Sol-Assignment 3 - Intégration et dérivation numérique

May 7, 2025

1 Assignment 3: Formule de quadrature d'ordres elévés

On considère une fonction $f : [a, b] \to \mathbb{R}$ dans $C^0([a, b])$. On est intéressé à approcher l'intégrale $I(f) = \int_a^b f(x)\,dx$ en utilisant des formules de quadrature d'ordres elévés.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.special import erf, expi
from scipy.integrate import quad
```

1.1 Partie 1

Dans un premier temps, nous essayons d'approximer l'intégrale de f en exploitant des règles de quadrature simples (c-à-d non composites), de la forme

$$I(f) = \int_a^b f(x)\ dx \approx I_n(f) = \sum_{i=1}^{n} w_i\ f(x_i)$$

Ces régles ont été construites à partir de l'interpolation polynomiale d'ordre élevé sur des nœuds équidistribués, c-à-d

$$I_n(f) = \int_a^b \Pi_n(f)(x)\ dx\ ,$$

où $\Pi_n(f)$ est l'interpolée de Lagrange de f sur des noeuds équidistribués.

Plus précisément, nous considérons : * **nœuds** : équidistribués dans $[a, b]$, donc $x_i = a + i\left(\frac{b-a}{n-1}\right)$
* **poids** : calculés à partir des fonctions de base de Lagrange définies sur les nœuds afin de garantir le degré d'exactitude maximale, donc $w_i = \int_a^b \varphi_i(x)\ dx$.

On considère la fonction

$$\kappa(x) = 1 - \kappa_1(\kappa_2(x)) \qquad \text{avec} \quad \kappa_1(y) = \frac{1}{1 - 2y + 4y^2} \quad \text{et} \quad \kappa_2(x) = \frac{x^2 - 4}{4}\ .$$

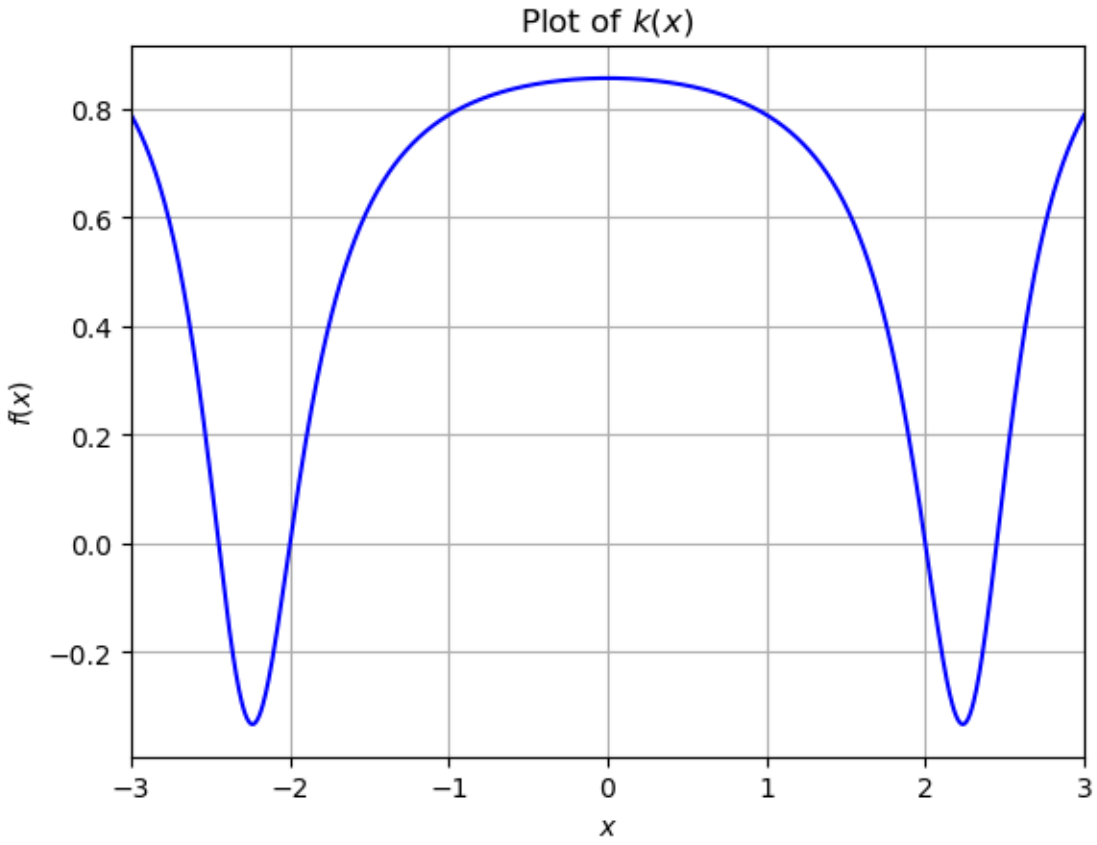
Approximer l'intégrale de κ dans $[-3, 3]$ pour $n = 2^j + 1$ noeuds, avec $j = 1, 2, \cdots, 5$, et comparer le résultat avec une estimation précise de l'intégrale (donnée).

Discuter les résultats obtenus. En particulier, est-ce que l'estimation de l'integrale s'améliore lorsqu'on considère plus de noeuds de quadrature ? Pourquoi ? (*Réponse 1*)

[2]:
```python
def Lagrange(n):
    """
    Compute Lagrange quadrature nodes and weights on equidistant nodes in␣
 ↪[-1,1].

    Inputs: [n]
    n : Number of nodes.

    Outputs: [x, w]
    x : Equidistant nodes in [-1,1].
    w : Quadrature weights.
    """

    x = np.linspace(-1, 1, n)
    w = np.zeros(n)

    for i in range(n):
        # Define Lagrange basis function
        def lagrange_basis(_x):
            res = 1
            for j in range(n):
                if i != j:
                    res *= (_x - x[j]) / (x[i] - x[j])
            return res

        # Integrate the Lagrange polynomial over [-1,1]
        w[i], _ = quad(lagrange_basis, -1, 1)

    return x, w
```

[3]:
```python
k2 = lambda x : (x**2 - 4) / 4
k1 = lambda y : 1 / (1 - 2*y + 4*y**2)
k = lambda x : 1 - k1(k2(x))
a, b = -3, 3

IexactK, _ = quad(k, a, b)
print(f'A precise estimate of the value of the integral is {IexactK:.7f}')

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

plt.plot(x, y, 'b')
plt.xlabel('$x$'); plt.ylabel('$f(x)$')
plt.title(r'Plot of $k(x)$')
plt.xlim([a,b])
plt.grid()
plt.show()
```

```
A precise estimate of the value of the integral is 3.1282214
```



[4]:
```python
n_range = [2**k + 1 for k in range(1,6)]

for n in n_range :
    nodes, weights = Lagrange(n)
    nodes = a + (b-a) / 2 * (nodes + 1)
    weights *= (b-a) / 2

    intQuad = sum(weights * k(nodes))
    errQuad = np.abs(intQuad - IexactK)

    print(f"n ={n:3d} - "
          f"Approximated integral: {intQuad:2.4f} - "
          f"Error: {errQuad:.2e}")
```

```
n =  3 - Approximated integral: 5.0075 - Error: 1.88e+00
n =  5 - Approximated integral: 4.0734 - Error: 9.45e-01
n =  9 - Approximated integral: 2.0960 - Error: 1.03e+00
n = 17 - Approximated integral: 3.3721 - Error: 2.44e-01
```

```
n = 33 - Approximated integral: -2.8408 - Error: 5.97e+00
```

1.1.1 Commentaire

Réponse 1 On remarque que l'approximation de l'integrale n'est jamais bonne et devient pire lorsque le nombre de noeuds n augmente. En genérale, ces grandes erreurs sont liées au fait que le polynôme interpolatoire $\Pi_n(f)$ n'est jamais précis, ce qui dépend du phenomène de Runge pour des valeurs de n suffisamment grandes. En fait, en dessinant le polynôme interpolatoire de Lagrange, on pourrait observer la présence d'oscillations de plus en plus évidentes aux éxtremités de l'intervalle lorsque n croît.

1.2 Partie 2

Maintenant, on considére des règles de quadrature composites, en subdivisant l'intervalle d'intégration $[a, b]$ en N sous-intervalles plus petits.

Écrivez une fonction qui implemente une formule de quadrature composite, étant donnée les noeuds (dans l'intervalle de reference $[-1, 1]$), les poids, les extrèmes de l'intervalle d'integration $[a, b]$ et le nombre de sous-intervalles N.

La fonction doit avoir la structure suivante:

```
def QuadratureComposite(nodes, weights, a, b, N, f) :
    # Function that implements a composite quadrature rule, being given in
    # input the nodes (in [-1,1]), the weights, the integration interval bounds,
    # the number of sub-intervals and the integrand function.

    # Inputs: [nodes, weights, a, b, N, f]
    # nodes    : quadrature nodes (in [-1,1])
    # weights    : quadrature weights
    # [a,b] : integration interval
    # N     : number of sub-intervals
    # f     : function to integrate
    #
    # Outputs : [Lh]
    # Lh      : integral of f in [a,b], approximated with the prescribed composite quadrature
```

[5]:
```
def QuadratureComposite(nodes, weights, a, b, N, f) :
    """Function that implements a composite quadrature rule, being given in
    input the nodes (in [-1,1]), the weights, the integration interval bounds,
    the number of sub-intervals and the integrand function.

    Inputs: [nodes, weights, a, b, N, f]
    nodes   : quadrature nodes (in [-1,1])
    weights : quadrature weights
    [a,b]   : integration interval
    N       : number of sub-intervals
    f       : function to integrate

    Outputs : [Lh]
```

```
    Lh      : integral of f in [a,b], approximated with the prescribed␣
↪composite quadrature rule
    """

    M = len(nodes)
    if len(nodes) != len(weights):
        raise ValueError(f"Invalid value of M: {M}")

    # 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) :
        # Quadrature points in the sub-interval
        z = (x[k] + x[k+1])/2 + nodes*(x[k+1] - x[k])/2

        # local quadrature on the subinterval
        Jgk = sum(weights * f(z))

        Lh += Jgk

    # scaling
    Lh *= H/2

    # approximate integral
    return Lh
```

1.2.1 Formules de quadrature de Gauss-Legendre-Lobatto.

Dans ce test, on prend les nœuds et les poids de quadrature de Gauss-Legendre-Lobatto (GLL), qui conviennent à l'intégration numerique d'ordre élevé.

La fonction `GaussLegendreLobatto` ci-dessous retourne les M noeuds (dans l'intervalle de reférence $[-1, 1]$) et les M poids de quadrature pour les formules de GLL.

Remarque: seulement les cas $M = 2, 3, 4, 5$ sont considerés.

M	noeuds (dans [1,1])	poids (dans le même ordre que les noeuds)
2	$-1, 1$	$1, 1$
3	$-1, 0, 1$	$\frac{1}{3}, \frac{4}{3}, \frac{1}{3}$
4	$-1, \frac{-\sqrt{5}}{5}, \frac{\sqrt{5}}{5}, 1$	$\frac{1}{6}, \frac{5}{6}, \frac{5}{6}, \frac{1}{6}$

M	noeuds (dans [1,1])	poids (dans le même ordre que les noeuds)
5	$-1, \frac{-\sqrt{21}}{7}, 0, \frac{\sqrt{21}}{7}, 1$	$\frac{1}{10}, \frac{49}{90}, \frac{32}{45}, \frac{49}{90}, \frac{1}{10}$

[6]:
```
def GaussLegendreLobatto(M):
    """Returns the GaussLegendreLobatto (GLL) quadrature nodes and weights for␣
 ↪the given value of M.

    Inputs   : [M]
    M        : number of quadrature nodes and weights

    Outputs  :
    nq       : quadrature nodes (in [-1,1])
    wq       : quadrature weights
    """

    if M == 2:
        nq = np.array([-1.0, 1.0])
        wq = np.ones(M)
    elif M == 3:
        nq = np.array([-1.0, 0.0, 1.0])
        wq = np.array([1/3, 4/3, 1/3])
    elif M == 4:
        nq = np.array([-1.0, -np.sqrt(5)/5, np.sqrt(5)/5, 1.0])
        wq = np.array([1/6, 5/6, 5/6, 1/6])
    elif M == 5:
        nq = np.array([-1.0, -np.sqrt(21)/7, 0.0, np.sqrt(21)/7, 1.0])
        wq = np.array([1/10, 49/90, 32/45, 49/90, 1/10])
    else:
        raise ValueError(f"Invalid value of M: {M}")

    return nq, wq
```

1.3 Partie 3

Éstimez numériquement le degré d'exactitude des formules de quadrature simples (i.e. pas composite, $N = 1$ sous-intervalles) de Gauss-Legendre-Lobatto (GLL) avec $M = 4$. Est-ce que les résultats obtenus sont en accord avec la théorie? (*Réponse 2*)

[7]:
```
# Checking quadrature fonction
a, b = 1, 4

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

[8]:
```
# recording for which degrees the integral is exact (up to epsilon)
exactDegree = -1
epsilon = 1e-12

M = 4
nodes, weights = GaussLegendreLobatto(M)

N = 1
for d in range(10) :
    intQuad = QuadratureComposite(nodes, weights, a, b, N, monomial)
    intExact = (4**(d+1) - 1) / (d+1)
    print(f'Quadrature on monomial x^{d} : {intQuad:.4f} - {intExact:.4f}  =␣
 ↪{intQuad-intExact:.6e}')

    if np.abs(intQuad-intExact) < epsilon :
        exactDegree = d

print(f'\nGLL composite quadrature with N = {N} and M = {M} is exact up to␣
 ↪degree {exactDegree}')
```

```
Quadrature on monomial x^0 : 3.0000 - 3.0000  = 0.000000e+00
Quadrature on monomial x^1 : 7.5000 - 7.5000  = 1.776357e-15
Quadrature on monomial x^2 : 21.0000 - 21.0000  = 0.000000e+00
Quadrature on monomial x^3 : 63.7500 - 63.7500  = 0.000000e+00
Quadrature on monomial x^4 : 204.6000 - 204.6000  = 2.842171e-14
Quadrature on monomial x^5 : 682.5000 - 682.5000  = 2.273737e-13
Quadrature on monomial x^6 : 2341.4700 - 2340.4286  = 1.041429e+00
Quadrature on monomial x^7 : 8210.1000 - 8191.8750  = 1.822500e+01
Quadrature on monomial x^8 : 29313.6240 - 29127.0000  = 1.866240e+02
Quadrature on monomial x^9 : 106322.7900 - 104857.5000  = 1.465290e+03

GLL composite quadrature with N = 1 and M = 4 is exact up to degree 5
```

1.3.1 Commentaire

Réponse 1 Le degré d'exactitude des formules de quadrature de Gauss-Legendre-Lobatto (GLL) est $2M-3$. Dans ce cas, étant $M=4$, on s'attend donc un degré d'exactitude égal à 5. Ce résultat est en effet confirmé par notre test numerique.

1.4 Partie 4

Considerons maintenant la fonction $f(x) = e^{-x^2} + \dfrac{e^x}{x}$.

Approximer $I(f) = \displaystyle\int_{1/e}^{e} f(x)\,dx$ en utilisant les formules de quadrature de GLL composites avec $M = 2, 3, 4, 5$ et $N = 2^j$ sous-intervalles pour $j = 3, 4, 5, 6$.

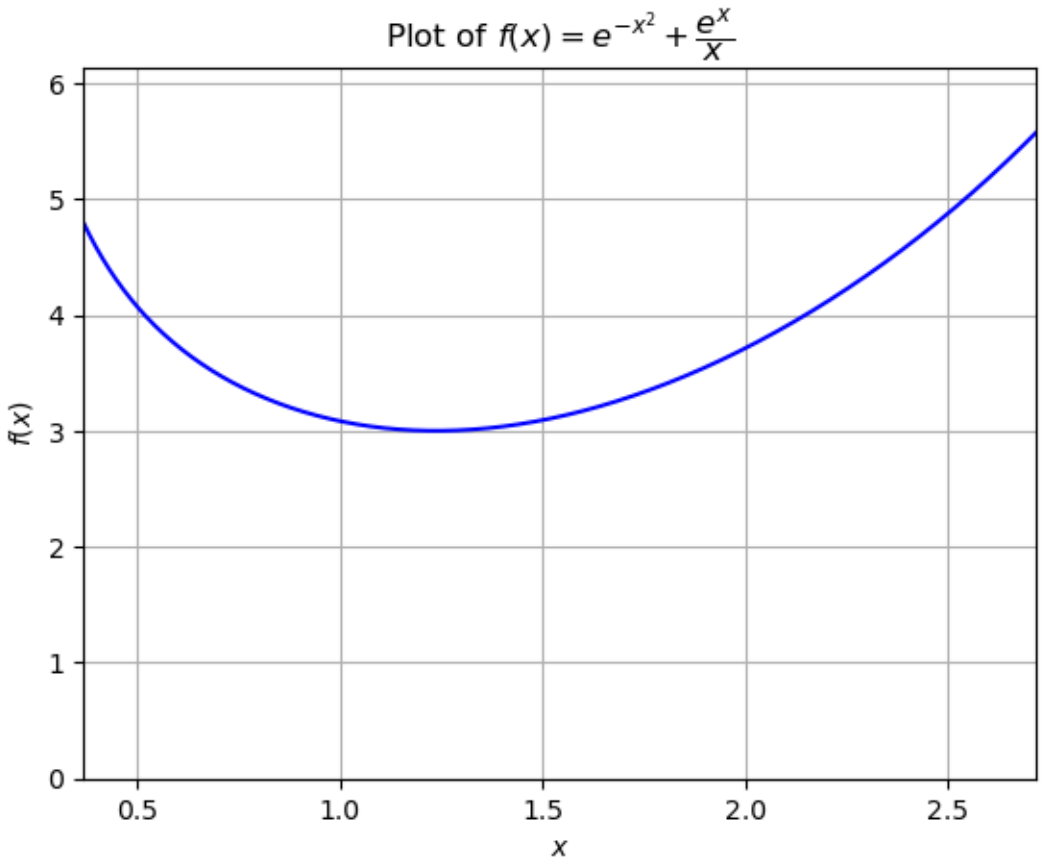
Ensuite calculer les erreurs $E_{GLL}^{M,N}(f) := |I(f) - I_{GLL}^{M,N}(f)|$, où $I_{GLL}^{M,N}(f)$ represente l'approximation

de l'integral de f obtenue avec la formule de quadrature de GLL composite, avec N sous-intervalles et de degré M.

Dessiner les erreurs en fonction de H sur une échelle logarithmique sur les deux axes. Que peut-on deduire par rapport aux ordres de convergence ? Sont-ils en accord avec la théorie? Motivez votre réponse (*Réponse 3*).

[9]:
```
f = lambda x : np.exp(-x**2) + np.exp(x) / x

a, b = 1 / np.exp(1), np.exp(1)
IexactF = (erf(b) - erf(a)) * np.sqrt(np.pi) / 2 + (expi(b) - expi(a))
print(f'The exact value of the integral of f is {IexactF:.7f}')

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

plt.plot(x, y, 'b')
plt.xlabel('$x$'); plt.ylabel('$f(x)$')
plt.title(r'Plot of $f(x) = e^{-x^2} + \dfrac{e^x}{x}$')
plt.ylim([0, 1.1*np.max(y)])
plt.xlim([a,b])
plt.grid()
plt.show()
```

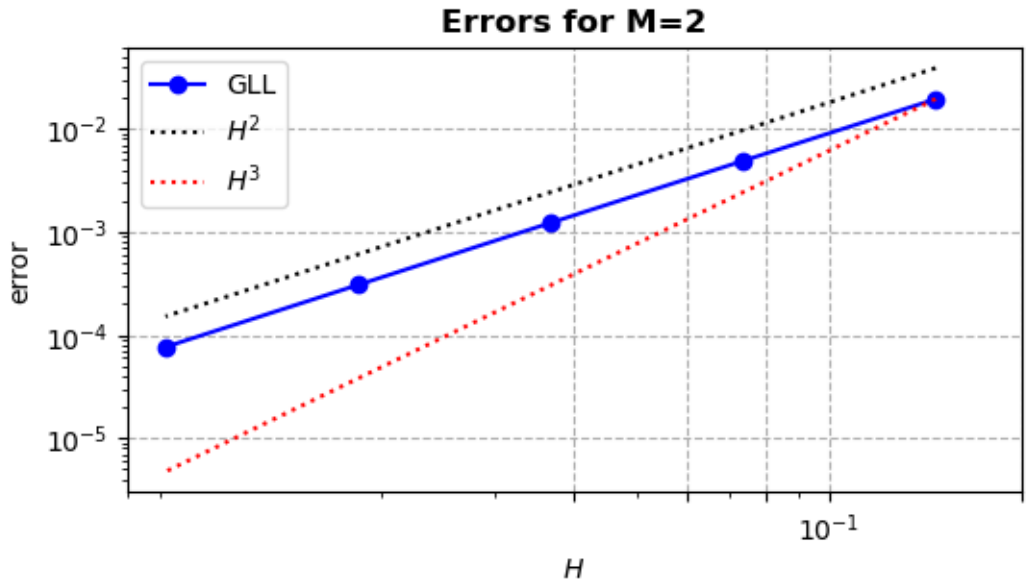
```
The exact value of the integral of f is 8.7639675
```

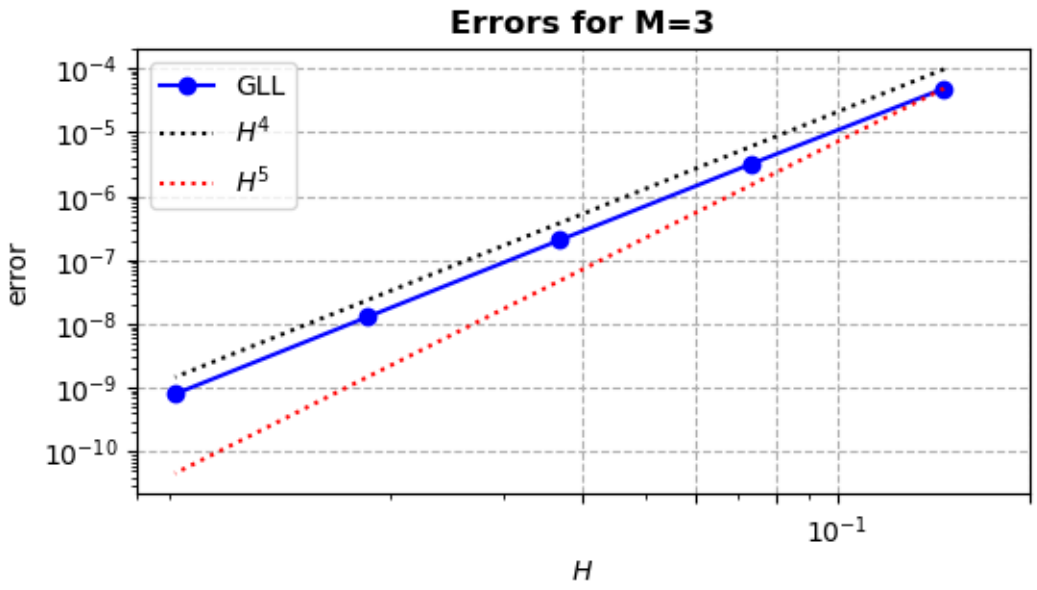


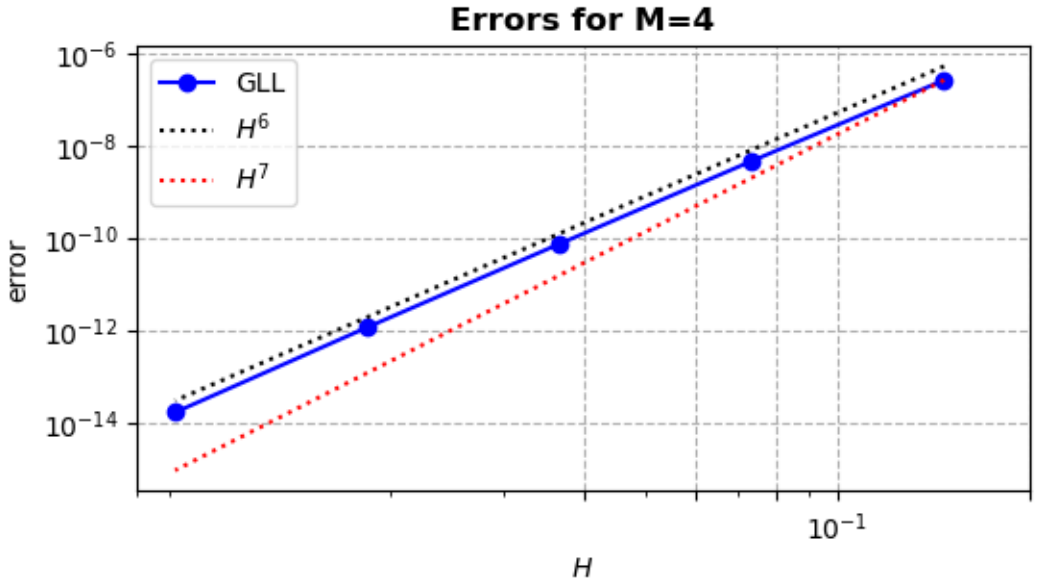
[10]:
```
Mrange = np.array([2, 3, 4, 5])
Nrange = np.array([2**k for k in range(4, 9)])

for M in Mrange :
    nodes, weights = GaussLegendreLobatto(M)
    errQuad = []

    for N in Nrange :
        intQuad = QuadratureComposite(nodes, weights, a, b, N, f)
        errQuad.append( np.abs( intQuad - IexactF ) )

    H = (b-a)/Nrange
    slopeQuad = ( np.log(errQuad[-1]) - np.log(errQuad[0]) ) / ( np.log(H[-1])␣
 ↪- np.log(H[0]) )

    print(f'Pour M = {M}, La convergence numérique est environ de {slopeQuad:.
 ↪2f}')

    plt.figure(figsize=(6,3))
```

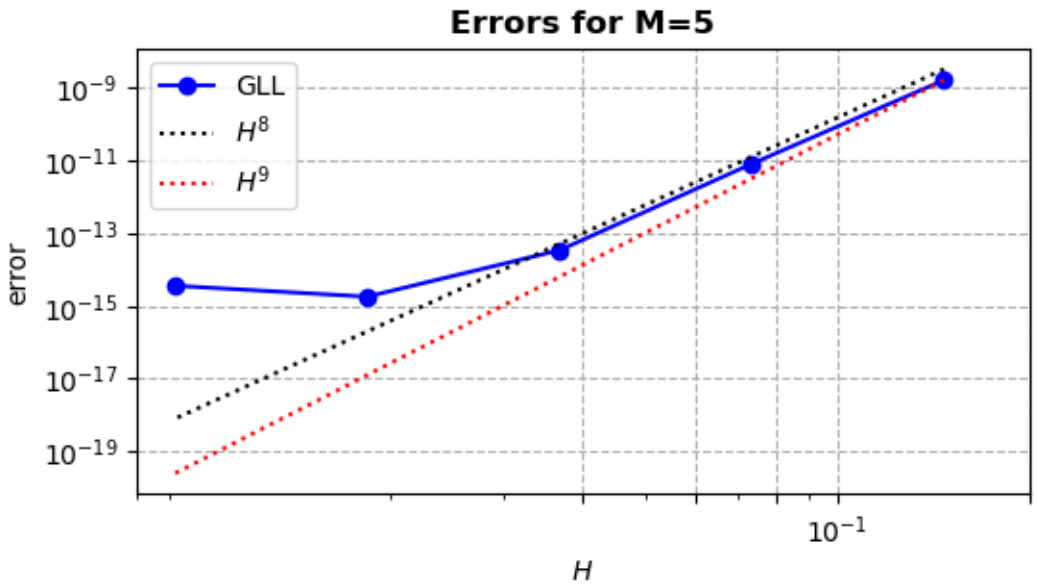
```
    plt.loglog(H, errQuad, 'b-o')

    plt.loglog(H, H**(2*M-2) * (errQuad[0]/H[0]**(2*M-2))*2, 'k:')
    plt.loglog(H, H**(2*M-1) * (errQuad[0]/H[0]**(2*M-1)), 'r:')

    plt.title(f"Errors for M={M}", fontweight='bold')
    plt.legend(['GLL', f'$H^{2*M-2}$', f'$H^{2*M-1}$'])
    plt.xlabel(r'$H$'); plt.ylabel('error')

    xticks = [4*10**(-2), 6*10**(-2), 8*10**(-2), 10**(-1), 2*10**(-1)]
    plt.xticks(xticks)
    plt.grid(which='major', linestyle='--')
```

```
Pour M = 2, La convergence numérique est environ de 2.00
Pour M = 3, La convergence numérique est environ de 3.97
Pour M = 4, La convergence numérique est environ de 5.96
Pour M = 5, La convergence numérique est environ de 4.71
```




Errors for M=3
GLL
H^4
H^5
error
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-1}
H


Errors for M=4
GLL
H^6
H^7
error
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-1}
H



1.4.1 Commentaire

Réponse 3 L'ordre de convergence des formules de quadrature de Gauss-Legendre-Lobatto (GLL) par rapport à H est $2M-2$. Dans ces cas, on s'attend donc * une convergence d'ordre 2 par rapport à H si $M = 2$; * une convergence d'ordre 4 par rapport à H si $M = 3$; * une convergence d'ordre 6 par rapport à H si $M = 4$; * une convergence d'ordre 8 par rapport à H si $M = 5$;

Nos attentes sont effectivement satisfaites par les résultats numeriques. D'abord, cela peut être déduit en regardant les graphiques ci-dessus, où, pour toute valeur de M, la courbe d'erreur de GLL est (presque) parallèle à la droite d'erreur avec pente (en echelle logarithmique) égal à $2M-2$. De plus, on peut estimer l'ordre de convergence numerique en calculant la pente des courbes d'erreur en fonction de H. On obtient les valeurs suivantes: * 2.00 (≈ 2) pour $M = 2$; * 3.97 (≈ 4) pour $M = 3$; * 5.96 (≈ 6) pour $M = 4$; * 7.79 (≈ 8) pour $M = 5$.

Ces valeurs sont bien en accord avec la théorie.

Remarque: Pour le cas $M = 5$, on ne considère que le premiers deux points pour determiner l'ordre, car pour des valeurs trop petites de n l'erreur devient approximativement constante et $\approx 10^{-15}$.

1.5 Partie 5

Répéter la Partie 4, mais en considerant la fonction $g_K : [-1, 1] \to \mathbb{R}$ definie comme suit

$$g_K(x) = \begin{cases} e^x & \text{if } x \leq -K \\ c_0 x^2 + c_1 |x| & \text{if } -K < x \leq K \\ e^{-x} & \text{if } x > K \end{cases} \tag{1}$$

avec $K = 0.70$, $c_0 = -\frac{K+1}{K^2} e^{-K}$, $c_1 = e^{-K}\left(1 + \frac{2}{K}\right)$ et en prenant $N = 2^k$, $k = 3, 4, 5, 6$ sous-intervalles.

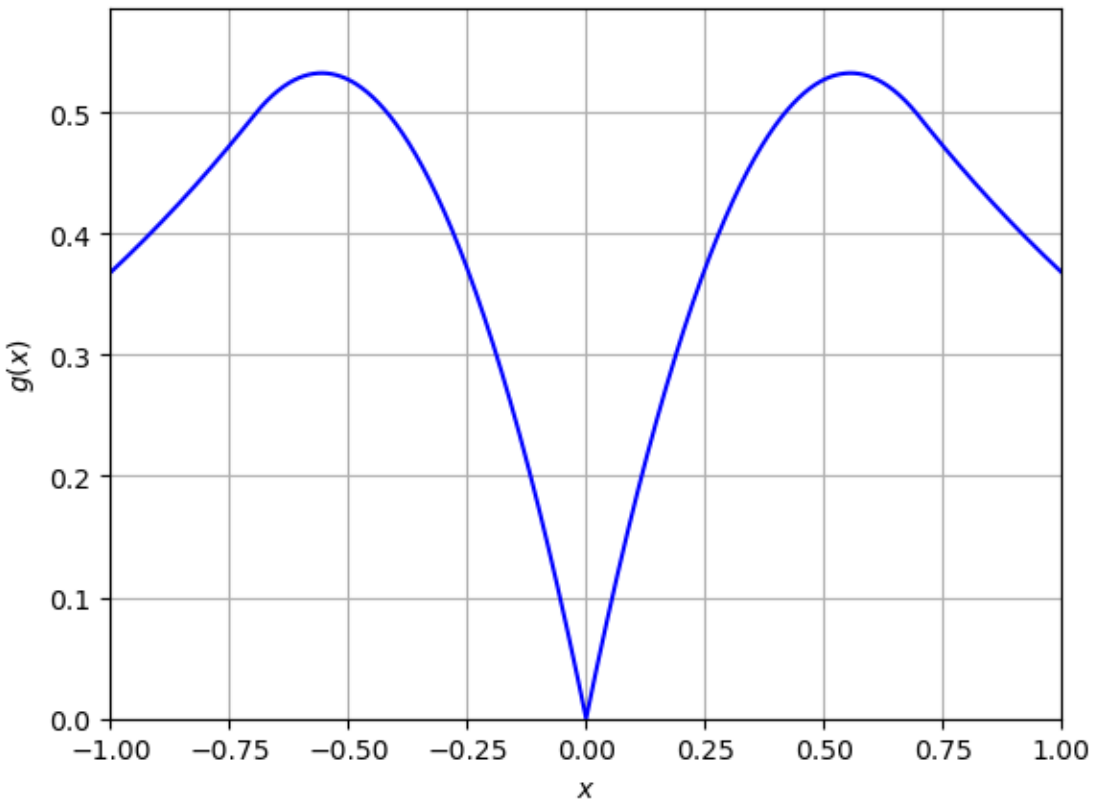
On a que l'integrale exacte de g_K est égale à

$$I_{g_K} = \int_{-1}^{1} g_K(x)\, dx = 2e^{-K} - \frac{2}{e} + \frac{2}{3}c_0K^3 + bK^2 \ .$$

Quels sont les ordres de convergence numérique obtenus? Sont-ils égaux à ceux obtenus avec la fonction f? Pourquoi?(*Réponse 4*)

[11]:
```
a,b = -1, 1
K = 0.70
c0 = -(K+1)/K**2 * np.exp(-K)
c1 = (1+2/K) * np.exp(-K)

g = lambda x : (np.exp(x)*(x<=-K) +
                (c0 * x**2 + c1 * np.abs(x))*(x>-K)*(x<=K) +
                np.exp(-x)*(x>K))

IexactG = (2*np.exp(-K) - 2*np.exp(-1) + 2*c0/3*K**3 + c1*K**2) / 1 # exact␣
 ↪value of the integral of g
print(f'The exact value of the integral of g is {IexactG:.7f}')

x = np.linspace(a,b,1001)
y = g(x)

plt.plot(x, y, 'b')
plt.xlabel(r'$x$'); plt.ylabel(r'$g(x)$')
plt.ylim([0, 1.1*np.max(y)])
plt.xlim([a,b])
plt.grid()
plt.show()
```

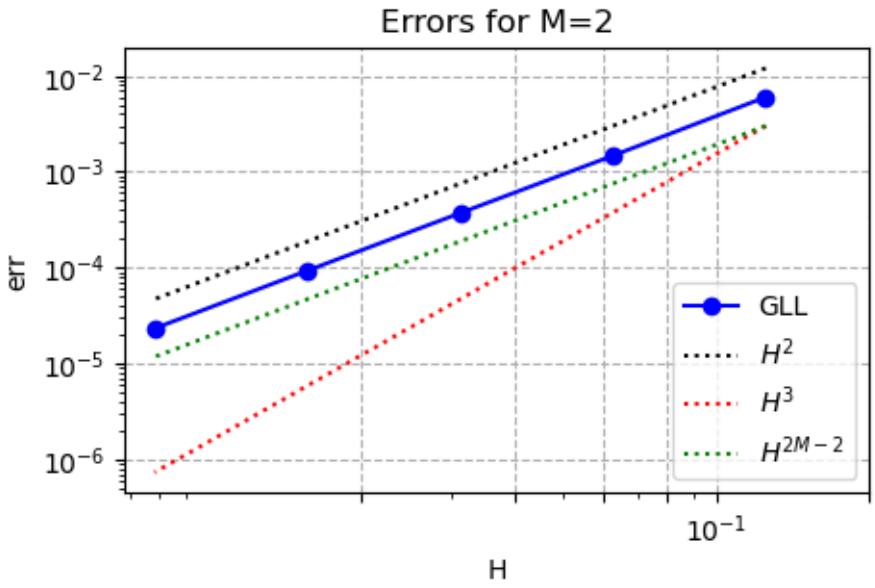
```
The exact value of the integral of g is 0.8020003
```

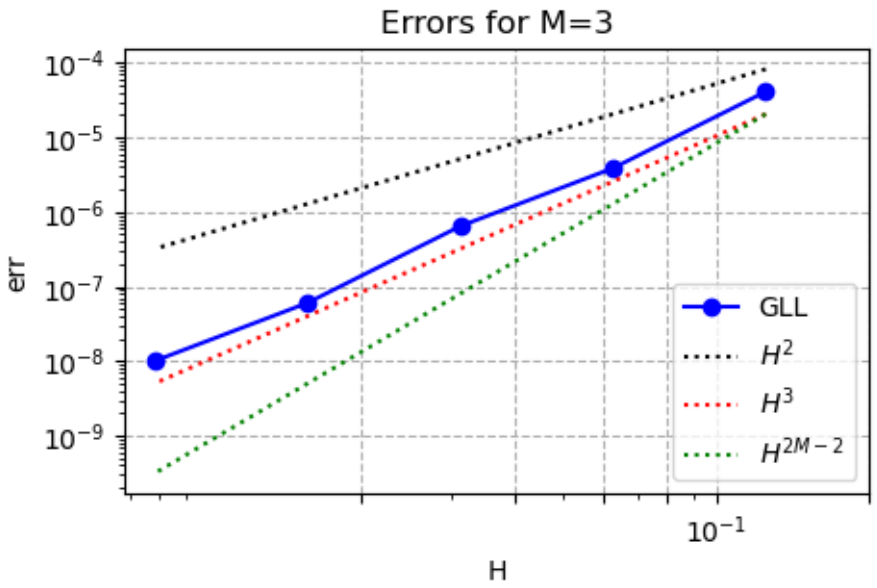


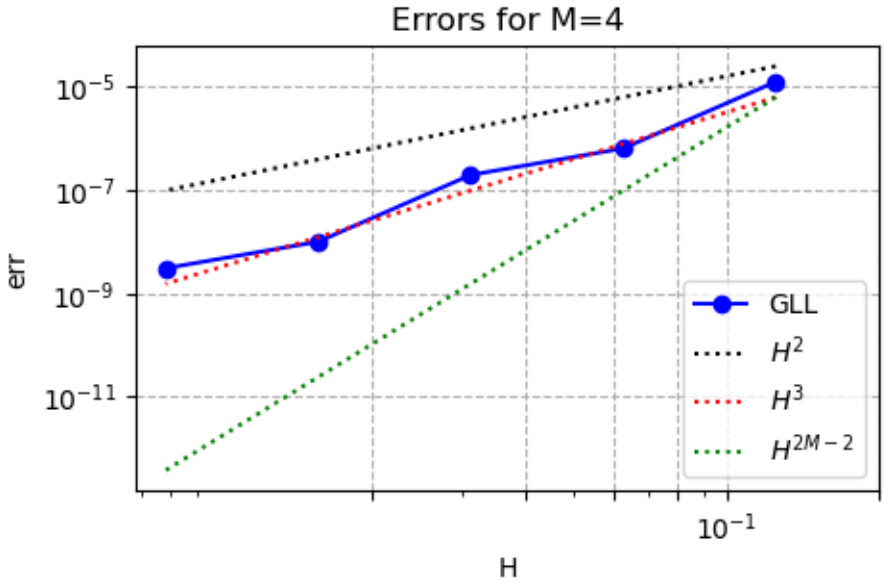
[12]:
```
Mrange = np.array([2, 3, 4, 5])
Nrange = np.array([2**k for k in range(4,9)])

for M in Mrange :
    nodes, weights = GaussLegendreLobatto(M)
    errQuad = []

    for N in Nrange:
        intQuad = QuadratureComposite(nodes, weights, a, b, N, g)
        errQuad.append( np.abs( intQuad - IexactG ) )

    H = (b-a)/Nrange
    slopeQuad = ( np.log(errQuad[-1]) - np.log(errQuad[0]) ) / ( np.log(H[-1])␣
 ↪- np.log(H[0]) )

    print(f'Pour M = {M}, La convergence numérique est environ de {slopeQuad:.
 ↪2f}')

    plt.figure(figsize=(5,3))

    plt.loglog(H, errQuad, 'b-o')
```

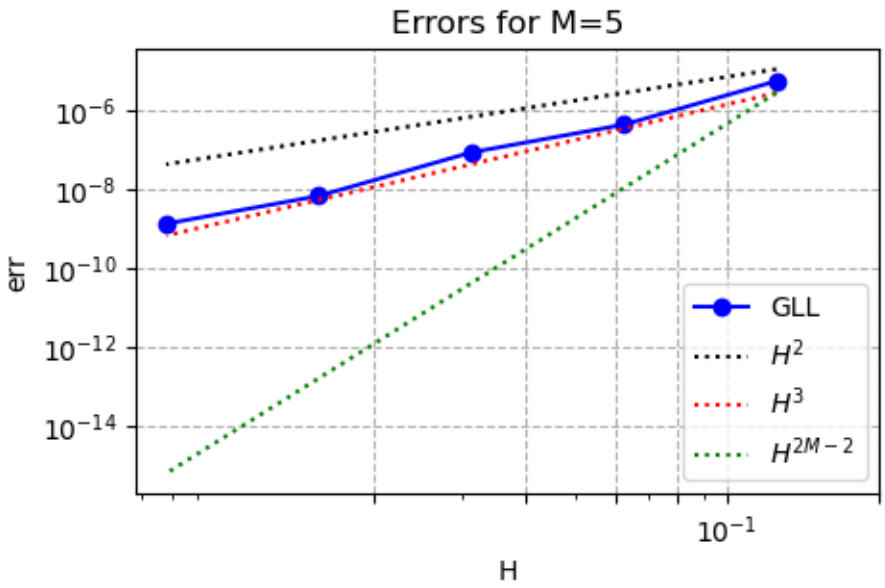
```
    plt.loglog(H, H**2 * (errQuad[0]/H[0]**2)*2, 'k:')
    plt.loglog(H, H**3 * (errQuad[0]/H[0]**3)/2, 'r:')
    plt.loglog(H, H**(2*M-2) * (errQuad[0]/H[0]**(2*M-2))/2, 'g:')

    plt.title(f"Errors for M={M}")
    plt.legend(['GLL', '$H^2$', '$H^3$', '$H^{2M-2}$'])
    plt.xlabel('H'); plt.ylabel('err')

    xticks = [2*10**(-2), 4*10**(-2), 6*10**(-2), 8*10**(-2), 10**(-1),␣
↪2*10**(-1)]
    plt.xticks(xticks)
    plt.grid(which='major', linestyle='--')
```

```
Pour M = 2, La convergence numérique est environ de 2.00
Pour M = 3, La convergence numérique est environ de 3.00
Pour M = 4, La convergence numérique est environ de 3.00
Pour M = 5, La convergence numérique est environ de 3.00
```




Errors for M=3
err
H
GLL
H^2
H^3
H^{2M-2}


Errors for M=4
err
H
GLL
H^2
H^3
H^{2M-2}



1.5.1 Commentaire

Réponse 4 La courbe d'erreur n'est parallele à la droite avec pente $2M-2$ que pour $M=2$. Pour des valeurs supérieures de M, en fait, on obtient des courbes d'erreur avec pente égale à 3. En genérale, on obtient donc des ordres de convergence inférieurs à ceux obtenus avec la fonction f.

Le problème vient du fait que la fonction g_K n'est pas assez régulière, puisque $g_K \in \mathcal{C}^0([-1,1])$ mais $g_K \notin \mathcal{C}^1([-1,1])$, car sa dérivée n'est pas continue en $x=0$. De plus, la dérivée deuxième de g_K n'est pas continue en $x=\pm K$. En fait, afin d'obtenir un ordre de convergence égal à $2M-2$ (c-à-d l'ordre qu'on s'attend aprés de la théorie), il faudrait que $g_K \in \mathcal{C}^{2M}([-1,1])$. Cependant, on obtient tout de même un ordre compris 3 (sauf que pour $M=2$) car, mis à part en $x=0,\pm K$, la fonction g_K est très régulière.