Sol-Assignment 2 - Interpolation

May 7, 2025

1 Assignment 2: Interpolation de Lagrange

On considère la fonction

$$f(x) = 1 - h(g(x)) \qquad \text{avec} \quad h(y) = \frac{1}{1 - 2y + 4y^2} \quad \text{et} \quad g(x) = \frac{x^2 - 4}{4}$$

sur l'intervalle $I = [-3, 3]$.

1.1 Partie 1

1.1.1 Partie 1a

Calculer le polynôme d'interpolation $\Pi_n f(x)$ de degré $n = 20$

- avec des noeuds équidistribués ;
- avec les noeuds de Chebyshev ;

et comparer graphiquement les résultats obtenus avec la fonction donnée. Est-ce que l'interpolation est toujours précise? Pourquoi? Y aurait-il des différences si nous utilisions l'interpolation polynômiale à l'aide de la matrice de Vandermonde au lieu de l'interpolation de Lagrange? (*Réponse 1*)

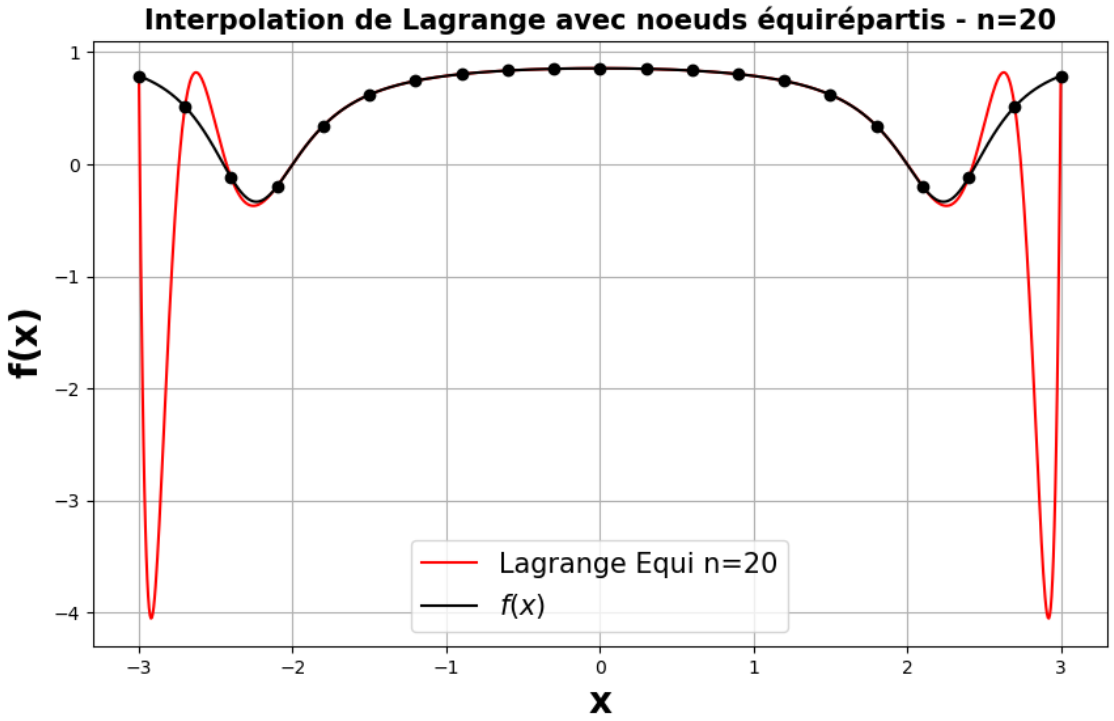
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# HELPER FUNCTIONS FOR LAGRANGE INTERPOLATION  -- THESE ARE GIVEN TO YOU,␣
 ↪NOTHING TO COMPLETE !

def LagrangeBasis(t,k,z):
    """Evaluate the 'k'-th Lagrange basis function at 'z',
       given the interpolatory points 't'.
    INPUT:
        t : array of interpolatory points
        k : number of the basis function
        z : array of points where to evaluate the basis
    OUTPUT:
        result: evaluation of the Lagrange basis function at 'z'
    """

    n = t.shape[0] - 1
```

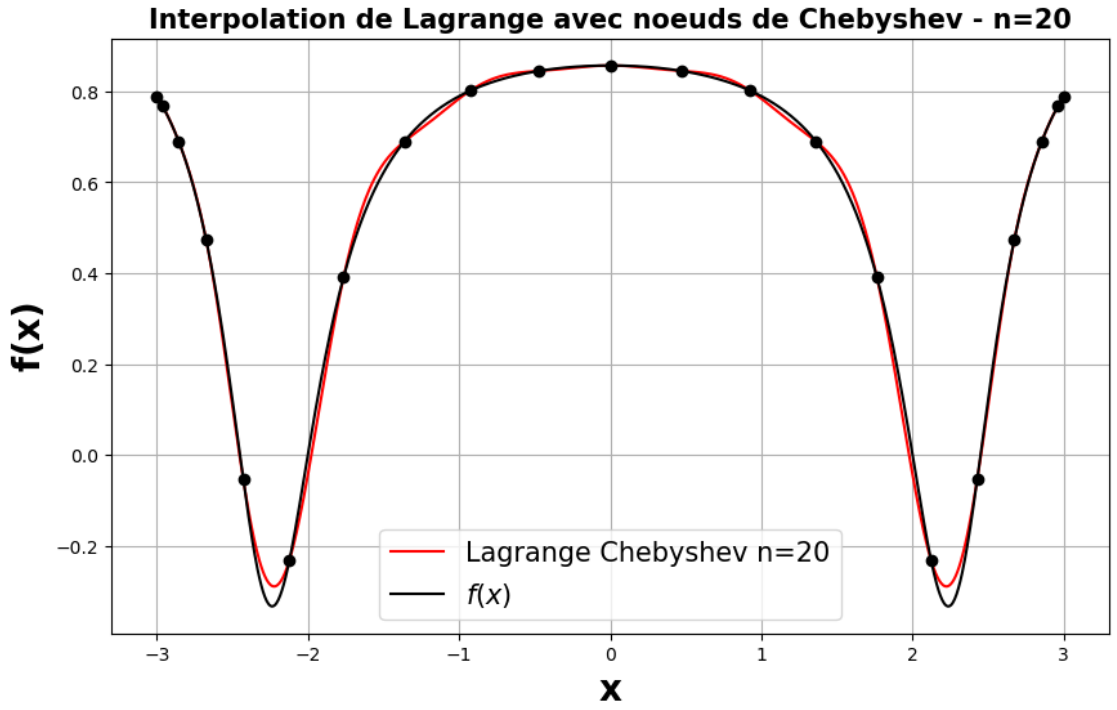
```
    # init result to one, of same type and size as z
    result = np.zeros_like(z) + 1

    # first few checks on k:
    if (type(k) is not int) or (t.shape[0] < 1) or (k > n) or (k < 0):
        raise ValueError('Lagrange basis needs a positive integer k, smaller␣
 ↪than the size of t')

    # loop on n to compute the product
    for j in range(n+1) :
        if (j == k) :
            continue
        if (t[k] == t[j]) :
            raise ValueError('All the interpolation points need to be distinct')

        result *= (z - t[j]) / (t[k] - t[j])

    return result


def LagrangeInterpolation(t,y,z):
    """Evaluate the Lagrange interpolant function at 'z',
       given the interpolatory points 't' and the data 'y'.
    INPUT:
        t : array of interpolatory points
        y : array of data to be interpolated
        z : array of points where to evaluate the interpolant
    OUTPUT:
        result: evaluation of the interpolant function at 'z'
    """

    # {phi(t,k,.), k=0,...,n} is a basis of the polynomials of degree n
    # y represents the coordinates of the interpolating polynomial with respect␣
 ↪to this basis.
    # Therefore LagrangePi(t,y,.) = y[0] phi(t,0,.) + ... + y[n] phi(t,n,.)

    n = t.size - 1

    # init result to zero, of same type and size as z
    result = np.zeros_like(z)

     # loop on n to compute the product
    for k in range(t.shape[0]) :
        result += y[k] * LagrangeBasis(t,k,z)

    return result
```

[3]:
```
# definition of the function to be interpolated
g1 = lambda x : (x**2 - 4) / 4
h = lambda y : 1 / (1 - 2*y + 4*y**2)
f = lambda x : 1 - h(g1(x))
a, b = -3, 3
```

[4]:
```
# EQUIDISTRIBUTED NODES - PLOT
x = np.linspace(a,b,1000) # fine partition, for nice plotting
n=20
xp = np.linspace(a, b, n+1) # interpolation partition
fn=LagrangeInterpolation(xp, f(xp), x)

plt.figure(figsize=(10,6))
plt.plot(x, fn ,'r')
plt.plot(x, f(x), 'k')
plt.plot(xp, f(xp), 'ok')
plt.xlabel(r'$\mathbf{x}$', fontsize=20)
plt.ylabel(r'$\mathbf{f(x)}$', fontsize=20)
plt.legend([f"Lagrange Equi n={n}", '$ f(x)$'], fontsize=15)
plt.grid()
plt.title(f"Interpolation de Lagrange avec noeuds équirépartis - n={n}",
          fontsize=15, fontweight="bold")
plt.show()
```



[5]:
```
# CHEBYSHEV NODES - PLOT
x = np.linspace(a,b,1000) # fine partition, for nice plotting
n=20

k=np.arange(n+1)
t=-np.cos(np.pi*k/n) # Creates the Chebyshev nodes in the interval [-1,1]
xc=((b-a)/2.)*t+(a+b)/2. # Affine rescaling to the interval [a,b]
fn=LagrangeInterpolation(xc, f(xc), x)

plt.figure(figsize=(10,6))
plt.plot(x, fn ,'r')
plt.plot(x, f(x), 'k')
plt.plot(xc, f(xc), 'ok')
plt.xlabel(r'$\mathbf{x}$', fontsize=20)
plt.ylabel(r'$\mathbf{f(x)}$', fontsize=20)
plt.legend([f"Lagrange Chebyshev n={n}", '$ f(x)$'], fontsize=15)
plt.grid()
plt.title(f"Interpolation de Lagrange avec noeuds de Chebyshev - n={n}",
          fontsize=15, fontweight="bold")
plt.show()
```



1.1.2 Commentaire

Réponse 1 *L'interpolation de Lagrange avec des noeuds équidistribués souffre du phénomène de Runge.* Le polynôme d'interpolation présente des oscillations dans les régions extrêmes de l'intervalle, ce qui entraîne une erreur d'interpolation importante. Ceci est dû au fait que le degré du polynome d'interpolation ($n = 20$) est assez élevé. En particulier, la magnitude de la 21-ième dérivée de la fonction est impliquée dans l'expression de la limite supérieure de l'erreur d'interpolation. Ce phénomène prouve que les polynômes de haut degré ne sont pas adaptés à l'interpolation sur des noeuds équirepartis.

Le phénomène de Runge peut être atténué en changeant les points d'interpolation. En particulier, *l'utilisation des noeuds de Chebyshev permet d'obtenir une bien meilleure interpolation de la fonction* en question, même aux extrema de l'intervalle.

En considérant l'interpolation polynomiale à l'aide de la matrice de Vandermonde, *très peu de différences sont à prévoir*. En effet, les deux méthodes, même si différentes, sont théoriquement équivalentes en ce qu'elles produisent le même polynôme d'interpolation.

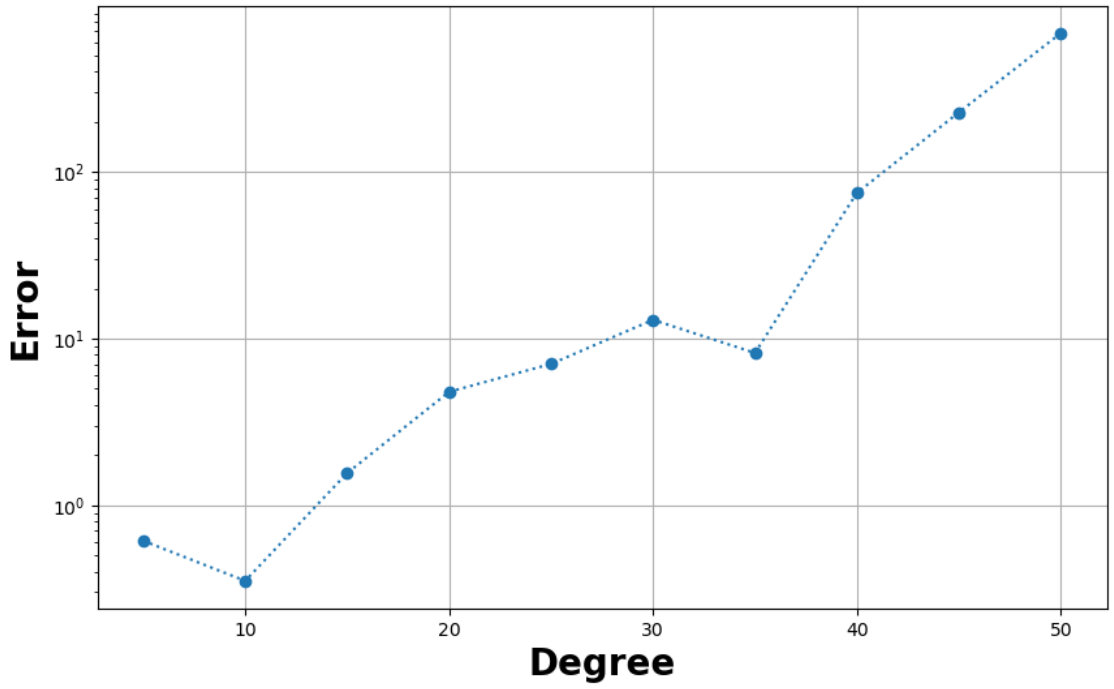
1.1.3 Partie 1b

Èvaluer les erreurs d'interpolation

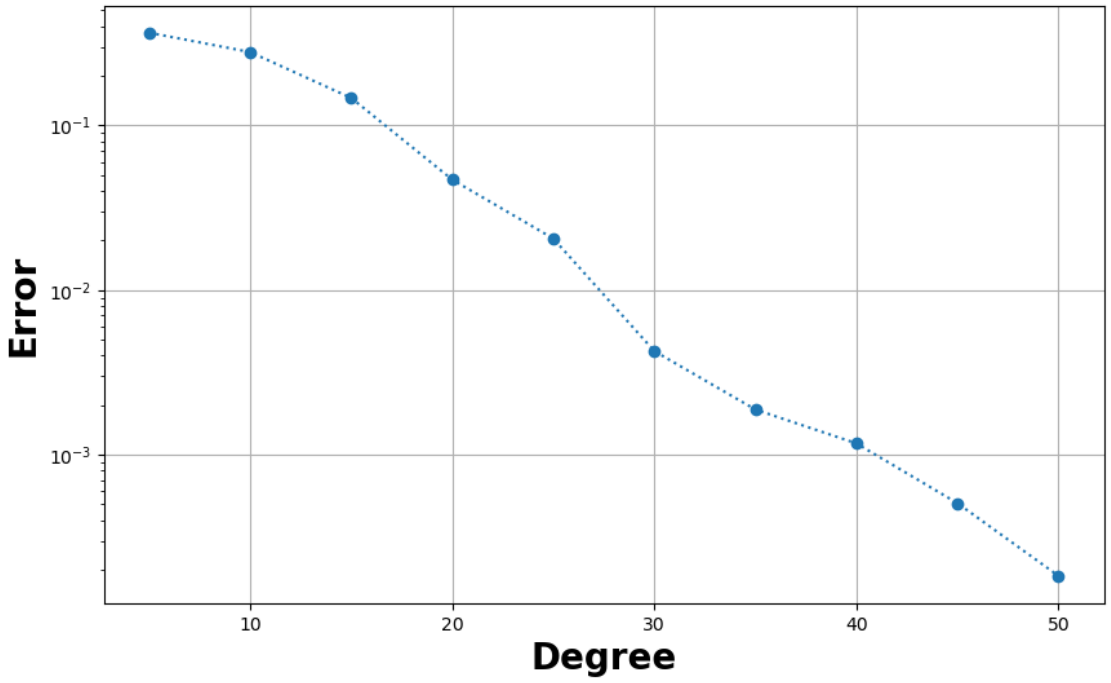
$$E_n^a(f) = \max_{x\in I} |f(x) - \Pi_n f(x)| \quad \text{(erreur absolue)} \qquad E_n^r(f) = \max_{x\in I} \frac{|f(x) - \Pi_n f(x)|}{|f(x)|} \quad \text{(erreur relative)}$$

Visualiser le graphe logarithmique de l'erreur absolue E_n^a en fonction de n pour les noeuds équirépartis et pour les noeuds de Chebyshev, avec $n = 5, 10, 15, \ldots, 50$. Est-ce que les résultats sont en accord avec la théorie vue au cours ? (*Réponse 2*)

[6]:
```
# EQUIDISTRIBUTED NODES - ERRORS
Nrange = np.arange(5,51,5)
errorLag = []
errorLagRel = []
for n in Nrange:
    xp = np.linspace(a, b, n+1);
    fn = LagrangeInterpolation(xp, f(xp), x)
    errorLag.append(np.max(np.abs(fn-f(x))))
    errorLagRel.append(np.max(np.abs(fn-f(x)) / np.abs(f(x))))

plt.figure(figsize=(10,6))
plt.plot(Nrange, errorLag, ':o');
plt.yscale('log')
plt.xlabel('Degree', fontsize=20, fontweight='bold')
plt.ylabel('Error', fontsize=20, fontweight='bold')
plt.grid()
plt.show()
```



[7]:
```
# CHEBYSHEV NODES - ERRORS
Nrange = np.arange(5,51,5)
errorChe = []
errorCheRel = []
for n in Nrange:
    k=np.arange(n+1)
    t=-np.cos(np.pi*k/n)
    xc = ((b-a)/2)*t+(a+b)/2
    fn = LagrangeInterpolation(xc, f(xc), x)
    errorChe.append(np.max(np.abs(fn -f(x))))
    errorCheRel.append(np.max(np.abs(fn-f(x)) / np.abs(f(x))))

plt.figure(figsize=(10,6))
plt.plot(Nrange, errorChe, ':o')
plt.yscale('log')
plt.xlabel('Degree', fontsize=20, fontweight='bold')
plt.ylabel('Error', fontsize=20, fontweight='bold')
plt.grid()
plt.show()
```



1.1.4 Commentaire

Réponse 2 Comme prévu, l'erreur augmente à mesure que le degré du polynôme croît avec des noeuds équidistants, car la magnitude de la dérivée d'ordre $(n+1)$ de f, impliquée dans l'expression de la borne supérieure d'erreur, croît rapidement avec n. Néanmoins, une tendance nettement décroissante de l'erreur peut être observée en passant aux noeuds de Chebyshev.

1.2 Partie 2

Ècrire une fonction `PiecewiseInterpolation`, qui implemente l'interpolation par intervalles. La fonction a la structure suivante

```
def PiecewiseInterpolation(N, LocalOrder, a, b, f, z):
    """
    This function implements piecewise interpolation considering (1) a user-defined number of

    INPUTS:
    N: the number of subintervals
    LocalOrder : order of the polynomial in each subinterval
    a,b : global extrema of the interval
    f : the values of the function at the interpolation nodes
    z : where to evaluate the function

    OUTPUTS:
    result: the evaluation of the piecewise interpolant of f at z
    """
```

[8]:
```
from collections.abc import Iterable

def PiecewiseInterpolation(N, LocalOrder, a, b, f, z):
    """
    This function implements piecewise interpolation considering (1) a user-defined number of subintervals and
    (2) a user-defined polynomial degree in each subinterval.

    INPUTS:
    N: the number of subintervals
    LocalOrder : order of the polynomial in each subinterval
    a,b : global extrema of the interval
    f : the values of the function at the interpolation nodes
    z : where to evaluate the function

    OUTPUTS:
    result: the evaluation of the piecewise interpolant of f at z
    """

    def localInterpolation(zk):
        """Interpolates the function f at the single point zk
        """

        # find out in which interval lies the point zk
        i = 0
        for k in range(x.shape[0] - 1):
            if x[k] <= zk <= x[k+1]:
                break
            else:
                i += 1

        # hereafter is an alternative code to detect the interval
        # i = np.where([x[k] <= zk <= x[k+1] for k in range(x.shape[0]-1)])[0][0]

        # if zk is not in the interval, return constant function
        if zk == x[i]:
            return f(x[i])
        elif zk == x[i+1]:
            return f(x[i+1])

        # otherwise, compute the local interpolation nodes between x[i] and x[i+1]
        x_loc = np.linspace(x[i], x[i+1], LocalOrder+1)

        # use formula for the standard interpolation
        return LagrangeInterpolation(x_loc, f(x_loc), zk)
```

```
    H = (b-a)/N
    if np.isclose(H,0) :
        raise ValueError('PiecewiseInterpolation : a and b are too close!')

    # Intervals of constant length
    x = np.linspace(a,b,N+1)

    if not isinstance(z, Iterable):
        z = [z]

    result = np.zeros_like(z)
    for k, zk in enumerate(z) :
        result[k] = localInterpolation(zk)

    return result
```

1.3 Partie 3

Considerer des noeuds équidistribués.

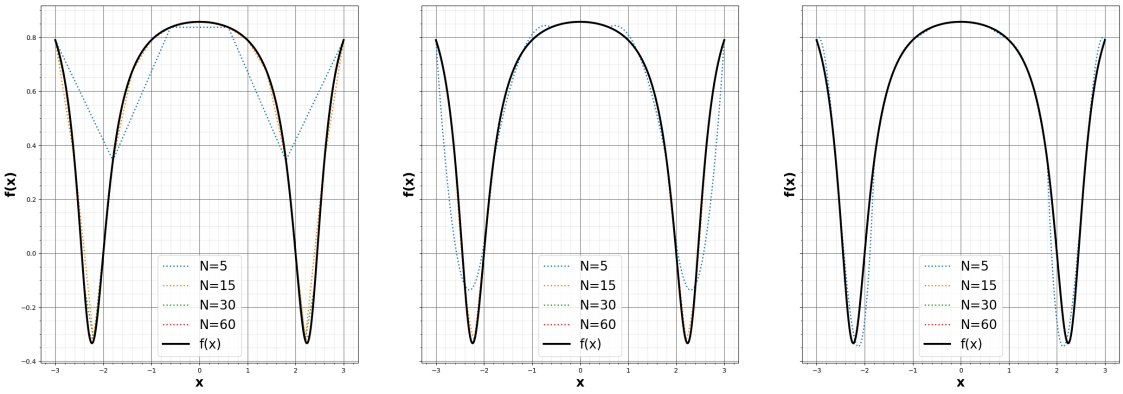
Calculer les polynômes interpolatoires par intervalles d'ordre 1,2,3 (c-à-d $\Pi_1^H f(x), \Pi_2^H f(x), \Pi_3^H f(x)$) sur N sous-intervalles de longeur $H = \frac{b-a}{N}$, avec $N = 5, 15, 30, 60$ et comparer graphiquement les résultats obtenus avec la fonction donnée.

Prendre $N = 5, 10, 15, \ldots, 50$ (nombre de sous-intervalles). Calculer les polynômes interpolatoires par intervalles d'ordre 1,2,3 et évaluer les erreurs absolue et rélative (cf. Section 1). Visualiser les graphes logarithmiques des erreures absolues E_H^a en fonction de N. Est-ce que l'erreur diminue en utilisant l'interpolation par intervalles? Pourquoi? (*Réponse 3*)

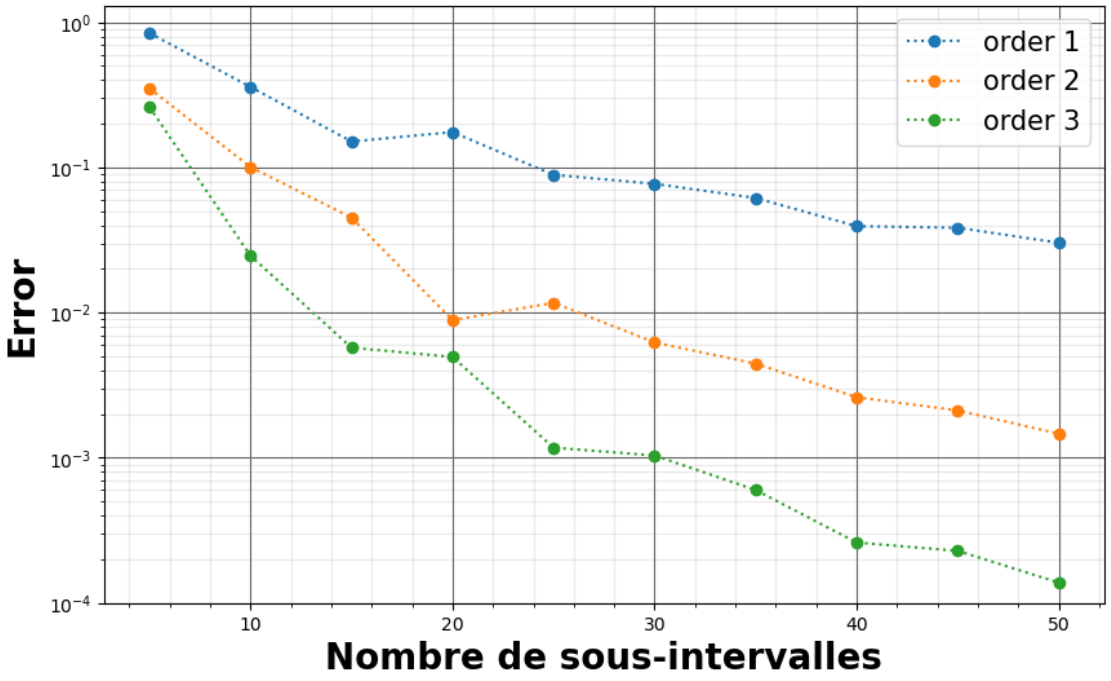
[9]:
```
localOrders = [1,2,3]
```

[10]:
```
# plots of piecewise interpolants of (local) degree 1,2,3

fig, axs = plt.subplots(1, 3, figsize=(30,10), sharey=True)

Nrangeplot = [5,15,30,60]
for cnt_o,order in enumerate(localOrders):
    for n in Nrangeplot:
        fn = PiecewiseInterpolation(n, order, a, b, f, x)
        axs[cnt_o].plot(x, fn, ':', label=f"N={n}", linewidth=2)

for ax in axs:
    ax.plot(x, f(x), 'k', label="f(x)", linewidth=3)
    ax.legend(fontsize=20)
    ax.set_xlabel(r'$\mathbf{x}$', fontsize=20)
    ax.set_ylabel(r'$\mathbf{f(x)}$', fontsize=20)
    ax.grid(visible=True, which='major', color='#666666', linestyle='-')
```

```
    ax.minorticks_on()
    ax.grid(visible=True, which='minor', color='#999999', linestyle='-',alpha=0.
 ↪2)

plt.show()
```



[11]:
```
# absolute error trends of piecewise interpolants of (local) degree 1,2,3

Nrange = np.arange(5,51,5)
errorPwi = np.zeros((len(localOrders), len(Nrange)))
errorPwiRel = np.zeros((len(localOrders), len(Nrange)))
for cnt_o,order in enumerate(localOrders):
    for cnt_n,n in enumerate(Nrange):
        fn = PiecewiseInterpolation(n, order, a, b, f, x)
        errorPwi[cnt_o, cnt_n] = np.max(np.abs(fn-f(x)))
        errorPwiRel[cnt_o, cnt_n] = np.max(np.abs(fn-f(x)) / np.abs(f(x)))

plt.figure(figsize=(10,6))
for cnt_o,order in enumerate(localOrders):
    plt.semilogy(Nrange, errorPwi[cnt_o], ':o', label=f"order {order}")
plt.xlabel('Nombre de sous-intervalles', fontsize=20, fontweight='bold')
plt.ylabel('Error', fontsize=20, fontweight='bold')

plt.grid(visible=True, which='major', color='#666666', linestyle='-')
plt.minorticks_on()
plt.grid(visible=True, which='minor', color='#999999', linestyle='-',alpha=0.2)
plt.legend(fontsize=15)
plt.ylim([1e-4, None])
plt.show()
```



1.3.1 Commentaire

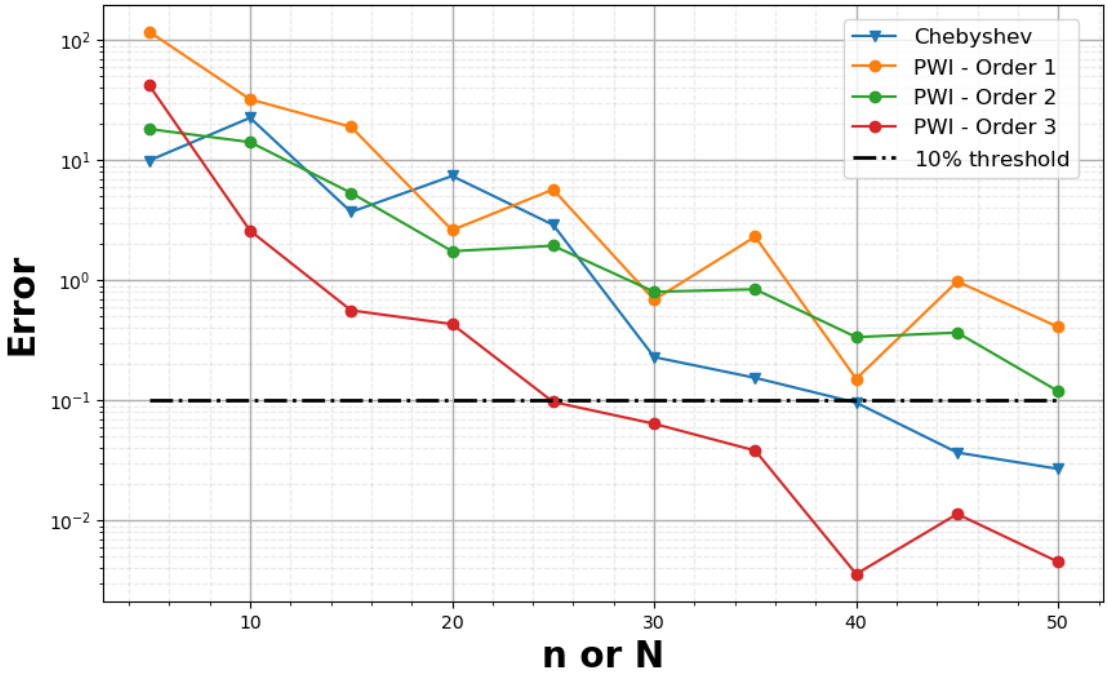
Réponse 3 *Un autre moyen efficace d'atténuer les effets du phénomène de Runge est de recourir à une interpolation par morceaux (ou par intervalles) de petit degré.* Dans un tel cas, en effet, seulement des dérivées de petit ordre sont impliquées dans l'expression de la limite supérieure de l'erreur d'interpolation, ce qui permet de la contrôler beaucoup mieux. Dans ce cas, pour $N = 50$, nous obtenons une erreur absolue $\approx 3 \cdot 10^{-2}$ en utilisant une interpolation linéaire par morceaux, $\approx 2 \cdot 10^{-3}$ avec une interpolation de degre 2 par morceaux et $\approx 2 \cdot 10^{-4}$ avec une interpolation de degre 3 par morceaux.

1.4 Partie 4

Finalement, parmi les méthodes analysées, laquelle nous permet d'interpoler la fonction donnée avec une **erreur relative** au plus de 10%, en minimisant le nombre de points où il faut évaluer la fonction f? **En regardant simplement la figure**, justifiez votre réponse. (*Réponse 4*)

[12]:
```
plt.figure(figsize=(10,6))

#plt.plot(Nrange, errorLagRel,  label = 'Lagrange')
plt.plot(Nrange, errorCheRel, '-v',  label='Chebyshev')
for cnt_o, order in enumerate(localOrders):
    plt.plot(Nrange, errorPwiRel[cnt_o], '-o',  label=f'PWI - Order {order}')

tol = 0.1
```

```
plt.plot(Nrange, tol*np.ones(len(Nrange)), 'k-.', label='$10 \%$ threshold',␣
 ↪linewidth=2)

plt.yscale('log')
plt.xlabel('n or N', fontsize=20, fontweight='bold')
plt.ylabel('Error', fontsize=20, fontweight='bold')
plt.grid(which='major', linestyle='-', linewidth=1)
plt.minorticks_on()
plt.grid(which='minor', color='#999999', linestyle='--',alpha=0.2)
plt.legend(fontsize=12, bbox_to_anchor=(0.975, 0.975), borderaxespad=0)
plt.show()
```



1.4.1 Commentaire

Réponse 4 Sur la base des resultats obtenus, une erreur relative plus petite de 10% peut être obtenue en utilisant:

- interpolation de Chebyshev, avec au moins $n = 40 \quad \Longrightarrow \quad N_f = n + 1 = 41$ évaluations de f.
- interpolation cubique par morceaux, avec au moins $N = 25 \quad \Longrightarrow \quad N_f = N + 1 + 2N = 76$ évaluations de f.

Du coup, parmi les méthodes analysées, *l'interpolation de Chebyshev de degré* $n = 40$ *est la methode plus efficace pour reduire l'erreur d'interpolation relative jusqu'a la tolerance souhaitée de* 10%, *en minimisant le nombre des points ou il faut évaluer la fonction* f.

2 Quelques petites questions finales (pas évaluées)

- What types of collaboration strategies did your group use?
  - Work in pairs on different sections.
  - Work individually on different sections.
  - Work together on the same section with one notebook opened.
  - Work together on the same section with multiple notebooks opened.
  - Other (please specify).
- How effective was your collaboration strategy today? Please rate from 1 (not at all) to 5 (very effective).
- How supported did you feel by your TA during the session today? Please rate from 1 (not at all) to 5 (very effective).

Please report your answers here. **Thank you!**