CONTRÔLE CONTINU

Méthodes numériques. Durée 1h30

Pour chacun des algorithmes à commenter, on donnera en entête l'objectif de chacun des algorithmes. On précisera aussi au cœur du programme toutes les subtilités que l'on jugera nécessaire de mettre en évidence.

Exercice 1

1. Commenter l'algorithme 1 donné en annexe 1. On remplacera en particulier le message d'erreur `"Message d'erreur 1"` par un message plus explicite.
2. Commenter l'algorithme 2 donné en annexe 2. On remplacera en particulier les messages d'erreur `"Message d'erreur 2"` et `"Message d'erreur 3"` par des messages plus explicites.

⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆

Exercice 2 Certaines méthodes de quadrature sont basées sur la propriété suivante : pour un degré p fixé, il existe $p+1$ coefficients $\omega_0^{(p)}, \ldots, \omega_p^{(p)}$, indépendant de l'intervalle $[a, b]$ tels que pour tout polynôme $P(x)$ de degré p, on ait

$$\int_a^b P(x)dx = (b-a)\sum_{k=0}^{p} \omega_k^{(p)} P(x_k)$$

où les x_k sont les $p+1$ points du segment $[a, b]$ découpant ce dernier en p intervalles de même longueur. Il est alors possible de déterminer, pour $p \in \mathbb{N}^*$ fixé, les coefficients $\omega_k^{(p)}$ en se basant sur les fonctions puissances, intégrées sur l'intervalle $[-1, 1]$.

1. *Cas* $p = 2$
   (a) Déterminer un système d'équations linéaires (S) vérifié par les coefficients $\omega_0^{(2)}$, $\omega_1^{(2)}$ et $\omega_2^{(2)}$ en s'appuyant sur les égalités suivantes :
   $$\forall i = 0,\ 1,\ 2, \quad \int_{-1}^{1} P_i(x)dx = 2\left(\omega_0^{(2)}.P_i(-1) + \omega_1^{(2)}.P_i(0) + \omega_2^{(2)}.P_i(1)\right)$$
   où
   $$P_0\ :\ x \mapsto 1, \quad P_1\ :\ x \mapsto x, \quad P_2\ x \mapsto x^2$$
   On précisera la matrice A et le second membre B de ce système linéaire.
   (b) Résoudre le système et donner les poids relatifs à la méthode de Simpson.
   (c) Montrer que, pour tout polynôme $P(x) = ax^2 + bx + c$ de degré 2, on a
   $$\int_{-1}^{1} P(x)dx = 2\left(\frac{1}{6}.P(-1) + \frac{2}{3}.P(0) + \frac{1}{6}.P(1)\right)$$
   (d) À l'aide du changement de variable
   $$t = \frac{h}{2}x + a + \frac{h}{2}$$
   montrer que pour tout polynôme $P(x) = ax^2 + bx + c$ et pour tout intervalle de la forme $[a, a+h]$, on a
   $$\int_a^{a+h} P(t)dt = h\left(\frac{1}{6}.P(a) + \frac{2}{3}.P\left(a + \frac{h}{2}\right) + \frac{1}{6}.P(a+h)\right)$$
2. *Cas général* : commenter l'algorithme 3 donné en annexe 3.
3. Expliquer en quoi ces méthodes permettent de calculer une valeur approchée de l'intégrale $I = \displaystyle\int_a^b f(x)dx$ à l'aide de polynômes interpolateurs.
4. Commenter les algorithmes 4.1 et 4.2 donnés en annexe 4.

⋆ ⋆
⋆

Nom : .... Prénom : ....

ANNEXE 1

Algorithme 1

```
    # ...
    # ...
    # ...
1.  def Fonction1(X,Y):

2.      p=len(X)-1                                     # ...
3.      if p <> len(Y)-1:
4.          return "Message d'erreur 1"                # ...

5.      P(x)=0
6.      for i in range(p+1):                           # ...
7.          Li(x)=1
8.          for j in range(p+1):
9.              if i<>j:
10.                 Li(x) = Li(x)*(x-X[j])/(X[i]-X[j])
            P(x) = P(x) + Y[i]*Li(x)                   # ...

11.     return P
```

Nom : .... Prénom : ....

ANNEXE 2

Algorithme 2

```
# ...
# ...
# ...
def Fonction2(X,Y,a,p):

    if p>len(X)-1:
        return "Message d'erreur 2"          # ...

    if a<X[0] or a>X[-1]:
        return "Message d'erreur 3"          # ...

    k=0
    while X[k]<a:                            # ...
        k += 1
    if k>=p:
        P=Fonction1(X[k-p:k+1],Y[k-p:k+1])   # ...
    else:
        P=Fonction1(X[0:p+1],Y[0:p+1])       # ...

    return P(a)                              # ...
```

Nom : ....   Prénom : ....

ANNEXE 3

Algorithme 3

```
# ...
# ...
# ...
def Fonction3(p):

    # ...
    X=srange(-1,1+2/p,2/p)

    # ...
    Lignes_A = []
    for i in range(p+1):                                   # ...
        Ligne_i = []                                       # ...
        for j in range(p+1):                               # ...
            Ligne_i.append(X[j]**i)                        # ...
        Lignes_A.append(Ligne_i)                           # ...
    A = matrix(Lignes_A)                                   # ...

    # ...
    Ligne_B=[]
    for k in range(p+1):                                   # ...
        Ligne_B.append((1/2)*(1/(k+1))*(1-(-1)**(k+1)))    # ...
    B = vector(Ligne_B)                                    # ...

    Wp = A**(-1)*B                                         # ...
    return Wp
```

Nom : .... Prénom : ....

ANNEXE 4

Algorithme 4.1

```
# ...
# ...
# ...
def Fonction41(f,a,b,Wp):

    p=len(Wp)-1                             # ...
    S=0
    h=(b-a)/p                               # ...
    for i in range(p+1):
        xi=a+i*h                            # ...
        S+=Wp[i]*f(xi)                      # ...
    S=S*(b-a)

    return S
```

Algorithme 4.2

```
# ...
# ...
# ...
def Fonction42(f,a,b,n,p):

    Wp=Fontion3(p)                          # ...
    h=(b-a)/n
    X=srange(a,b+h,h)                       # ...
    Ip=0
    for k in range(n):
        Ip += Fonction41(f,X[k],X[k+1],Wp)  # ...

    return Ip
```

CORRECTION

Exercice 1 :

1. c.f. annexe 1 ci-dessous.
2. c.f. annexe 2 ci-dessous.

⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆

Exercice 2 :

1. (a) — Pour $i = 0$,

$$\int_{-1}^{1} P_0(x)dx = \int_{-1}^{1} dx = 2$$

et

$$2(\omega_0^{(2)}.P_0(-1) + \omega_1^{(2)}.P_0(0) + \omega_2^{(2)}.P_0(1)) = 2(\omega_0^{(2)} + \omega_1^{(2)} + \omega_2^{(2)})$$

D'où

$$\boxed{(E_0)\ :\ \omega_0^{(2)} + \omega_1^{(2)} + \omega_2^{(2)} = 1}$$

— Pour $i = 1$,

$$\int_{-1}^{1} P_1(x)dx = \int_{-1}^{1} xdx = 0$$

et

$$2(\omega_0^{(2)}.P_1(-1) + \omega_1^{(2)}.P_1(0) + \omega_2^{(2)}.P_1(1)) = 2(\omega_0^{(2)}.(-1) + \omega_1^{(2)}.0 + \omega_2^{(2)}.1)$$

D'où

$$\boxed{(E_1)\ :\ -\omega_0^{(2)} + \omega_2^{(2)} = 0}$$

— Pour $i = 2$,

$$\int_{-1}^{1} P_2(x)dx = \int_{-1}^{1} x^2dx = \frac{2}{3}$$

et

$$2(\omega_0^{(2)}.P_2(-1) + \omega_1^{(2)}.P_2(0) + \omega_2^{(2)}.P_2(1)) = 2(\omega_0^{(2)}.(-1)^2 + \omega_1^{(2)}.0^2 + \omega_2^{(2)}.1^2)$$

D'où

$$\boxed{(E_2)\ :\ \omega_0^{(2)} + \omega_2^{(2)} = \frac{1}{3}}$$

Les coefficients cherchés sont forment donc la solution du système

$$(S)\ :\ \left\{\begin{array}{rcccccl} \omega_0^{(2)} & + & \omega_1^{(2)} & + & \omega_2^{(2)} & = & 1 \\ -\omega_0^{(2)} & & & + & \omega_2^{(2)} & = & 0 \\ \omega_0^{(2)} & & & + & \omega_2^{(2)} & = & \frac{1}{3} \end{array}\right. \Leftrightarrow AX = B$$

avec

$$A = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 \\ 0 \\ \frac{1}{3} \end{pmatrix} \qquad \text{et} \qquad X = \begin{pmatrix} \omega_0^{(2)} \\ \omega_1^{(2)} \\ \omega_2^{(2)} \end{pmatrix}$$

(b)

$$(S) \iff \left(\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 0 \\ 1 & 0 & 1 & \frac{1}{3} \end{array}\right)$$

$$\underset{\substack{L_1 \leftarrow L_1 - L_3 \\ L_2 \leftarrow L_2 + L_3}}{\iff} \left(\begin{array}{ccc|c} 0 & 1 & 0 & \frac{2}{3} \\ 0 & 0 & 2 & \frac{1}{3} \\ 1 & 0 & 1 & \frac{1}{3} \end{array}\right)$$

$$\iff \begin{cases} \omega_1^{(2)} = \frac{2}{3} \\ 2\omega_2^{(2)} = \frac{1}{3} \\ \omega_0^{(2)} + \omega_2^{(2)} = \frac{1}{3} \end{cases}$$

$$\iff \begin{cases} \omega_1^{(2)} = \frac{2}{3} \\ 2\omega_2^{(2)} = \frac{1}{3} \\ \omega_0^{(2)} + \omega_2^{(2)} = \frac{1}{3} \end{cases}$$

$$\iff \begin{cases} \omega_1^{(2)} = \frac{2}{3} \\ \omega_2^{(2)} = \frac{1}{6} \\ \omega_0^{(2)} = \frac{1}{3} - \frac{1}{6} = \frac{1}{6} \end{cases}$$

On retrouve ici les poids de la méthode de Simpson :

$$\omega_0^{(2)} = \omega_2^{(2)} = \frac{1}{6} \quad \text{et} \quad \omega_1^{(2)} = \frac{2}{3}$$

(c) Soit $P(x) = ax^2 + bx + c$ un polynôme de degré 2. On a

$$\begin{aligned} \int_{-1}^{1} P(x) = \int_{-1}^{1} (ax^2 + bx + c)dx \\ &= a \int_{-1}^{1} P_2(x)dx + b \int_{-1}^{1} P_1(x)dx + c \int_{-1}^{1} P_0(x)dx \\ &= a.2\left(\omega_0^{(2)}.P_2(-1) + \omega_1^{(2)}.P_2(0) + \omega_2^{(2)}.P_2(1)\right) \\ &\quad + b.2\left(\omega_0^{(2)}.P_1(-1) + \omega_1^{(2)}.P_1(0) + \omega_2^{(2)}.P_1(1)\right) \\ &\quad + c.2\left(\omega_0^{(2)}.P_0(-1) + \omega_1^{(2)}.P_0(0) + \omega_2^{(2)}.P_0(1)\right) \\ &= 2.\left(\omega_0^{(2)}.(aP_2(-1) + bP_1(-1) + cP_0(-1))\right. \\ &\quad + \omega_1^{(2)}.(aP_2(0) + bP_1(0) + cP_0(0)) \\ &\quad \left. + \omega_2^{(2)}.(aP_2(1) + bP_1(1) + cP_0(1))\right) \\ &= 2.\left(\omega_0^{(2)} P(-1) + \omega_1^{(2)} P(0) + \omega_2^{(2)} P(1)\right) \end{aligned}$$

(d) En posant $t = \frac{h}{2}x + a + \frac{h}{2}$, on a $dt = \frac{h}{2}$. De plus, pour $t = a$, on a $x = -1$ et pour $t = a + h$, on a $x = 1$. D'où

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

Or l'expression $\frac{h}{2}x + a + \frac{h}{2}$ étant linéaire en x, le polynôme $P\left(\frac{h}{2}x + a + \frac{h}{2}\right)$ est un polynôme de degré 2 en x. D'après la question précédente, on a donc

$$\begin{aligned} \int_a^{a+h} P(t)dt &= \frac{h}{2}.2\left(\omega_0^{(2)} P\left(\frac{h}{2}(-1) + a + \frac{h}{2}\right) + \omega_1^{(2)} P\left(\frac{h}{2}.0 + a + \frac{h}{2}\right) + \omega_2^{(2)} P\left(\frac{h}{2}.1 + a + \frac{h}{2}\right)\right) \\ &= \frac{h}{2}.2\left(\omega_0^{(2)} P(a) + \omega_1^{(2)} P\left(a + \frac{h}{2}\right) + \omega_2^{(2)} P(a+h)\right) \end{aligned}$$

2. c.f. annexe 3 ci-dessous.

3. Pour obtenir une valeur approchée de l'intégrale $I = \displaystyle\int_a^b f(x)dx$, on peut interpoler la fonction f par un polynôme ou une famille de polynômes coïncidant avec f en une famille de points équi-répartis sur l'intervalle $[a, b]$. L'intégrale de chacun de ces polynômes dépend alors des poids $\omega_i^{(p)}$ associés au degré p choisi ainsi que des valeurs des polynômes aux points d'interpolation. Or ces valeurs sont, par définitions, les valeurs de f. On peut donc intégrer les polynômes interpolateurs en se basant uniquement sur les valeurs de la fonction f.

4. c.f. annexe 4 ci-dessous.

⋆ ⋆
⋆

ANNEXE 1

Algorithme 1

```
# La fonction Fonction1 renvoie l'unique polynôme P
# interpolateur de degré p dont la courbe passe par
# les p+1 points définis par les abscisses de la
# liste X et les ordonnées de la liste Y.
# On pourrait la renommer Interpolation(X,Y)

def Fonction1(X,Y):

    p=len(X)-1                                        # p contient le degré du polynôme
    if p <> len(Y)-1:
        return "Erreur : taille des listes"           # la fonction s'arrête si les deux listes n'ont pas la même taille

    P(x)=0
    for i in range(p+1):                              # Pour chaque i, on construire le polynôme de Lagrange Li(x)
        Li(x)=1                                       # associé à la liste des abscisses
        for j in range(p+1):
            if i<>j:
                Li(x) = Li(x)*(x-X[j])/(X[i]-X[j])
        P(x) = P(x) + Y[i]*Li(x)                      # On construit le polynôme P(x) en tenant compte des ordonnées de la liste Y

    return P
```

ANNEXE 2

Algorithme 2

```
# La fonction Fonction2 renvoie une valeur approchée de
# f(a), calculée par interpolation par morceaux par des
# polynômes de degré p fixé par l'utilisateur.
# La fonction f est ici connue sous la forme d'un nuage
# de point défini par les listes X et Y et
# le polynôme interpolateur est calculé
# à l'aide de la fonction Fonction1.
# Cette fonction peut être nommée Valeur_interpolee(X,Y,a,p).

def Fonction2(X,Y,a,p):

    if p>len(X)-1:                                          # Il faut au moins p+1 points pour interpoler le nuage
        return "Trop peu de points pour le degré demandé"   # à l'aide de polynômes de degré p

    if a<X[0] or a>X[-1]:                                   # L'abscisse a doit être dans l'intervalle de mesure
        return a," n'est pas dans l'itervalle de mesures"

    k=0
    while X[k]<a:                                           # On cherche ici le premier point de mesure X[k]
        k += 1                                              # dont l'abscisse est supérieure à a

    if k>=p:                                                # Si cette abscisse est au delà de la p-ième,
        P=Fonction1(X[k-p:k+1],Y[k-p:k+1])                  # on prend les p points précédents et X[k] pour interpoler.
    else:                                                   # Sinon, on prend les p+1 premiers points.
        P=Fonction1(X[0:p+1],Y[0:p+1])

    return P(a)                                             # On renvoie la valeur du polynôme au point a
```

ANNEXE 3

Algorithme 3

```
# La fonction Fonction3 calcule les p+1 poids permettant
# de calculer l'intégrale de n'importe quel polynôme de
# degré p à l'aide de p+1 valeur régulièrement réparties
# sur l'intervalle d'intégration.
# Cette fonction peut être nommée Poids_quadrature(X,Y,a,p).

def Fonction3(p):

    # On découpe l'intervalle [-1,1] en p intervalles de même longeur
    X=srange(-1,1+2/p,2/p)

    # On construit la matrice A du système
    # linéaire donnant les poids cherchés.
    Lignes_A = []
    for i in range(p+1):                            # Chaque ligne est associée à
        Ligne_i = []                                # une fonction puissance différente.
        for j in range(p+1):                        # et contient les puissances i-èmes
            Ligne_i.append(X[j]**i)                 # des abscisses de la liste X
        Lignes_A.append(Ligne_i)
    A = matrix(Lignes_A)

    # On construit le vecteur B, second membre du
    # système linéaire donnant les poids cherchés.
    Ligne_B=[]
    for k in range(p+1):                            # Les coefficients de B sont les
        Ligne_B.append((1/2)*(1/(k+1))*(1-(-1)**(k+1)))  # intégrales des fonctions puissances
    B = vector(Ligne_B)                             # sur l'intervalle [-1,1].

    Wp = A**(-1)*B                                  # On renvoie la liste des poids sous la forme d'un vecteur.
    return Wp
```

ANNEXE 4

Algorithme 4.1

```
# La fonction Fonction41 est la fonction Newton_Cotes vue en TD. Elle approche l'intégrale de f sur [a,b] a l'aide
# d'un polynôme de degré p. ATTENTION : le degré p est donné calculé ici à partir de la liste des poids donnée en argument.

def Fonction41(f,a,b,Wp):

    p=len(Wp)-1                          # Le degré du polynôme est le nombre de poids moins 1.
    S=0
    h=(b-a)/p                            # Définition du pas d'interpolation.
    for i in range(p+1):
        xi=a+i*h                         # On définit le point d'interpolation.
        S+=Wp[i]*f(xi)                   # On ajoute sa contribution à l'intégrale totale.
    S=S*(b-a)

    return S
```

Algorithme 4.2

```
# La fonction Fonction42 est la fonction Newton_Cotes_compose vue en TD. Elle approche l'intégrale de f sur [a,b] a l'aide
# d'une famille de polynômes de degré p déterminés à l'aide de la fonction précédente.

def Fonction42(f,a,b,n,p):

    Wp=Fontion3(p)                       # Les poids sont calculés à partir de la fonction Fonction3
    h=(b-a)/n
    X=srange(a,b+h,h)                    # Découpage de l'intervalle [a,b] en n intervalles de même longueur h
    Ip=0
    for k in range(n):                   # La valeur approchée Ip est obtenue par n approximations
        Ip += Fonction41(f,X[k],X[k+1],Wp)  # calculées à l'aide de la fonction Fonction41.

    return Ip
```