création des données pour la démonstration
x = np.linspace(0,10,num=12)
y = x**2

# Appel de la fonction de sommes de Riemann à gauche
integrale = riemann_gauche(x,y)

# Affichage du résultat
print("Résultat de l'intégration numérique par sommes de Riemann à gauche : %.4f" %
    integrale)
```

```
Résultat de l'intégration numérique par sommes de Riemann à gauche : 289.2562
```

4.3 Trapèzes composés

La méthode des trapèzes composés ressemble beaucoup aux sommes de Riemann. La différence étant que la forme choisie est le trapèze au lieu du rectangle. La formule de base de la méthode des trapèzes est

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{N}(x_i - x_{i-1})\frac{f(x_{i-1}) + f(x_i))}{2}$$

Contrairement aux sommes de Riemann, cette méthode ne contient pas de variation directionnelle car les informations des 2 points sont utilisées. L'application de cette méthode est donc très rapide et constante à appliquer.

4.3.1 Exemple

Le script 5 démontre un exemple d'intégrale numérique par la méthode des trapèzes composés.

SCRIPT 5
Exemple d'intégrale par la méthode des trapèzes

```
# Importation des modules
import matplotlib.pyplot as plt
import numpy as np

# Fonction de calcul de trapèzes
def aire(f, n, x0, xf):
    xx=np.linspace(x0, xf, n)
    A = 0
    for i in range(0, n-1):#dans range le nbr (n-1) est exclu
        A = A + (f(xx)[i+1]+f(xx)[i])*(xx[i+1]-xx[i])/2

    return A

#Fonction désirée sinus
def h(x):
    y = np.sin(x)
    return y

# Génération des données
npt = 10
limite_1 = 0
limite_2 = np.pi
x=np.linspace(limite_1, limite_2, npt)
# appel de l'integrale numérique
Int_N = aire(h, npt, limite_1, limite_2)

valeurExacte = 2.0
erreur = abs(valeurExacte - Int_N)

print(Int_N)
F = h(x)

print('Integration numerique de sin(x) sur lintervalle')
print('x1 = {:.4f}, x2 = {:.4f}, avec le nombre des points = {:.4f}, donne = {:.4f},
    erreur= {:.4f}'.format(limite_1,limite_2,npt,Int_N,erreur))
```

```
1.9796508112164835
Integration numerique de sin(x) sur lintervalle
x1 = 0.0000, x2 = 3.1416, avec le nombre des points = 10.0000, donne = 1.9797, erreur=
    0.0203
```