Série 6 (Corrigé)

Pour mardi : apprentissage vidéos chapitres 3.4 (environ 40 minutes de travail)

Mardi prochain : résolvez les exercices Python qui se trouvent dans cette série

Partiellement en classe vendredi

Exercice 1

On considère une fonction $f(x)$, $x \in [a, b]$.

1. Écrivez le polynôme quadratique interpolant $f(x)$ dans les nœuds $x_0 = a$, $x_1 = \frac{a+b}{2}$ et $x_2 = b$ en utilisant la base de Lagrange.
2. Combien valent les intégrales $\int_a^b \varphi_k$ pour $k = 0, 1, 2$? Utilisez une formule de quadrature appropriée pour calculer !
3. On veut approcher l'intégrale de $f(x)$ par l'intégration de l'interpolation quadratique. Écrivez l'intégrale exacte de l'interpolant. Quelle formule d'intégration numérique pouvez-vous reconnaître ?

Sol. :

1. *Commençons par consideérer les polynômes de la base de Lagrange pour l'interpolation :*

$$\begin{aligned}
\varphi_0(x) &= \frac{\left(x-\frac{a+b}{2}\right)(x-b)}{\left(a-\frac{a+b}{2}\right)(a-b)} = 2\frac{\left(x-\frac{a+b}{2}\right)(x-b)}{(a-b)^2} \\
\varphi_1(x) &= \frac{(x-a)(x-b)}{\left(\frac{a+b}{2}-a\right)\left(\frac{a+b}{2}-b\right)} = -4\frac{(x-a)(x-b)}{(a-b)^2} \\
\varphi_2(x) &= \frac{(x-a)\left(x-\frac{a+b}{2}\right)}{(b-a)\left(b-\frac{a+b}{2}\right)} = 2\frac{(x-a)\left(x-\frac{a+b}{2}\right)}{(b-a)^2}
\end{aligned}$$

Où on a utilisé le fait que $\frac{a+b}{2}$ est le point milieu de $[a, b]$ et que $b - \frac{a+b}{2} = \frac{b-a}{2}$ et $a - \frac{a+b}{2} = \frac{a-b}{2}$. Par conséquent, l'interpolation quadratique peut être écrite comme :

$$\begin{aligned}
\Pi_h^2 f(x) = &\quad f(a)\varphi_0(x) + f(\tfrac{a+b}{2})\varphi_1(x) + f(b)\varphi_2(x) \\
= \tfrac{1}{(b-a)^2} &\quad \left(2f(a)\left(x-\tfrac{a+b}{2}\right)(x-b) - 4f(\tfrac{a+b}{2})(x-a)(x-b) + 2f(b)(x-a)\left(x-\tfrac{a+b}{2}\right)\right)
\end{aligned}$$

2. *On opère le changement de variable $x = \frac{a+b}{2} + t\left(\frac{b-a}{2}\right)$ et on obtient :*

$$\int_a^b \varphi_k(x)dx = \frac{b-a}{2}\int_{-1}^1 \varphi_k\left(\frac{a+b}{2} + t\left(\frac{b-a}{2}\right)\right)dt$$

$\hat{\varphi}_k(t) = \varphi_k\left(\frac{a+b}{2} + t\left(\frac{b-a}{2}\right)\right)$ *est un polynôme de degré deux, il est intégré exactement sur* $[-1,1]$ *par la formule de quadrature de Simpson :*

$$\int_{-1}^{1} \hat{\varphi}_k(t)dt = \frac{2}{6}\left(\hat{\varphi}_k(-1) + 4\hat{\varphi}_k(0) + \hat{\varphi} + (1)\right) =$$

donc

$$\int_a^b \varphi_k(x)dx = \frac{2}{6}\frac{b-a}{2}\left(\varphi_k(a) + 4\varphi_k\left(\frac{a+b}{2}\right) + \varphi_k(b)\right)$$

Mais $a, b, \frac{a+b}{2}$ *sont les noeuds d'interpolation, donc* φ_k *dans ces points vuax 1 ou 0. Plus précisement :*

$$\begin{array}{rcl} \int_a^b \varphi_0(x)dx & = & \frac{2}{6}\frac{b-a}{2}\left(\varphi_0(a) + 4\varphi_0\left(\frac{a+b}{2}\right) + \varphi_0(b)\right) = \frac{b-a}{6} \\ \int_a^b \varphi_1(x)dx & = & \frac{2}{6}\frac{b-a}{2}\left(\varphi_1(a) + 4\varphi_1\left(\frac{a+b}{2}\right) + \varphi_1(b)\right) = \frac{2}{3}(b-a) \\ \int_a^b \varphi_2(x)dx & = & \frac{2}{6}\frac{b-a}{2}\left(\varphi_2(a) + 4\varphi_2\left(\frac{a+b}{2}\right) + \varphi_2(b)\right) = \frac{b-a}{6} \end{array}$$

3. *Si on combine les résultats des deux points précédents on obtient :*

$$\begin{aligned} \int_a^b \Pi_h^2 f(x) &= \int_a^b \Big(f(a)\varphi_0(x) + f\left(\frac{a+b}{2}\right)\varphi_1(x) + f(b)\varphi_2(x)\Big)dx \\ &= f(a)\int_a^b \varphi_0(x)dx + f\left(\frac{a+b}{2}\right)\int_a^b \varphi_1(x)dx + f(b)\int_a^b \varphi_2(x)dx \\ &= \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) \end{aligned}$$

Nous reconnaissons (que c'est) la formule de Simpson pour l'intégration numérique sur un intervalle quelconque.

Exercice 2

Soit g une fonction continue définie sur l'intervalle $[-1,1]$. On choisit trois nœuds d'interpolation nommés x_1, x_2 et x_3 tels que $x_1 = -1$, $x_2 = \beta$ et $x_3 = 1$, où β est un nombre réel donné tel que $|\beta| < 1$. Pour approcher l'intégrale $\int_{-1}^{1} g(x)dx$, on considère la formule de quadrature suivante :

$$I_2(g) = \sum_{j=0}^{2} \alpha_j\, g(x_j) = \alpha_0\, g(-1) + \alpha_1\, g(\beta) + \alpha_2\, g(1)$$

a) Trouvez les poids α_0, α_1 et α_2 en fonction de β tels que la formule de quadrature soit exacte de degré 2.

b) Trouvez ensuite β tel que $I_2(p) = \int_{-1}^{1} p(x)dx$ pour tout polynôme p de degré 3.

c) Reécrivez cette formule sur l'intervalle $[a,b]$ $(a < b)$ pour intégrer une fonction continue f définie sur cet intervalle. Quelle formule d'intégration reconnaissez-vous ?

Sol. :

a) Notons d'abord qu'ici on a $M = 3$ *noeuds de quadrature et que le degré d'exactitude souhaité est* $M - 1$ *égale à 2. Cela suggère d'utiliser le théorème suivant :*

Theorem 1 (Poids de quadrature) *Soit J une formule de quadrature avec M noeuds.*

$$J \text{ est exacte de degré } M-1 \Longleftrightarrow \omega_j = \int_{-1}^{1} \varphi_j(t)dt,\ j=1,\ldots,M,$$

où $\{\varphi_j, j=1,\ldots,M\}$ est la base de Lagrange associée aux noeuds de quadrature.

Ici les poids de quadrature sont $\alpha_0,\alpha_1,\alpha_2$,faut donc les choisir tels que $\alpha_j = \int_{-1}^{1} \varphi_j(t)dt,\ j=0,\ldots,2$ (notez le décalage des indices).

$$\varphi_0 = \frac{\beta - x}{\beta+1}\cdot\frac{1-x}{1+1} = \frac{(\beta-x)(1-x)}{2(\beta+1)} = \frac{\beta-(\beta+1)x+x^2}{2(\beta+1)}$$

$$\int_{-1}^{1}\varphi_0 = \frac{1}{2(\beta+1)}\left[\beta x - \frac{1}{2}(\beta+1)x^2 + \frac{1}{3}x^3\right]\Big|_{-1}^{1} = \frac{1}{2(\beta+1)}[2\beta+\frac{2}{3}] = \frac{\beta+\frac{1}{3}}{\beta+1} = \alpha_0$$

$$\varphi_1 = \frac{-1-x}{-1-\beta}\cdot\frac{1-x}{1-\beta} = \frac{(1-x^2)}{(1-\beta^2)}$$

$$\int_{-1}^{1}\varphi_1 = \frac{1}{1-\beta^2}\left[x - \frac{1}{3}x^3\right]\Big|_{-1}^{1} = \frac{1}{(1-\beta^2)}[2-\frac{2}{3}] = \frac{4/3}{1-\beta^2} = \alpha_1$$

$$\varphi_2 = \frac{\beta - x}{\beta-1}\cdot\frac{-1-x}{-1-1} = \frac{(\beta-x)(1+x)}{2(\beta-1)} = \frac{\beta+(\beta-1)x-x^2}{2(\beta-1)}$$

$$\int_{-1}^{1}\varphi_0 = \frac{1}{2(\beta-1)}\left[\beta x + \frac{1}{2}(\beta-1)x^2 - \frac{1}{3}x^3\right]\Big|_{-1}^{1} = \frac{1}{2(\beta-1)}[2\beta-\frac{2}{3}] = \frac{\beta-\frac{1}{3}}{\beta-1} = \alpha_2$$

b) Il suffit que la formule intègre exactement le polynôme $p(x)=x^3$, c'est-à-dire

$$I_2(x^3) = -\alpha_0 + \beta^3\alpha_1 + \alpha_2 = \int_{-1}^{1} x^3 dx = 0\ .$$

On obtient l'équation

$$-(1+3\beta)(1-\beta) + 4\beta^3 + (1-3\beta)(1+\beta) = 0 \qquad \Longrightarrow 4\beta(\beta^2-1) = 0.$$

La seule solution dans l'intervalle $]-1,1[$ est $\beta=0$ et les poids deviennent dans ce cas

$$\alpha_0 = \alpha_2 = \frac{1}{3}, \qquad \alpha_1 = \frac{4}{3}\ .$$

c) Il suffit d'introduire la transformation $\tilde{x} = a + \dfrac{b-a}{2}(x+1)$. On a $d\tilde{x} = \frac{b-a}{2}dx$ et donc :

$$\int_a^b f(\tilde{x})d\tilde{x} = \frac{b-a}{2}\int_{-1}^{1} g(x)dx = \frac{b-a}{2}I_2(g(x))$$

où $g(x) = f(a+\frac{b-a}{2}(x+1))$. La formule devient donc :

$$I_2(f) = \frac{b-a}{2}\left[\alpha_0 f(a) + \alpha_1 f\left(\frac{b+a}{2}\right) + \alpha_2 f(b)\right]\ ,$$

soit :

$$I_2(f) = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{b+a}{2}\right) + f(b)\right]\ .$$

On reconnaît la formule de Simpson.

Exercice 3

Définition. Le degré d'exactitude d'une méthode d'intégration est donné par la valeur maximale de r pour laquelle la méthode intègre exactement tout polynôme de degré inférieur ou égal à r.

a) D'après la définition ci-dessus, déterminez le degré d'exactitude de la formule du rectangle (point milieu) simple, c-à-d sur un seul intervalle en prenant comme fonctions tests pour l'intégrale $I = \int_a^b f(x)$ les polynômes : $1, x, x^2, x^3, \ldots$.

b) On considère maintenant le calcul de l'intégrale

$$I = \int_0^1 x^{m-1}(1-x)^{n-1}\,dx,$$

où m et n sont deux valeurs entières $m, n \geq 1$. Écrivez la méthode du rectangle simple pour approcher I. Pour quelles valeurs de n et m cette formule donne la valeur exacte de I ?

Sol. :

a) On prend d'abord $f(x) = 1$ et on calcule l'intégrale exacte :

$$I = \int_a^b 1 = b - a.$$

Par la méthode du rectangle on trouve :

$$I_{pm} = (b-a) \cdot 1 = b - a.$$

Donc $I = I_{pm}$ pour $r = 0$. Considérons maintenant $f(x) = x$. L'intégrale exacte est :

$$I = \int_a^b x = \frac{b^2 - a^2}{2},$$

et l'intégrale approchée par la méthode du rectangle est :

$$I_{pm} = (b-a) \cdot \frac{a+b}{2}.$$

Donc $I = I_{pm}$ pour $r = 1$. Finalement, si on prend $f(x) = x^2$, on trouve

$$I = \int_a^b x^2 = \frac{b^3 - a^3}{3} = (b-a) \cdot \frac{b^2 + a^2 + ab}{3},$$

tandis que par la méthode du rectangle

$$I_{pm} = (b-a) \cdot \frac{(a+b)^2}{4}.$$

Vu que $I = I_{pm}$ pour $r = 0, 1$, mais $I \neq I_{pm}$ si $r = 2$, on peut conclure que la méthode du rectangle simple a degré d'exactitude 1, c-à-d qu'elle peut intégrer exactement les fonctions constantes et linéaires.

b) La méthode du rectangle simple s'écrit :

$$I_{pm} = \frac{1}{2}^{m-1} \cdot \frac{1}{2}^{n-1}.$$

Puisque la méthode a degré d'exactitude 1, on doit imposer

$$(m-1) + (n-1) \leq 1 \quad \Leftrightarrow \quad m + n - 2 \leq 1 \quad \Leftrightarrow \quad m + n \leq 3.$$

Donc, $I = I_{pm}$ si $m = 1$ et $n = 2$, ou $m = 2$ et $n = 1$.

Python

Exercice 4

On considère le calcul de l'intégrale

$$I = \int_0^1 f(x)\,dx,$$

où $f(x)$ est une fonction continue sur $[0, 1]$.

1. Ecrivez une fonction `midpoint` qui implémente la formule composite du rectangle (point milieu) pour l'approximation de l'intégrale ci-dessus. Pour permettre le choix d'un intervalle d'intégration de la forme $[a, b]$, le nombre de sous-intervalles N et la fonction $f(x)$, définie par la commande `f = lambda x : ...` utilisez la structure suivante :

```
def midpoint( a, b, M, f ) :
    # [a,b] : inteval
    # M : number of subintervals
    # f : fonction to integrate using the midpoint rule
```

2. Testez le code en considérant la fonction $f(x) = x^2$ et $M = 10$. Tracer ensuite le graphe de l'erreur $|I(f) - I^c_{pm}(f)|$ en fonction de M (utilisez à cette fin la commande `plt.xscale('log'); plt.yscale('log')`), en choisissant $a = 0$, $b = 1$, et $M = 10^1, 10^2, 10^3, \ldots 10^5$. Quel est l'ordre de convergence de la formule composite du rectangle par rapport à $H = (b-a)/M$? Donnez une interprétation des résultats d'après la théorie.

3. Modifiez le code du point a) pour permettre le calcul de l'intégrale avec la formule composite du trapèze. Tracer le graphe de l'erreur $|I(f) - I^c_t(f)|$ pour les mêmes valeurs de M. Comparez les résultats avec ceux obtenus au point b).

```
  def trapeziodal( a, b, N, f ) :
    # [a,b] : inteval
    # N : number of subintervals
    # f : fonction to integrate using the trapezoidal rule
```

Sol. : *Cf Jupyther notebooks*

Exercice 5

On considère la fonction $f : [a, b] \to \mathbb{R}$ dans $C^0([a, b])$; on est intéressé à approcher l'intégrale $I(f) = \int_a^b f(x)\,dx$.

1. Ecrivez une fonction `Simpson` qui implémente la formule composite de Simpson pour l'approximation de l'intégrale ci-dessus. Utilisez la structure suivante :

```
def Simpson( a, b, N, f ) :
    # [a,b] : inteval
    # N : number of subintervals
    # f : fonction to integrate using the Simpson rule
```

2. Testez ensuite pour quels monômes $f(x) = x^d$ cette formule intègre exactement la fonction, pour $d = 0, 1, ...$ sur l'intervalle $[1, 4]$, avec $N = 1$ et ensuite $N = 10$.
3. Vérifiez numériquement pour quelques polynômes que la fonction ainsi écrite est linéaire en f pour $N = 10$.

Suggestion : En utilisant une fonction 'lambda' il est possible de décider le paramètre d d'un monôme à un moment ultérieur :

```
monomial = lambda x :  x**d
```

Sol. : *Cf Jupyther notebooks*

Exercice 6

On considère la fonction $f : [a, b] \to \mathbb{R}$ dans $C^0([a, b])$; on est intéressé à approcher l'intégrale $I(f) = \int_a^b f(x)\, dx$.

Plus précisément on prend $f(x) = \sin(\frac{7}{2}\, x) + e^x - 1$ avec $a = 0$ et $b = 1$ ($f \in C^\infty([a, b])$) et on peut calculer $I(f) = \frac{2}{7}\,(1 - \cos(\frac{7}{2})) + e - 2$.

1. Calculez une approximation de l'intégrale en utilisant les formules du rectangle, du trapèze et de Simpson *simples*, c-à-d avec un seul intervalle.
2. Calculez une approximation de l'intégrale en utilisant les fonctions `midpoint`, `trapezoidal` et `simpson` (déjà codées) avec $N = 10$ sous-intervalles de même taille. On notera les valeurs approchées de l'intégrale $I^c_{mp}(f)$, $I^c_t(f)$, and $I^c_s(f)$, respectivement.
3. Répétez le point 2. avec $N = 2^k$ pour $k = 2, \ldots, 7$ et calculez les erreurs $E^c_{mp}(f) := |I(f) - I^c_{mp}(f)|$, $E^c_t(f) := |I(f) - I^c_t(f)|$, et $E^c_s(f) := |I(f) - I^c_s(f)|$. Dessinez les erreurs en fonction de $H = (b-a)/N$ sur une échelle logarithmique sur les deux axes. Quel est l'ordre de convergence de ces méthodes ? Est-ce en accord avec la théorie ? Motivez votre réponse.
4. On prend maintenant $f(x) = x^d$, $a = 0$ et $b = 1$, avec $d \in \mathbb{N}$. L'intégrale de f vaut $I(f) = 1/(d + 1)$. Vérifiez numériquement les degrés d'exactitude de chacune des formules de quadrature du point 1. Pour cela, il faut choisir plusieurs valeurs de $d = 0, 1, 2, \ldots$. Motivez votre réponse.

Sol. : *Cf Jupyther notebooks*

Exercice 7

On considère, sur l'intervalle $[-1, 1]$, la fonction f suivante :

$$f(x) = \begin{cases} e^x & \text{si } x \leq 0 \\ 1 & \text{si } x > 0 \end{cases}$$

On peut définir une telle fonction en utilisant la commande

```
f = lambda x :  np.exp(x)*(x<=0) + (1)*(x>0)
```

1. Utilisez 1000 points équirépartis dans l'intervalle $[-1, 1]$ pour afficher la fonction f (utilisez la commande `axis` pour recadrer l'image).
2. On s'intéresse à présent à l'intégrale $I = \int_{-1}^{1} f(x)\, dx$. On peut calculer la valeur de I analytiquement et on trouve $I = 2 - \frac{1}{e} \cong 1.6321$. Calculez des valeurs approchées de I en considérant les formules du point milieu, du trapèze et de Simpson avec $N = 1, 9, 99, 999$ où N est le nombre de sous-intervalles de formules composites. Utilisez les fonctions `midpoint`, `trapezoidal` et `simpson` (déjà codées) .
3. Reportez les erreurs calculées au point (b) dans un graphe montrant l'erreur en fonction de H avec des échelles logarithmiques (commande `loglog`).
4. Estimez, à partir des graphes obtenus au point précédent l'ordre de chacune des méthodes. Comparez-les avec les ordres donnés au cours. Y a-t-il des différences ? Pourquoi ? (Regardez les dérivées de f).
5. Pourquoi obtient-on de bien meilleurs résultats pour la méthode de Simpson avec un nombre pair d'intervalles qu'avec un nombre impair (essayez avec $N = 99$ et $N = 100$) ?
6. Refaites l'exercice avec $N = 2, 10, 100, 1000$.

Sol. : *Cf Jupyther notebooks*

Sol. :

Analyse Numérique
Série 06

Simone Deparis

March 28, 2025

Contents

1 Exercice Série 6, Ex 4

1.1 Formule du rectangle et du trapèze

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

On considère le calcul de l'intégrale

$$I = \int_0^1 f(x)\,dx,$$

où $f(x)$ est une fonction continue sur $[0, 1]$.

1. Ecrivez une fonction `midpoint` qui implémente la formule composite du rectangle (point milieu) pour l'approximation de l'intégrale ci-dessus. Pour permettre le choix d'un intervalle d'intégration de la forme $[a, b]$, le nombre de sous-intervalles N et la fonction $f(x)$, définie par la commande `f = lambda x : ...` utilisez la structure suivante:

```
def midpoint( a, b, N, f ) :
    # [a,b] : inteval
    # N : number of subintervals
    # f : fonction to integrate using the midpoint rule
```

2. Testez le code en considérant la fonction $f(x) = x^2$ et $M = 10$. Tracez ensuite le graphe de l'erreur $|I(f) - I_{pm}^c(f)|$ en fonction de N (utilisez à cette fin les comman-

des `plt.xscale('log'); plt.yscale('log')`), en choisissant $a = 0$, $b = 1$, et $M = 10^1, 10^2, 10^3, \ldots 10^5$. Quel est l'ordre de convergence de la formule composite du rectangle par rapport à $H = (b-a)/M$? Donnez une interprétation des résultats d'après la théorie.

3. Modifiez le code du point 1. pour permettre le calcul de l'intégrale avec la formule composite du trapèze. Tracez le graphe de l'erreur $|I(f) - I_t^c(f)|$ pour les mêmes valeurs de M. Comparez les résultats avec ceux obtenus au point 2.

```
def trapeziodal( a, b, N, f ) :
    # [a,b] : inteval
    # N : number of subintervals
    # f : fonction to integrate using the trapezoidal rule
```

Partie 1

[2]:
```
def midpoint( a, b, N, f ) :
    # [a,b] : inteval
    # M : number of subintervals
    # f : fonction to integrate using the midpoint rule

    # size of the subintervals
    H = (b - a) / N

    # quadrature nodes
    x = np.linspace(a+H/2, b-H/2, N)

    # approximate integral
    return H * np.sum( f(x) );
```

Partie 2

[3]:
```
f = lambda x :  x**2

a = 0; b = 1; N = 10;

intmp = midpoint( a, b, N, f )

print(intmp)
```

```
0.3325
```

En sachant que la valeur exacte de l'intégrale est $I(f) = 1/3$, on peut écrire les commandes suivantes:

[4]:
```
N = 10**np.linspace(1,5,5).astype(int)

errmp = []
for i in range(5) :
    intmp = midpoint( a, b, N[i], f)
    errmp.append( np.abs( intmp - 1.0/3 ) )

plt.plot(N, errmp, 'b:.')
```

```
plt.xlabel('M'); plt.ylabel('err');

plt.xscale('log')
plt.yscale('log')
plt.grid(True)
plt.title('Le graphe loglog est une droite avec pente égale à -2:')
plt.show()
print('Figure: Erreur relative à la méthode du rectangle en échelle␣
 ↪logarithmique.\n\n')

slope = ( np.log(errmp[4]) - np.log(errmp[0]) ) / ( np.log(N[4]) - np.log(N[0]) )

print(f'La pente est de {slope:.6f}')
```

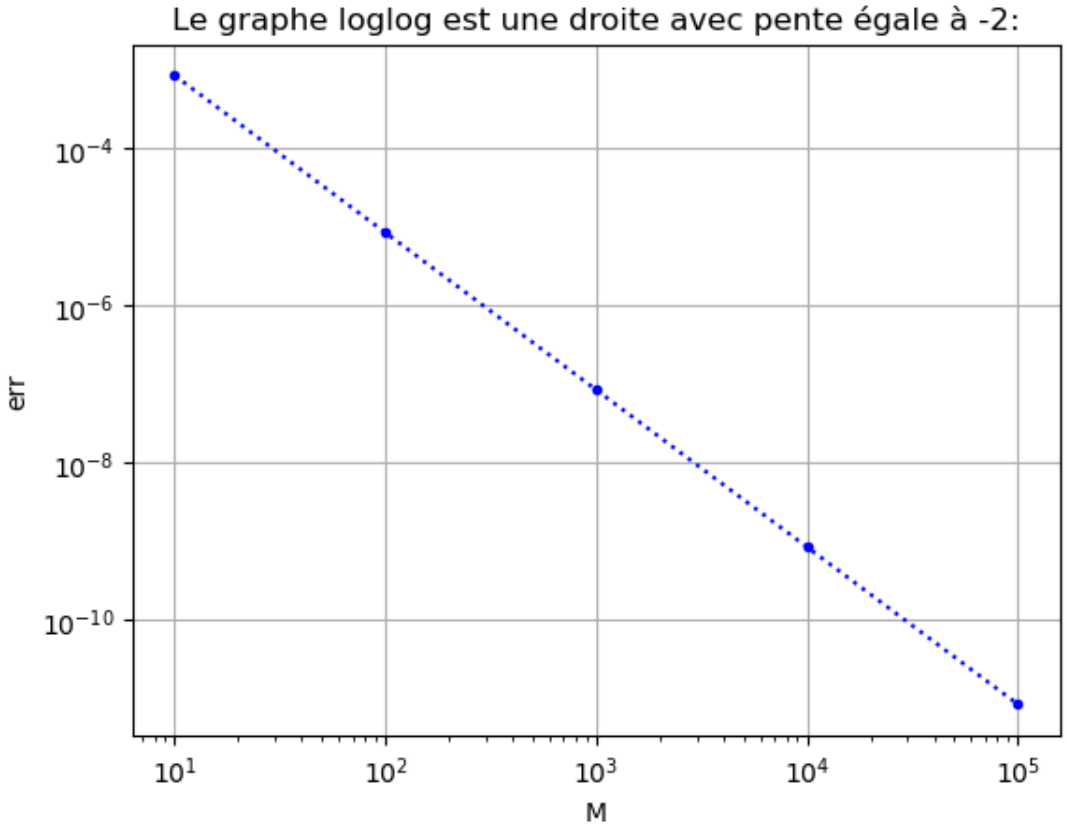


Figure: Erreur relative à la méthode du rectangle en échelle logarithmique.

La pente est de -2.000001

Donc, l'erreur décroît comme la puissance -2 du paramètre N. Par conséquent, l'ordre de convergence par rapport à $H = 1/N$ est 2.

Partie 3 On peut modifier le code du point milieu pour implémenter la méthode du trapèze comme suit.

[5]:
```
def trapezoidal( a, b, N, f ) :
    # [a,b] : inteval
    # M : number of subintervals
    # f : fonction to integrate using the trapezoidal rule

    # size of the subintervals
    H = (b - a) / N

    # quadrature nodes
    x = np.linspace(a, b, N+1)

    # approximate integral
    return H/2 * ( f(a) + f(b) ) + H * sum( f(x[1:N]) );
```

[6]:
```
errtrap = []
for i in range(5) :
    inttrap = trapezoidal( a, b, N[i], f)
    errtrap.append( np.abs( inttrap - 1.0/3 ) )

plt.plot(N, errmp, 'b:.')
plt.plot(N, errtrap, 'c:*')


plt.legend(['rectangle', 'trapeze'])
plt.xlabel('N'); plt.ylabel('err');

plt.xscale('log')
plt.yscale('log')
plt.grid(True)
plt.title('Les graphes loglog sont des droites avec pente égale à -2:')
plt.show()
print('Figure: Erreur relative aux méthodes du rectangle et du trapèze en␣
 ↪échelle logarithmique.\n\n')

slope = ( np.log(errtrap[4]) - np.log(errtrap[0]) ) / ( np.log(N[4]) - np.
 ↪log(N[0]) )

print(f'La convergence numérique est d\'environ {slope:.2f}')

print(f'En moyenne, le rapport entre les deux erreurs est de {np.mean(np.abs(np.
 ↪array(errtrap)/np.array(errmp))):.6f}')
```

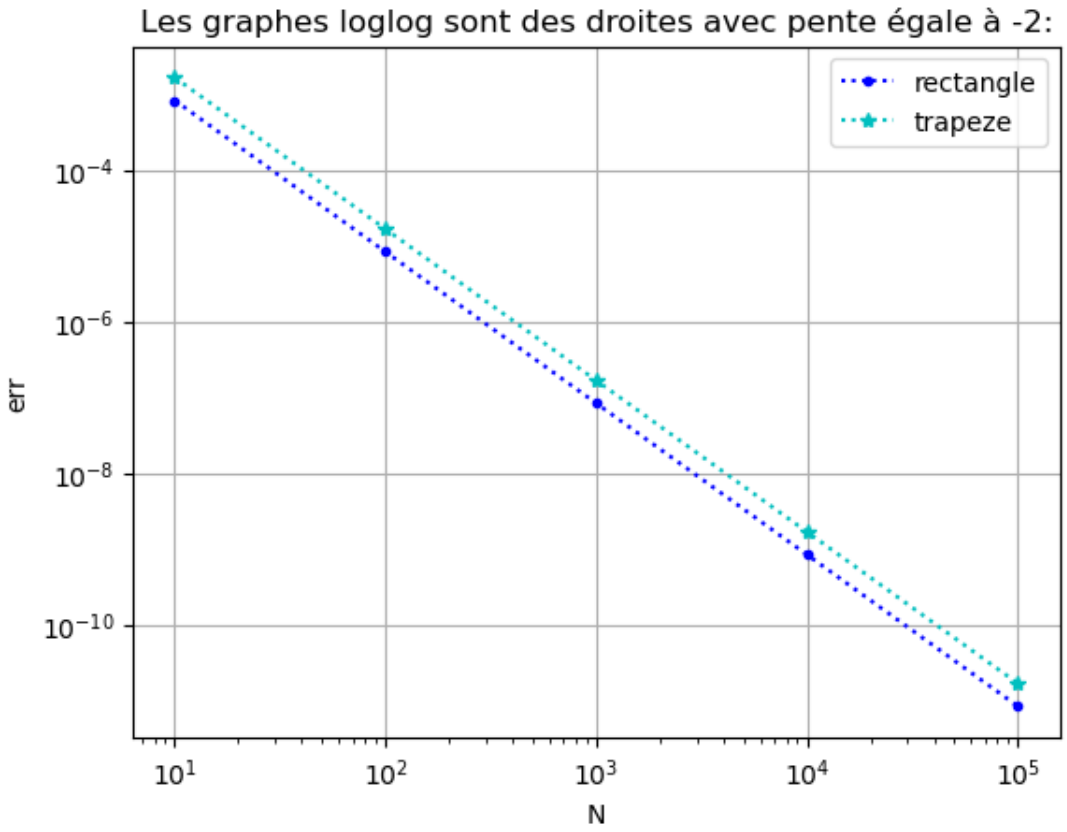


Figure: Erreur relative aux méthodes du rectangle et du trapèze en échelle logarithmique.

```
La convergence numérique est d'environ -2.00
En moyenne, le rapport entre les deux erreurs est de 1.999993
```

On observe que la précision est encore $\mathcal{O}(h^2)$, mais que l'erreur est plus grande (le double, précisément) que dans le cas du point milieu.

[ ]:

2 Exercice Série 6, Ex 5

2.1 Formule de Simpson

[7]:
```
import matplotlib.pyplot as plt
import numpy as np
```

On considère la fonction $f : [a, b] \to \mathbb{R}$ dans $C^0([a, b])$; on est intéressé à approcher l'intégrale $I(f) = \int_a^b f(x)\,dx$.

1. Ecrivez une fonction `Simpson` qui implémente la formule composite de Simpson pour l'approximation de l'intégrale ci-dessus. Utilisez la structure suivante:

```
def Simpson( a, b, N, f ) :
    # [a,b] : inteval
    # N : number of subintervals
    # f : fonction to integrate using the Simpson rule
```

2. Testez ensuite pour quels monômes $f(x) = x^d$ cette formule intègre exactement la fonction, pour $d = 0, 1, ...$ sur l'intervalle $[1, 4]$, avec $N = 1$ et ensuite $N = 10$.

3. Vérifiez numériquement pour quelques polynômes que la fonction ainsi écrite est linéaire en f pour $N = 10$

Suggestion: En utilisant une fonction `lambda` il est possible de décider le paramètre d d'un monôme à un moment ultérieur:

```
monomial = lambda x :  x**d
```

Partie 1

[8]:
```
# This function just provides the Simpson quadrature rule.
def Simpson( a, b, N, f ) :
    # [a,b] : inteval
    # N : number of subintervals
    # f : fonction to integrate using the trapezoidal rule

    M=3

    nodes = np.array([-1, 0, 1])
    weights = np.array([1./3., 4./3., 1./3.])

    # size of the subintervals
    H = (b - a) / N
    # points defining intervals
    x = np.linspace(a, b, N+1)

    Lh = 0;

    z = np.zeros(M);

    for k in range(N) :
        # left of the subinterval, also first quadrature point
        z[0] = x[k]
        # right of the subinterval, also third quadrature point
        z[2] = x[k+1]
        # mid point, , also second quadrature point
        z[1] = (x[k] + x[k+1])/2
        # can also be computed as
        z[1] = (x[k] + x[k+1])/2 + nodes[1]*(x[k+1] - x[k])/2
```

```
        # local quadrature:
        Jgk = weights[0] * f(z[0]) + weights[1] * f(z[1]) + weights[2] * f(z[2])
        # or as a single sum
        Jgk = sum(weights * f(z))

        Lh = Lh + Jgk

    # approximate integral
    return H/2 * Lh
```

Partie 2

$$\int_1^4 1dx = 3$$

$$\int_1^4 x^d dx = \frac{1}{d+1} x^{d+1}|_1^4 = \frac{4^{d+1}-1}{d+1}$$

[9]:
```
# Checking simpson fonction
a = 1; b = 4;

# with lambda fonctions, it is possible to determine a parameter (here d)
# at a later moment
monomial = lambda x :  x**d

# recording for which degrees the integral s exact (up to epsilon)
exactDegree = -1
epsilon = 1e-12
```

[10]:
```
N = 1
for d in range(7) :
    intsim = Simpson( a, b, N, monomial )
    intExact = (4**(d+1) - 1)/(d+1)
    print(f'Simpson on x^{d} : {intsim:.6f} - {intExact:.6f}  = {intsim-intExact:
 ↪.6e}')
    if np.abs(intsim-intExact) < epsilon :
        exactDegree = d

print(f'Simpson with N = {N} is exact up to degree {exactDegree}')
```

```
Simpson on x^0 : 3.000000 - 3.000000  = -4.440892e-16
Simpson on x^1 : 7.500000 - 7.500000  = 0.000000e+00
Simpson on x^2 : 21.000000 - 21.000000  = 0.000000e+00
Simpson on x^3 : 63.750000 - 63.750000  = 0.000000e+00
Simpson on x^4 : 206.625000 - 204.600000  = 2.025000e+00
Simpson on x^5 : 707.812500 - 682.500000  = 2.531250e+01
Simpson on x^6 : 2536.781250 - 2340.428571  = 1.963527e+02
Simpson with N = 1 is exact up to degree 3
```

[11]:
```
N = 10
exactDegree = -1

for d in range(7) :
    intsim = Simpson( a, b, N, monomial )
    intExact = (4**(d+1) - 1)/(d+1)
    print(f'Simpson on x^{d} : {intsim:.6f} - {intExact:.6f}  = {intsim-intExact:
 ↪.6e}')
    if np.abs(intsim-intExact) < epsilon :
        exactDegree = d

print(f'Simpson with N = {N} is exact up to degree {exactDegree}')
```

```
Simpson on x^0 : 3.000000 - 3.000000  = -4.440892e-16
Simpson on x^1 : 7.500000 - 7.500000  = 0.000000e+00
Simpson on x^2 : 21.000000 - 21.000000  = 0.000000e+00
Simpson on x^3 : 63.750000 - 63.750000  = -1.421085e-14
Simpson on x^4 : 204.600202 - 204.600000  = 2.025000e-04
Simpson on x^5 : 682.502531 - 682.500000  = 2.531250e-03
Simpson on x^6 : 2340.449818 - 2340.428571  = 2.124623e-02
Simpson with N = 10 is exact up to degree 3
```

Partie 3

$$\int_1^4 1dx = 3$$

$$\int_1^4 x^d dx = \frac{1}{d+1} x^{d+1}|_1^4 = \frac{4^{d+1}-1}{d+1}$$

[12]:
```
maxD = 5;
N = 10

p = lambda x :  np.polyval(coefs,x)

# pre-computing integrls of monomials up to degree maxD
intMono = np.zeros(maxD+1)
for d in range(maxD+1) :
    intMono[d] = Simpson( a, b, N, monomial )
```

[13]:
```
# generating random coefficients
coefs = np.random.rand(maxD+1)

# evaluating Simpson on the polynomial :
intPoly = Simpson(a,b,N, p)

# computing integral by linearity. Remeber that in polyval, the order of the␣
 ↪coefficients is opposite !
intSum = 0
```

```
for d in range(maxD+1) :
    intSum = intSum + coefs[maxD-d]*intMono[d]

print(f'Simpson on a_{d} + a_{d-1} x + ... + a_0x^{d} by linearity : \n'
      f'\t {intPoly:.6f} - {intSum:.6f}  = {intPoly-intSum:.6e}')
```

```
Simpson on a_5 + a_4 x + ... + a_0x^5 by linearity :
        416.109118 - 416.109118  = 0.000000e+00
```

[ ]:

3 Exercice Série 6, Ex 6

3.1 Degré d'exactitude d'une formule de quadrature

On considère la fonction $f : [a,b] \to \mathbb{R}$ dans $C^0([a,b])$; on est intéressé à approcher l'intégrale $I(f) = \int_a^b f(x)\,dx$.

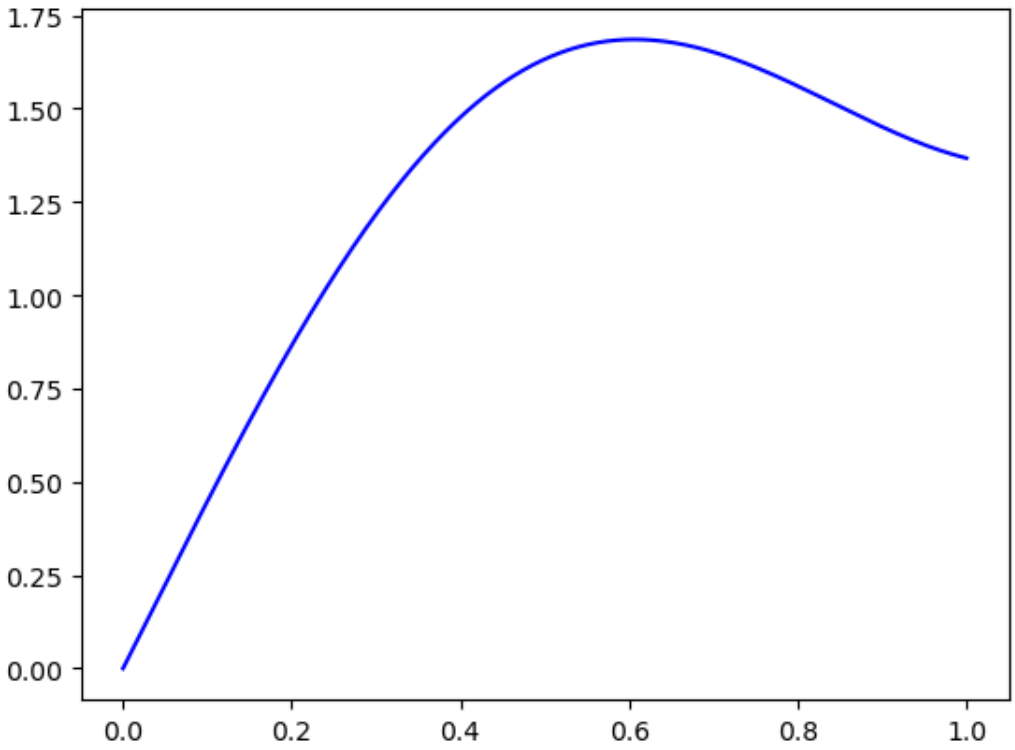
Plus précisément on prend $f(x) = \sin(\frac{7}{2}x) + e^x - 1$ avec $a = 0$ et $b = 1$ ($f \in C^\infty([a,b])$) et on peut calculer $I(f) = \frac{2}{7}(1 - \cos(\frac{7}{2})) + e - 2$.

1. Calculez une approximation de l'intégrale en utilisant les formules du rectangle, du trapèze et de Simpson *simples*, c-à-d avec un seul intervalle.
2. Calculez une approximation de l'intégrale en utilisant les fonctions `Midpoint`, `Trapezoidal` et `Simpson` (déjà codées). avec $N = 10$ sous-intervalles de même taille. On notera les valeurs approchées de l'intégrale $I^c_{mp}(f)$, $I^c_t(f)$, and $I^c_s(f)$, respectivement.
3. Répétez le point 2. avec $N = 2^k$ pour $k = 2, \ldots, 7$ et calculez les erreurs $E^c_{mp}(f) := |I(f) - I^c_{mp}(f)|$, $E^c_t(f) := |I(f) - I^c_t(f)|$, et $E^c_s(f) := |I(f) - I^c_s(f)|$. Dessinez les erreurs en fonction de $H = (b-a)/N$ sur une échelle logarithmique sur les deux axes. Quel est l'ordre de convergence de ces méthodes ? Est-ce en accord avec la théorie ? Motivez votre réponse.
4. On prend maintenant $f(x) = x^d$, $a = 0$ et $b = 1$, avec $d \in \mathbb{N}$. L'intégrale de f vaut $I(f) = 1/(d+1)$. Vérifiez numériquement les degrés d'exactitude de chacune des formules de quadrature du point 1. Pour cela, il faut choisir plusieurs valeurs de $d = 0, 1, 2, \ldots$. Motivez votre réponse.

[14]:
```
import matplotlib.pyplot as plt
import numpy as np

# In my case, the IntegrationLib is in the parent directory,
# therfore have have to add the aprent directory to path :
import sys
sys.path.append('..')

from IntegrationLib import *
```

Partie 1

[15]:
```
f = lambda x :  np.sin(7/2*x) + np.exp(x) - 1

a = 0; b = 1
Iexact = 2/7*(1-np.cos(7/2) ) + np.exp(1) - 2

x = np.linspace(a,b,1000)
y=f(x)

plt.plot(x, y, 'b')
plt.show()
```



[16]:
```
intMD   = (b-a) * f( (a+b)/2 )
intTrap = (b-a) * ( f(a)+f(b) )/2
intSimp = (b-a)/6 * ( f(a) + 4*f((a+b)/2) + f(b) )

print(f'exact \t rect \t\t trap \t\t Simpson')
print(f'{Iexact:.4f} \t {intMD:.4f} \t {intTrap:.4f} \t {intSimp:.4f}')
```

```
exact    rect            trap            Simpson
1.2716   1.6327          0.6837          1.3164
```

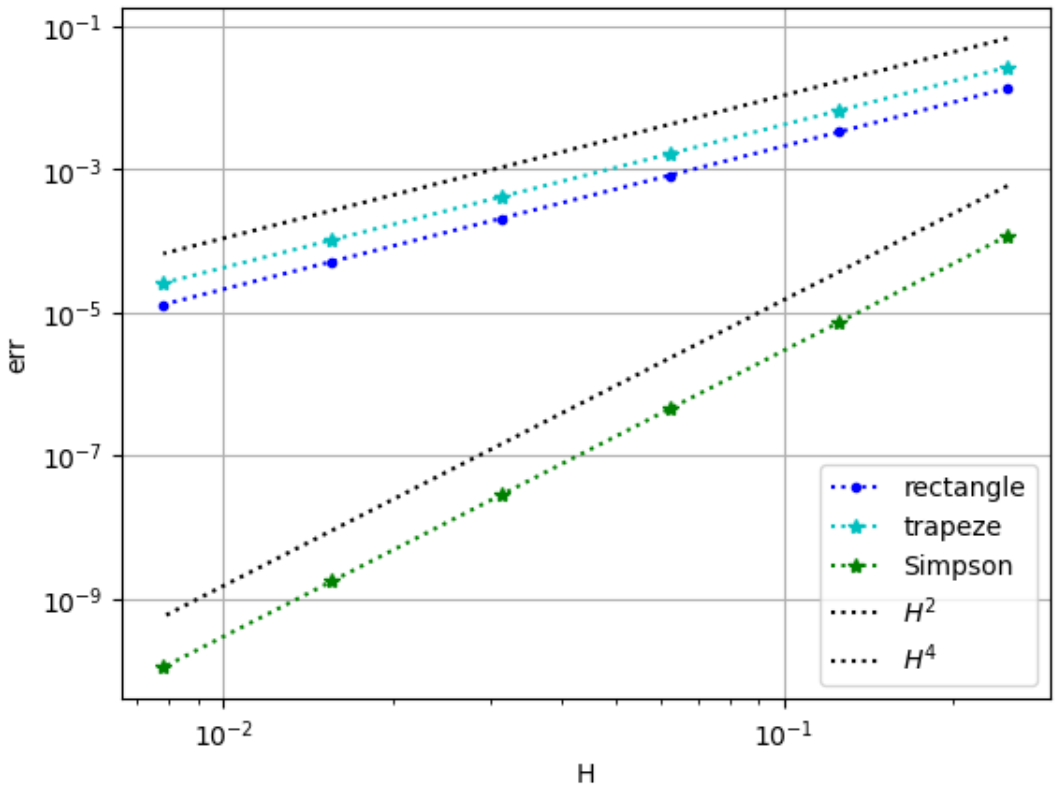
Partie 2

[17]:
```
N = 10
intMD   = Midpoint(a,b,N,f)
intTrap = Trapezoidal(a,b,N,f)
intSimp = Simpson(a,b,N,f)

print(f'exact \t rect \t\t trap \t\t Simpson')
print(f'{Iexact:.4f} \t {intMD:.4f} \t {intTrap:.4f} \t {intSimp:.4f}')
```

```
exact    rect            trap            Simpson
1.2716   1.2737          1.2673          1.2716
```

Partie 3

[18]:
```
errmp = []
errtrap = []
errSim = []

N = 2**np.linspace(2,7,6).astype(int)

for n in N :
    errmp.append( np.abs( Midpoint( a, b, n, f) - Iexact ) )
    errtrap.append( np.abs( Trapezoidal( a, b, n, f) - Iexact ) )
    errSim.append( np.abs( Simpson( a, b, n, f) - Iexact ) )

H = (b-a)/N
plt.plot(H, errmp, 'b:.',  H, errtrap, 'c:*', H, errSim, 'g:*')

plt.plot(H, H**2 * (errmp[0]/H[0]**2)*5, 'k:', H, H**4 * (errSim[0]/H[0]**4)*5,␣
 ↪'k:')

plt.legend(['rectangle', 'trapeze', 'Simpson', '$H^2$', '$H^4$'])
plt.xlabel('H'); plt.ylabel('err');

plt.xscale('log')
plt.yscale('log')
plt.grid(True)
```



[Pardon d'écrire en anglais..]

We recall that the errors $E^c_{mp}(f) = |e^c_{mp}(f)|$, $E^c_t(f) = |e^c_t(f)|$, and $E^c_s(f) = |e^c_s(f)|$ read, for a sufficiently regular function $f(x)$:

$$e^c_{mp}(f) = I(f) - I^c_{mp}(f) = \frac{b-a}{24} H^2 f''(\xi), \qquad \text{for some } \xi \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e^c_t(f) = I(f) - I^c_t(f) = -\frac{b-a}{12} H^2 f''(\eta), \qquad \text{for some } \eta \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e^c_s(f) = I(f) - I^c_s(f) = -\frac{b-a}{16 \cdot 180} H^4 f^{(4)}(\zeta), \qquad \text{for some } \zeta \in [a,b], \qquad \text{if } f \in C^4([a,b]).$$

Since in this case $f(x) \in C^\infty([a,b])$, we expect the orders of accuracy (convergence orders of the errors) to be equal to 2, 2, and 4 for the composite midpoint, trapezoidal, and Simpson quadrature formulas, respectively. This is confirmed by the figure above, where we can observe that the plots of the errors $E_{mp}(f)$ and $E_t(f)$ vs. H are, in log-log scale, parallel to the line representing the curve (H, H^2), thus indicating the order of accuracy (convergence order) 2 for the composite midpoint and trapezoidal quadrature formulas. Similarly, the plot of the error $E_s(f)$ is, in log-log scale, parallel to the line representing the curve (H, H^4), from which we deduce the order of accuracy (convergence order) 4 for the composite Simpson quadrature formula.

We notice that the orders of accuracy could be deduced by computing the errors for two different values of H, say H_1 and H_2; for example for a generic composite quadrature formula

we obtain the corresponding errors $E^c_{H_1}(f)$ and $E^c_{H_2}(f)$. If we assume that the error $E^c_H(f)$ can be expressed as $E^c_H(f) = C\,H^\alpha$, with $\alpha > 0$ and C a positive constant independent of H and α, we can estimate the order of accuracy (convergence order) of the quadrature formula as $\alpha = \log_\beta\left(E^c_{H_2}(f)/E^c_{H_1}(f)\right)/\log_\beta\left(H_2/H_1\right)$, for any $\beta > 1$ and H_1 and H_2 "sufficiently small''. For the composite midpoint, trapezoidal, and Simpson quadrature formulas, we use the following commands, for which the results ($\alpha = 2.01$, 2.01, and 4.01, respectively) confirm the expected orders of accuracy (convergence orders).

[19]:
```
slopeMP = ( np.log(errmp[-1]) - np.log(errmp[0]) ) / ( np.log(N[-1]) - np.
 ↪log(H[0]) )
slopeTrap = ( np.log(errtrap[-1]) - np.log(errtrap[0]) ) / ( np.log(N[-1]) - np.
 ↪log(H[0]) )
slopeSim = ( np.log(errSim[-1]) - np.log(errSim[0]) ) / ( np.log(N[-1]) - np.
 ↪log(H[0]) )

print(f'La convergence numérique est environ de ')
print(f'Rectangle : {slopeMP:.2f}')
print(f'Trapeze : {slopeTrap:.2f}')
print(f'Simpson : {slopeSim:.2f}')
```

```
La convergence numérique est environ de
Rectangle : -1.12
Trapeze : -1.11
Simpson : -2.23
```

Partie 4 The function $f(x) = x^d$ is a polynomial of degree d for $d \in \mathbb{N}$. The simple midpoint, trapezoidal, and Simpson quadrature formulas possesses degree of exactness equal to 1, 1, and 3, respectively.

[20]:
```
N = 1
# with lambda fonctions, it is possible to determine a parameter (here d)
# at a later moment
monomial = lambda x :  x**d

# recording for which degrees the integral s exact (up to epsilon)
exactDegree = -1
epsilon = 1e-12

for d in range(7) :
    intsim = Midpoint( a, b, N, monomial )
    intExact = 1/(d+1)
    print(f'Midpoint on x^{d} : {intsim:.6f} - {intExact:.6f}  =␣
 ↪{intsim-intExact:.6e}')
    if np.abs(intsim-intExact) < epsilon :
        exactDegree = d

print(f'Midpoint is exact up to degree {exactDegree}')
```

```
Midpoint on x^0 : 1.000000 - 1.000000  = 0.000000e+00
Midpoint on x^1 : 0.500000 - 0.500000  = 0.000000e+00
Midpoint on x^2 : 0.250000 - 0.333333  = -8.333333e-02
Midpoint on x^3 : 0.125000 - 0.250000  = -1.250000e-01
Midpoint on x^4 : 0.062500 - 0.200000  = -1.375000e-01
Midpoint on x^5 : 0.031250 - 0.166667  = -1.354167e-01
Midpoint on x^6 : 0.015625 - 0.142857  = -1.272321e-01
Midpoint is exact up to degree 1
```

[21]:
```
for d in range(7) :
    intsim = Trapezoidal( a, b, N, monomial )
    intExact = 1/(d+1)
    print(f'Trapezoidal on x^{d} : {intsim:.6f} - {intExact:.6f}  =␣
 ↪{intsim-intExact:.6e}')
    if np.abs(intsim-intExact) < epsilon :
        exactDegree = d

print(f'Trapezoidal is exact up to degree {exactDegree}')
```

```
Trapezoidal on x^0 : 1.000000 - 1.000000  = 0.000000e+00
Trapezoidal on x^1 : 0.500000 - 0.500000  = 0.000000e+00
Trapezoidal on x^2 : 0.500000 - 0.333333  = 1.666667e-01
Trapezoidal on x^3 : 0.500000 - 0.250000  = 2.500000e-01
Trapezoidal on x^4 : 0.500000 - 0.200000  = 3.000000e-01
Trapezoidal on x^5 : 0.500000 - 0.166667  = 3.333333e-01
Trapezoidal on x^6 : 0.500000 - 0.142857  = 3.571429e-01
Trapezoidal is exact up to degree 1
```

[22]:
```
for d in range(7) :
    intsim = Simpson( a, b, N, monomial )
    intExact = 1/(d+1)
    print(f'Simpson on x^{d} : {intsim:.6f} - {intExact:.6f}  = {intsim-intExact:
 ↪.6e}')
    if np.abs(intsim-intExact) < epsilon :
        exactDegree = d

print(f'Simpson is exact up to degree {exactDegree}')
```

```
Simpson on x^0 : 1.000000 - 1.000000  = 0.000000e+00
Simpson on x^1 : 0.500000 - 0.500000  = 0.000000e+00
Simpson on x^2 : 0.333333 - 0.333333  = 0.000000e+00
Simpson on x^3 : 0.250000 - 0.250000  = 0.000000e+00
Simpson on x^4 : 0.208333 - 0.200000  = 8.333333e-03
Simpson on x^5 : 0.187500 - 0.166667  = 2.083333e-02
Simpson on x^6 : 0.177083 - 0.142857  = 3.422619e-02
Simpson is exact up to degree 3
```

De ces simulations nous vérifions que les monômes $f(x) = x^d$ sont intégrées exactement pour les degrées $d = 0$ et $d = 1$ dans le cas des formules du rectangle et du trapèze, et pour $d = 0, 1, 2, 3$

dans le cas de la formule de Simpson.

[ ]:

4 Exercice Série 6, Ex 7

4.1 Convergence pour fonction non-lisse

[23]:
```
import matplotlib.pyplot as plt
import numpy as np

# In my case, the IntegrationLib is in the parent directory,
# therfore have have to add the aprent directory to path :
import sys
sys.path.append('..')

from IntegrationLib import Midpoint
from IntegrationLib import Trapezoidal
from IntegrationLib import Simpson
```

On considère, sur l'intervalle $[-1, 1]$, la fonction f suivante:

$$f(x) = \begin{cases} e^x & \text{si } x \leq 0 \\ 1 & \text{si } x > 0 \end{cases}$$

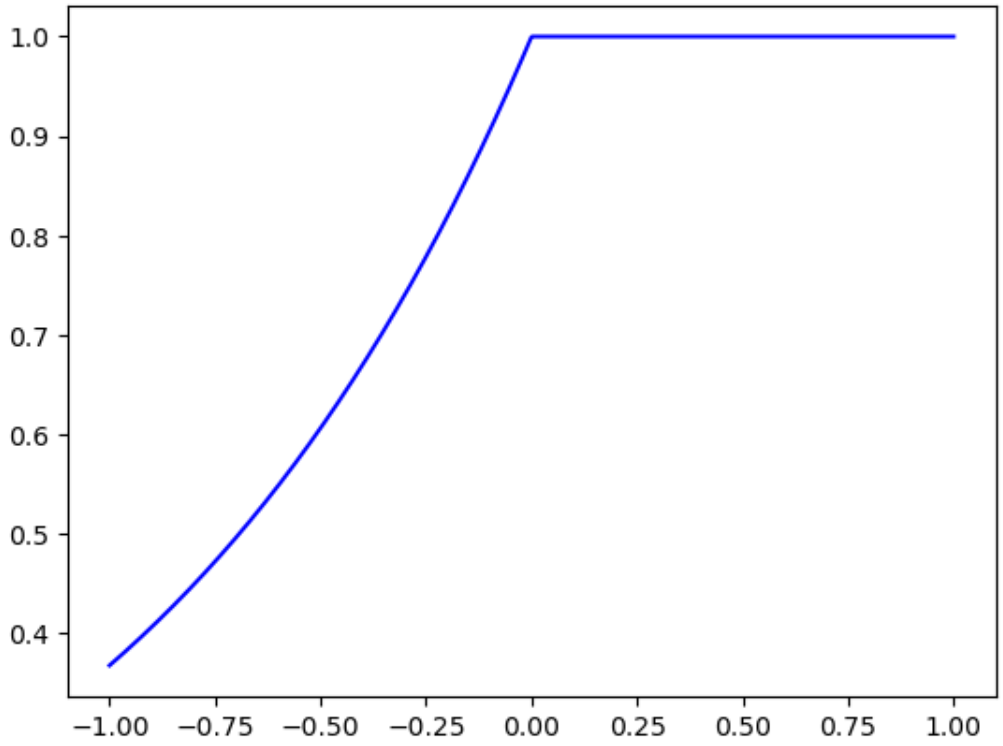
On peut définir une telle fonction en utilisant la commande

```
f = lambda x :  np.exp(x)*(x<=0) + (1)*(x>0)
```

1. Utilisez 1000 points équirépartis dans l'intervalle $[-1, 1]$ pour afficher la fonction f (utilisez la commande `axis` pour recadrer l'image).
2. On s'intéresse à présent à l'intégrale $I = \int_{-1}^{1} f(x)\ dx$. On peut calculer la valeur de I analytiquement et on trouve $I = 2 - \frac{1}{e} \cong 1.6321$. Calculez des valeurs approchées de I en considérant les formules du point milieu, du trapèze et de Simpson avec $N = 1, 9, 99, 999$ où N est le nombre de sous-intervalles de formules composites. Utilisez les fonctions `midpoint`, `trapezoidal` et `simpson` (déjà codées).
3. Reportez les erreurs calculées au point (b) dans un graphe montrant l'erreur en fonction de H avec des échelles logarithmiques.
4. Estimez, à partir des graphes obtenus au point précédent, l'ordre de chacune des méthodes. Comparez-les avec les ordres donnés au cours. Y a-t-il des différences? Pourquoi? (Regardez les dérivées de f).
5. Pourquoi obtient-on de bien meilleurs résultats pour la méthode de Simpson avec un nombre pair d'intervalles qu'avec un nombre impair (essayez avec $N = 99$ et ensuite $N = 100$)?
6. Refaites l'exercice avec $N = 2, 10, 100, 1000$.

Partie 1

[24]:
```
f = lambda x :  np.exp(x)*(x<=0) + (1)*(x>0)

a = -1; b = 1
x = np.linspace(a,b,1000)
y=f(x)

plt.plot(x, y, 'b')
plt.show()
```



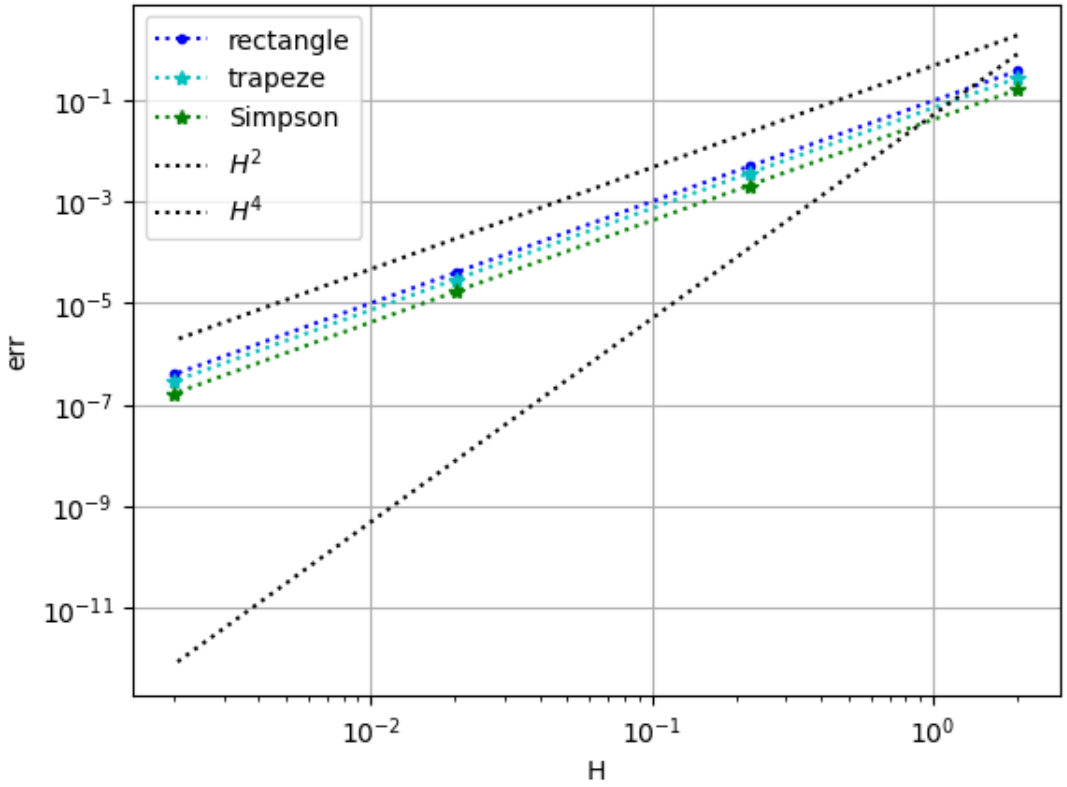
Partie 2 On calcule tout d'abord la valeur exacte de l'intégrale, puis on calcule les erreurs, que l'on stocke dans des vecteurs:

[25]:
```
Iexact = 2 - np.exp(-1)

errmp = []
errtrap = []
errSim = []

N = [1,9,99,999]
#N = [2,10,100,1000]

for i in range(4) :
```

```
    errmp.append( np.abs( Midpoint( a, b, N[i], f) - Iexact ) )
    errtrap.append( np.abs( Trapezoidal( a, b, N[i], f) - Iexact ) )
    errSim.append( np.abs( Simpson( a, b, N[i], f) - Iexact ) )
```

Partie 3

[26]:
```
H = 2./np.array(N)
plt.plot(H, errmp, 'b:.',  H, errtrap, 'c:*', H, errSim, 'g:*')

plt.plot(H, H**2 * (errmp[0]/H[0]**2)*5, 'k:', H, H**4 * (errSim[0]/H[0]**4)*5,␣
↪'k:')

plt.legend(['rectangle', 'trapeze', 'Simpson', '$H^2$', '$H^4$'])
plt.xlabel('H'); plt.ylabel('err');

plt.xscale('log')
plt.yscale('log')
plt.grid(True)
plt.show()
```



Les graphes pour toutes les méthodes sont des droites. On remarque que lorsque H est divisé par

10, les erreurs sont environ divisées par 100. On peut donc supposer que les ordres sont 2 par rapport à H. Pour le confirmer, on peut ajouter sur le graphe la pente représentant l'ordre 2 et vérifier qu'elle est parallèle aux droites d'erreur.

Partie 4

[27]:
```
slopeMP = ( np.log(errmp[-1]) - np.log(errmp[0]) ) / ( np.log(H[-1]) - np.
 ↪log(H[0]) )
slopeTrap = ( np.log(errtrap[-1]) - np.log(errtrap[0]) ) / ( np.log(H[-1]) - np.
 ↪log(H[0]) )
slopeSim = ( np.log(errSim[-1]) - np.log(errSim[0]) ) / ( np.log(H[-1]) - np.
 ↪log(H[0]) )

print(f'La convergence numérique est environ de ')
print(f'Rectangle : {slopeMP:.2f}')
print(f'Trapeze : {slopeTrap:.2f}')
print(f'Simpson : {slopeSim:.2f}')
```

```
La convergence numérique est environ de
Rectangle : 1.99
Trapeze : 1.99
Simpson : 1.99
```

Les graphiques sont parallèles à la droite H^2 et donc l'ordre est 2 pour toutes les méthodes. Pour la méthode du point milieu et du trapèze, c'est l'ordre auquel on s'attend d'après la théorie. Par contre, pour Simpson, on pouvait s'attendre à un ordre 4. Le problème vient du fait que la fonction n'est pas très régulière, puisque $f \in C^0([-1,1])$ mais $f \notin C^1([-1,1])$ puisque sa dérivée n'est pas continue en $x = 0$. Dans la théorie, on demande $f \in C^4([-1,1])$ pour assurer l'ordre 4. On obtient tout de même l'ordre 2 car, mis à part en $x = 0$, la fonction f est très régulière.

Partie 5 Si on regarde l'erreur pour la méthode de Simpson avec $N = 99$ et $N = 100$ sous-intervalles

[28]:
```
print (np.abs( Simpson( a, b, 99, f) - Iexact ) )
print (np.abs( Simpson( a, b, 100, f) - Iexact ) )
```

```
1.7004959483424287e-05
3.5117464491918327e-11
```

On obtient des valeurs d'environ $1.70\ 10^{-5}$ et $3.51\ 10^{-11}$. Les erreurs sont très différentes ! Voilà pourquoi:

- si N est pair, le point $x = 0$ est un point x_i (pour $i = N/2$) exactement entre deux sous-intervalles. La fonction f est régulière à droite et à gauche, en particulier, on retrouve l'ordre 4 par rapport à H de chaque côté.
- si N est impair, le point $x = 0$ est au milieu d'un sous-intervalle. La fonction f n'est pas régulière dans cet intervalle, ce qui donnt un ordre de convergence plus petit.

Partie 6 Il faut relancer les parties 2 et 3 avec `N= [2,10,100,1000]` à la place de `N = [1,9,99,999]`

[ ]: