© *Jean-Baptiste APOUNG KAMGA <jean-baptiste.apoung@math.u-psud.fr>*

Exercices d'entraînement : Eléments de réponse

Thème - 1 *Interpolation polynomiale : Apprentissage du cours*

Exercice-1-1 : Familiarisation avec le résultat principal du cours

Soit f de classe C^{n+1} sur $]a,b[$ et P_n son polynôme d'interpolation de Lagrange aux points deux à deux distincts $x_0,\ldots,x_n$. On a l'estimation d'erreur suivante :

$$\begin{cases} \forall x\in[a,b], \exists t_x \in]a,b[\quad \text{tel que} \\ \mathcal{E}_n(x) = (x-x_0)\ldots(x-x_n)\dfrac{f^{(n+1)}(t_x)}{(n+1)!} \end{cases} \tag{1}$$

Ce résultat a été démontré en cours.

Exercice-1-2 : Calcul à la main des polynômes d'interpolation

On se place dans le cas particulier suivant :

$$\left|\begin{array}{l} f(x)=\frac{4}{x}, \\ a=1,\ b=4, \\ n=2, x_0=1, x_1=2, x_2=4 \end{array}\right. \tag{2}$$

Q-2-1 : Calcul de P_2 par utilisation de la base de Lagrange

On construit la base de Lagrange, i.e les 3 polynômes $l_i(x), i=0,1,2$ tels que $l_i(x_j)=\begin{cases} 1 & \text{si } j==i \\ 0 & \text{sinon} \end{cases}$

i	x_i	$y_i=f(x_i)$	$l_i(x)$
0	1	4	$\dfrac{(x-2)(x-4)}{(1-2)(1-4)}$
1	2	2	$\dfrac{(x-1)(x-4)}{(2-1)(2-4)}$
2	4	1	$\dfrac{(x-1)(x-2)}{(4-1)(4-2)}$

Et on définit $P_2(x)=y_0\,l_0(x)+y_1\,l_1(x)+y_2\,l_2(x)$. Et après calculs, $\boxed{P_2(x)=7-\frac{7}{2}x+\frac{1}{2}x^2}$

Calcul de P_2 par utilisation de la base de Newton
(**Nous n'avons pas fait cette interpolation en cours: voir donc Thème2 en fin de ce document pour rappels**)

La base de Newton est dans le cas présent : $w_0(x)=1, \quad w_1(x)=x-x_0, \quad w_2=(x-x_0)(x-x_1)$.

On détermine les coefficients du polynôme d'interpolation de Lagrange dans cette base par construction de la matrice (triangulaire inférieure) des différences divisées : $(d_{i,j})_{0\le j\le i\le 2}$, via la relation :

$$\begin{cases} d_{i,0} &= y_i, & 0\le i\le 2 \\ d_{i,j} &= \dfrac{d_{i,j-1}-d_{i-1,j-1}}{x_i-x_{i-j}}, & 1\le j\le 2, \text{et } j\le i\le 2 \end{cases}$$

Les éléments diagonaux sont alors les coefficients recherchés: $P_2(x)=d_{0,0}w_0(x)+d_{1,1}w_1(x)+d_{2,2}w_2(x)$.

On organise les calculs en introduisant la table dite des différences divisées :

i	x_i	$d_{i,0} := y_i$	$d_{i,1}$	$d_{i,2}$
0	1	4		
1	2	2	-2	
2	4	1	$-\frac{1}{2}$	$\frac{1}{2}$

Ainsi, $P_2(x) = 4 - 2(x-1) + \frac{1}{2}(x-1)(x-2)$, soit encore $\boxed{P_2(x) = 7 - \frac{7}{2}x + \frac{1}{2}x^2}$.

Q-2-2 : On déduit alors l'expression de $\mathcal{E}_2(x) = f(x) - P_2(x)$, à savoir $\boxed{\mathcal{E}_2(x) = \frac{4}{x} - 7 + \frac{7}{2}x - \frac{1}{2}x^2}$

Q-2-2-1 : Expression de t_x de la formule (1) pour $x \in [1,4]$.

Soit $x \in [1,4]$. La fonction f étant de classe C^3 sur $]1,4[$, par application du résultat de l'exercice 1, on a d'une part l'existence de $t_x \in]1,4[$ tel que $\mathcal{E}_2(x) = (x-1)(x-2)(x-4)\frac{f'''(t_x)}{6}$. Soit, puisque $f'''(x) = -\frac{24}{x^4}$,

$$\mathcal{E}_2(x) = -(x-1)(x-2)(x-4)\frac{4}{t_x^4} \tag{3}$$

Par ailleurs, on a aussi $\mathcal{E}_2(x) = \frac{4}{x} - 7 + \frac{7}{2}x - \frac{1}{2}x^2 = \frac{8-14x+7x^2-x^3}{2x}$. *On factorise alors le numérateur de cette dernière en remarquant que l'erreur d'interpolation s'annule aux points d'interpolation.* On obtient

$$\mathcal{E}_2(x) = -\frac{(x-1)(x-2)(x-4)}{2x}. \tag{4}$$

Et par comparaison de (3) et (4), on obtient l'équation $\frac{1}{2x} = \frac{4}{t_x^4}$ d'où l'on tire $\boxed{t_x = (8x)^{\frac{1}{4}}}$.

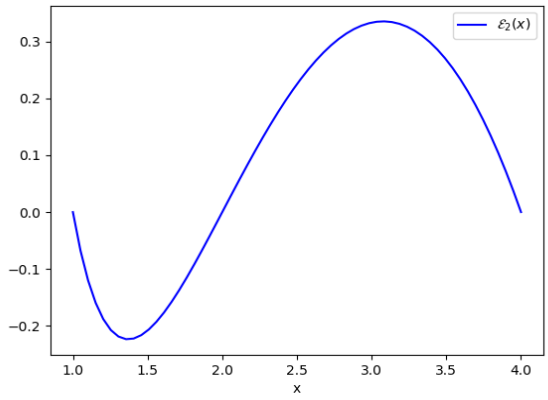


Figure 1: Erreur d'interpolation $\mathcal{E}_2(x)$

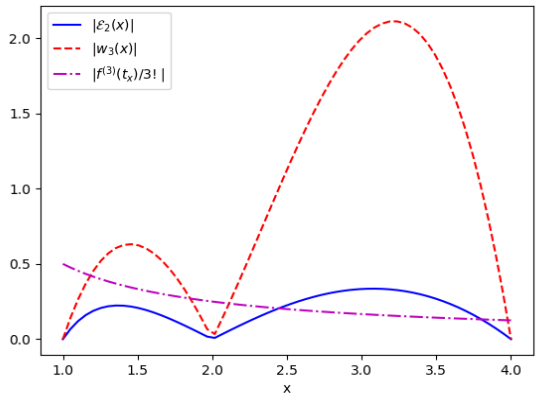


Figure 2: Apports de w_3 et $\frac{f^{(3)}(t_x)}{3!}$ dans l'erreur $\mathcal{E}_2(x)$

Q-2-2-2 : Quantification des apports de $(x-1)(x-2)(x-4)$ et $\frac{f^{(3)}(t_x)}{3!}$ dans l'erreur $\mathcal{E}_2(x)$.

On a $|\mathcal{E}_2(x)| = |w_3(x)|\left|\frac{4}{t_x^4}\right|$, où $w_3(x) = (x-1)(x-2)(x-4)$.

Or sur $[1,4]$, on a $\boxed{\max_{1\le x\le 4}|w_3(x)| \ge |w_3(3)| = 2 > \frac{1}{2} = \max_{1\le x\le 4}\frac{1}{2x} = \max_{1\le x\le 4}\frac{f^{(3)}(t_x)}{3!}}$.

Il apparaît pour ce problème particulier que la contribution de $|w_3(x)|$ dans l'erreur est bien plus importante. Ceci motive sur le fait que dans la théorie de l'interpolation, il faudra investir dans la réduction de la contribution de cette quantité, par exemple par un choix convenable des points x_i (voir cours sur phénomène de Runge)

Comme conseillé, on peut s'aider de graphique pour mieux s'en rendre compte: voir Figure 2

Q-2-3 : Montrons que $\mathcal{E}_2$ atteint sa valeur maximale en un point $\tilde{x}$ sur $[2,4]$.

La Figure 1 permet encore de fournir une réponse. ***Laissons le en exercice***.

Proposons ici une autre démonstration:
On sait que $\mathcal{E}_2$ est de classe C^2 sur $]1,4[$. Si elle atteint donc sa valeur maximale en un point x sur $]1,4[$ ce point annulera sa dérivée : $\mathcal{E}_2^{'}(x)=0$. Dressons donc le tableau de variation de $\mathcal{E}_2^{'}(x)$ afin de conclure.

En effet, nous avons $\mathcal{E}_2^{'}(x)=\frac{7}{2}-x-\frac{4}{x^2}$ et $\mathcal{E}_2^{''}(x)=\frac{8-x^3}{x^3}$.

x	1		$\tilde{x}_1$		2		$\tilde{x}_2$		4
$\mathcal{E}_2^{''}(x)$		$+$	$\|$	$+$	0	$-$	$\|$	$-$	
$\mathcal{E}_2^{'}$	$-\frac{3}{2}$	$\nearrow$	0	$\nearrow$	$\frac{1}{2}$	$\searrow$	0	$\searrow$	$-\frac{3}{4}$
$\mathcal{E}_2^{'}(x)$		$-$	0	$+$	$\|$	$+$	0	$-$	
$\mathcal{E}_2$	0	$\searrow$	$\mathcal{E}_2(\tilde{x}_1)$	$\nearrow$	0	$\nearrow$	$\mathcal{E}_2(\tilde{x}_2)$	$\searrow$	0

Ainsi :

- $\mathcal{E}_2^{''}$ s'annule en $x=2$, est strictement positive sur $]1,2[$ et strictement négative sur $]2,4[$.
- Puisque $\mathcal{E}_2^{'}(1)=-\frac{3}{2}<0$, $\mathcal{E}_2^{'}(2)=\frac{1}{2}>0$, $\mathcal{E}_2^{'}(4)=-\frac{3}{4}<0$, $\mathcal{E}_2^{'}$ s'annule en exactement deux points $\tilde{x}_1\in]1,2[$ et $\tilde{x}_2\in]2,4[$.

Comme $\mathcal{E}_2^{''}(\tilde{x}_2)<0$ et $\mathcal{E}_2^{''}(\tilde{x}_1)>0$, c'est donc $\tilde{x}_2$ qui réalise le maximum de $\mathcal{E}_2$ sur $]1,4[$ pendant que $\tilde{x}_1$ réalise le minimum de $\mathcal{E}_2$ sur $]1,4[$.

Attention, même si ce n'est pas le but de la question, un travail supplémentaire est nécessaire pour identifier lequel de $|\mathcal{E}_2(\tilde{x}_1)|$ et $|\mathcal{E}_2(\tilde{x}_2)|$ correspond au maximum de $|\mathcal{E}_2|$ sur $[1,4]$. Ceci est visible graphiquement voir Figure 1. La preuve est laissée comme **exercice d'exploration**, au terme duquel on a bien $\boxed{\tilde{x}=\tilde{x}_2}$.

Q-2-4 : On a déjà montré ci-dessus que $\tilde{x}$ est racine $\mathcal{E}_2^{'}$ donc de $g(x)=\frac{7}{2}-x-\frac{4}{x^2}$.

Q-2-5 : De l'étude du sens de variation menée ci-dessus, il découle que g est strictement décroissante sur $[2,4]$. Elle est donc injective sur cet intervalle et réalise une bijection entre cet intervalle et son image : $[-\frac{3}{4},\frac{1}{2}]$.

Comme $0\in[-\frac{3}{4},\frac{1}{2}]$, il vient que $g^{-1}(0)$ existe et est unique. Or $g(\tilde{x})=0$ d'où $\boxed{\tilde{x}=g^{-1}(0)}$.

Exercice-1-3 : Exemple d'utilisation de l'interpolation polynomiale

On souhaite utiliser les polynômes d'interpolation de g^{-1} pour approcher $\tilde{x}$.

Q-3-1 : On pose $y_0=g(2)$, $y_1=g(4)$. Soit q_1 le polynôme d'interpolation aux points $(y_0,2),(y_1,4)$. Il est donnée par (**on utilise ici la base de Lagrange, voir la question Q-3-3 pour l'utilisation de la base de Newton**):

$$\boxed{q_1(y)=2\frac{y-y_1}{y_0-y_1}+4\frac{y-y_0}{y_1-y_0}=\frac{14-8y}{5}}$$

Q-3-2 : On évalue alors $\boxed{q_1(0) = \frac{14}{5}.}$ Montrons qu'on obtient ainsi une approximation de de $\tilde{x}$.

En effet, q_1 étant une approximation de g^{-1}, il en découle que $\frac{14}{5}$ est une approximation de $\tilde{x}$.

La réponse à la question s'arrête là. *Néanmoins on peut aller plus loin. Il est conseillé dans ce cas d'utiliser une approche graphique beaucoup plus abordable (voir deuxième colonne ci-dessous : Approche 2).*

Approche 1:

Puisque g est de classe C^2 avec $g'(x) \neq 0$ sur $]2,4[$, il vient que g^{-1} est continue sur $[-\frac{3}{4}, \frac{1}{2}]$ et de classe C^2 sur $]-\frac{3}{4}, \frac{1}{2}[$ avec $\boxed{(g^{-1})''(y) = -\frac{g''(x)}{(g'(x))^3} \quad \forall y \in]-\frac{3}{4}, \frac{1}{2}[\text{ tel que } y = g(x)}$.

Les conditions d'application du résultat de l'exercice 1 sont donc réunies: $\exists ! \, t_0 \in]-\frac{3}{4}, \frac{1}{2}[$ tel que

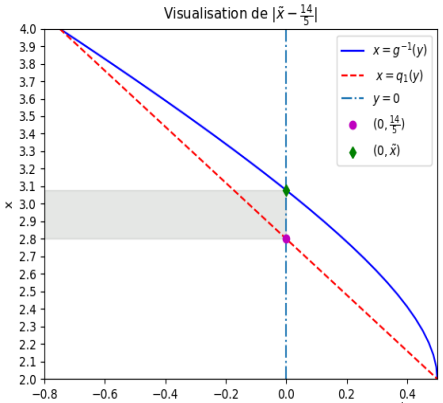
$$g^{-1}(0) - q_1(0) = (0 - \frac{1}{2})(0 + \frac{3}{4})\frac{(g^{-1})''(t_0)}{2}.$$

Soit encore $\tilde{x} - \frac{14}{5} = -\frac{3}{16}(g^{-1})''(t_0) = \frac{9}{2}\frac{a_0^5}{(a_0^3-8)^3}$ où $t_0 = g(a_0)$.

Ainsi, $\forall a \in]2,4[$ tel que $a \leq a_0$, on a : $|\tilde{x} - \frac{14}{5}| \leq \frac{9}{2}\frac{a^5}{(a^3-8)^3}$.

Il s'avère pour cet exemple qu'on peut prendre $a = \frac{14}{5}$. Dès lors, $|\tilde{x} - \frac{14}{5}| \leq \frac{94539375}{331527424} \leq 0.2852$.

Approche 2:



La hauteur de la partie hachurée majore $|\tilde{x} - \frac{14}{5}|$ i.e $|\tilde{x} - \frac{14}{5}| \leq 3.1 - 2.8 = 0.3$

Q-3-3 : Fort de ce qui précède on peut donc construire la suite $(x_n)_n$ d'approximation de $\tilde{x}$ par la formule

$$\begin{cases} x_0 = 2, x_1 = 4 \\ \forall n = 1, \ldots \\ \quad x_{n+1} = q_n(0), \text{ avec } q_n \text{ le polynôme interpolateur aux points } (g(x_i), x_i), i = 0, \ldots, n \end{cases} \tag{5}$$

Q-3-3-3 : Déterminons x_3, x_4.

L'organisation des calculs se fait comme illustrée ci-dessous :
un script Python décrivant encore mieux cette organisation est fourni, voir Listing1.

Note 1 (Organisation des calculs pour déterminer x_3)**.**

On commence par compléter la ligne correspondante de la table des différences divisées:

1) →

i	y_i	x_i	$d_{i,1}$
0	$\frac{1}{2}$	2	
1	$-\frac{3}{4}$	4	$-\frac{8}{5}$

On construit le polynôme d'interpolation $\boxed{q_1(y) = 2 - \frac{8}{5}(y - \frac{1}{2})}$ et on déduit $\boxed{x_2 = q_1(0) = \frac{14}{5}}$

On complète ensuite la ligne correspondant à $i = 2$:

2) →

i	y_i	x_i	$d_{i,1}$	$d_{i,2}$
0	$\frac{1}{2}$	2		
1	$-\frac{3}{4}$	4	$-\frac{8}{5}$	
2	$g(\frac{14}{5})$	$\frac{14}{5}$		

3) →

i	y_i	x_i	$d_{i,1}$	$d_{i,2}$
0	$\frac{1}{2}$	2		
1	$-\frac{3}{4}$	4	$-\frac{8}{5}$	
2	$\frac{93}{490}$	$\frac{14}{5}$	$-\frac{392}{307}$	$-\frac{6076}{5833}$

On peut alors calculer le polynôme d'interpolation $\boxed{q_2(y) = q_1(y) - \frac{6076}{5833}(y - \frac{1}{2})(y + \frac{3}{4})}$,

duquel on déduit $\boxed{x_3 = q_2(0) = x_2 - \frac{6076}{5833}(-\frac{1}{2})(\frac{3}{4}) = \frac{186109}{58330}}$

On peut compléter la ligne $i = 3$ en vue de la construction de x_4

4)− >

i	y_i	x_i	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$
0	$\frac{1}{2}$	2			
1	$-\frac{3}{4}$	4	$-\frac{8}{5}$		
2	$\frac{93}{490}$	$\frac{14}{5}$	$-\frac{392}{307}$	$-\frac{6076}{5833}$	
3	$g(\frac{186109}{58330})$	$\frac{186109}{58330}$			

5)− > **Suite laissée en exercice.**

Note 2 (Résultats attendus : utiliser le Listing 1).

`r = CalculSymboliqueDeZeros()`

- `r.x3` donnera $\frac{186109}{58330} = x_3$
- `r.x4` donnera $\frac{29624684255979638061082124747176943081}{9582856562032169254671164155409203670} = x_4$

Listing 1: code de calcul de x_3 et x_4

```
import sympy as sym

def CalculSymboliqueDeZeros():
    """
    Calcul a partir de x0, x1, des approximations x3, x4 du zero x de  g(x) = 7/2 - x - 4/x^2,
    par la methode d'interpolation inverse utilisant le polynome d'interpolation de Newton.
    voir Question Q.3.3.3 de la fiche  entrainement_1.pdf

    Cours Math209: Analyse et Simulations   Univ. Paris sud 2018-2019
    @copyright Jean-Baptiste APOUNG KAMGA

    Utilisation :
       r = CalculSymboliqueDeZeros()

       r sera une structure dont les donnees (accessibles via l'operateur '.' e.g r.x0) sont :
       1/ la fonction dont on cherche le zero  g
          ---------------------------------------
       2/ les elements de la table des differences divisees
          -------------------------------------------------
       y0 | x0
       y1 | x1  d11
       y2 | x2  d21  d22
       y3 | x3  d31  d32  d33
       y4 | x4  d41  d42  d43  d44
       3/ les polynomes
       -----------------
        q1(y),  q2(y),  q3(y)
    """
    class TypeResultat :
        """ type permettant de stocker le resultat de cette fonction """
        pass

    r = TypeResultat()  # variable locale stockant le resultat

    x, y = sym.symbols('x y')

    r.g = sym.Rational(7,2) - x -4/x**2

    """ Etape1: """
    """ construction de q1 """
    r.x0, r.y0 = 2, r.g.subs(x,2)
    r.x1, r.y1 = 4, r.g.subs(x,4)
    r.d11 = (r.x1 - r.x0)/(r.y1 - r.y0)
    r.q1 = r.x0 + r.d11 * (y - r.y0)
    """ édtermination de x2 et y2"""
    r.x2 = r.q1.subs(y,0)
    r.y2 = r.g.subs(x, r.x2)

    """ Etape 2: """
    """ construction de q2 """
    r.d21 = (r.x2 - r.x1)/(r.y2 - r.y1)
    r.d22 = (r.d21 - r.d11)/(r.y2 - r.y0)
    r.q2 = r.q1 + r.d22 * (y - r.y0) * (y - r.y1)
    """ édtermination de x3, y3 """
    r.x3 = r.q2.subs(y,0)
    r.y3 = r.g.subs(x, r.x3)

    """ Etape 3: """
    """ construction de q3 """
    r.d31 = (r.x3 - r.x2)/(r.y3 - r.y2)
    r.d32 = (r.d31 - r.d21)/(r.y3 - r.y1)
    r.d33 = (r.d32 - r.d22)/(r.y3 - r.y0)
    r.q3 = r.q2 + r.d33 * (y - r.y0) * (y - r.y1) * (y-r.y2)
    """ determination de x4, y4 """
    r.x4 = r.q3.subs(y,0)
    r.y4 = r.g.subs(x, r.x4)
    return r
```

Q-3-3-4 : Un calcul donne : $\boxed{|x_4 - x_3| = 0.09919712209696561}$, $\boxed{|g(x_4)| = 0.009970534393078786}$.
On remarque, du moins pour le problème présent, que la suite x_n varie un peu moins vite que $|g(x_n)|$: $\boxed{|g(x_3)| = 0.08354689712131365}$. Il semble par conséquent plus préférable de traquer $\tilde{x}$ en disant qu'elle est approximativement égale à x_n pour laquelle $|x_n - x_{n-1}| \leq \varepsilon$ plutôt qu'à celle pour laquelle $|g(x_n)| \leq \varepsilon$.

Q-3-4 : **Mise en oeuvre**.

Note 3 (Test d'arrêt).

Le test d'arrêt peut-être au choix : $1 : |g(x_n)| \leq \varepsilon_f$ ou $2 : |x_{n+1} - x_n| \leq \varepsilon_x$
Mais comme nous l'avons vu précédemment, le second choix est plus avantageux:

- Il est moins coûteux, car ne nécessite pas d'évaluation de la fonction.
- Il est rattaché à la résolution de la grille: en effet, si la représentation graphiquement de la courbe de la fonction avec maillage de pas Δx confère une résolution suffisante, alors on pourra prendre pour ε_x une fraction de Δx, par exemple $\varepsilon_x = \frac{\Delta x}{10}$.
- Par ailleurs voir Note4, $|x_{n+1} - x_n| = |d_{n,n}| \prod_{i=0}^{n-1} |g(x_i)|$, sachant qu'on cherche le zéro de g.

Q-3-4-5 : Pour la mise en oeuvre de cet algorithme, on fournit le Listing 2 ci-dessous

Listing 2: Recherche de zéro par l'interpolation inverse

```
def ZeroParInterpInverse (f, x0, x1, eps):
    """
    Recherche de x* tel que f(x*) = 0, par la methode de l'interpolation inverse
    Entrees :
       f      : la fontion dont on cherche le zero, supposee bijective dans l'intervalle considere
       x0, x1 : deux approximations distinctes initiales de la racine
       eps    : la tolerance necessaire pour le test d'arret
    Sorties :
       x : la suite generee des valeurs approchees de la racine (toute la suite pour une analyse)
       y : la suite generee des valeurs de f en chaque element de x

    @c Jean-Baptiste APOUNG KAMGA
    Cours Math209: Analyse et Simulations   Univ. Paris sud 2018-2019
    """
    xi = [x0,x1]
    yi = [f(x0), f(x1)]
    conv = False
    niter = 0
    while(not(conv)):
        xx = InterpNewton(yi, xi, 0) # optimisable voir Note $\ref{note:2}$
        yy = f(xx)
        xi.append(xx)
        yi.append(yy)
        #conv = np.abs(yy) < eps
        conv = np.abs(xi[-1] - xi[-2]) < eps
        niter += 1
    return xi, yi, niter
```

Note 4.

Dans le Listing 2, on fait appel à **`InterpNewton`**. Mais on peut réduire le coût des calculs en tirant profit :

- de la récurrence qu'offre les polynômes d'interpolation de Newton,
- du fait que ce polynôme n'est utilisé que pour déterminer sa valeur en 0.

C'est-à-dire :

$\boxed{q_n(y) = q_{n-1}(y) + d_{n,n} \prod_{i=0}^{n-1} (y - y_i)}$ et évalué en $y = 0$ donne : $\boxed{x_{n+1} = x_n + d_{n,n} \prod_{i=0}^{n-1} (-y_i)}$

Comme $y_n = g(x_n)$, la suite (y_n) tend vers 0. Ainsi $\prod_{i=0}^{n-1} (-y_i)$ est de plus en plus petit.
La mise en oeuvre de cette optimisation est laissée en exercice.

FIN DE LA CORRECTION

Note 5 (Objectif: construction et évaluation du polynôme d'interpolation de Lagrange)**.**

Soit donnés une fonction f de classe C^{n+1}, et $n+1$ réels $x_0, \ldots, x_n$ deux à deux distincts. On cherche l'unique polynôme P_n de degré n qui coincide avec f au points $x_i, i = 0, \ldots, n$. On est guidé dans cette recherche par deux objectifs:

- sa construction simple
- son évaluation efficace

Le principe de construction consiste à choisir une base $\mathcal{B} = \{\varphi_0, \varphi_1, \ldots, \varphi_n\}$ *de* $\mathbb{R}_n[X]$ *puis de déterminer les composantes de* P_n *dans cette base. On chercehe donc* $(a_i)_{i=0}^n$ *telsque*

$$P_n(X) = \sum_{i=0}^{n} a_i \varphi_i(X)$$

par résolution du système

$$\sum_{j=0}^{n} a_j \varphi_j(x_i) = f(x_i),\ i = 0, \ldots, n$$

Dans toute la suite on posera $y_i = f(x_i),\ i = 0, \ldots,\ n$, et A la matrice dont $A_{i,j} = \varphi_j(x_i), 0 \leq i, j \leq n$.

Nous avons vu en cours deux approches pour construire le polynôme d'interpolation de Lagrange.

1. L'approche utilisant la base canonique (ou base des monômes) de $\mathbb{R}_n[X]$: $\boxed{\mathcal{B} = \left\{1, X, X^2, \ldots, X^n\right\}}$.
   Nous avons appelé cette approche (par abus **méthode de Vandermonde**).
   - **Inconvénients**: La matrice de Vandermonde est pleine et *mal conditionnée* rendant la méthode instable lorsque le nombre n de points d'interpolation devient grand.
   - **Avantages**: On dispose d'un algorithme efficace pour évaluer le polynôme: algorithme de **Horner**.
   - **Script (vu en cours Groupe 5)**: voir Listing 8
2. Approche par utilisation de la base duale de l'application linéaire

   $$\Phi : \mathbb{R}_n[X] \ni P \mapsto (P(x_0), \ldots, P(x_n)) \in \mathbb{R}^{n+1}$$

   appélée aussi **base de Lagrange.** C'est-à-dire $\boxed{\mathcal{B} = \{L_0, L_1, \ldots, L_n\}}$ où L_i est l'unique élément de $\mathbb{R}_n[X]$ valant 1 en x_i et 0 en x_j tel que $j \neq i$:

   $$L_i(X) = \prod_{j=0, j\neq i}^{n} \frac{X - x_j}{x_i - x_j} \tag{6}$$

   - **Avantages**: La construction du polynôme est simple: $P_n(X) = \sum_{i=0}^{n} y_i L_i(X)$
   - **Inconvénients**: l'évaluation de cette expression peut être coûteux si elle n'est pas menée de manière optimale. L'une des évaluations optimales est la **formule barycentrique de Lagrange** :

   $$P_n(X) = \frac{\displaystyle\sum_{i=0}^{n} \frac{y_i}{(X - X_i)\, w'_{n+1}(x_i)}}{\displaystyle\sum_{i=0}^{n} \frac{1}{(X - X_i)\, w'_{n+1}(x_i)}}$$

où w_{n+1} est le polynôme de degré $n+1$ qui s'annule en tous les points d'interpolation :

$$w_{n+1}(X) = (X - x_0)(X - x_1)\dots(X - x_n).$$

Malheureusement l'évaluation de cette formule peut être difficile (on vous l'a donc épargnée en TP). Néanmoins elle offre un gain considérable lorsqu'elle est bien menée.

- **Script (vu en cours Groupe 5)**: voir Listing 9.

L'approche que nous décrivons à présent est intermédiaire entre les deux approches précédententes. Elle offre une efficacité d'évaluation comparable à celle de la base canonique et une simplicité de construction comparable à celle de la base de lagrange.

L'idée est de prendre la base adaptée de $\mathbb{R}_n[X]$ formée des polynômes suivants : $\boxed{\mathcal{B} = \{w_0, w_1 \dots, w_n\}}$ où

$$\begin{cases} w_0 &= 1 \\ w_1 &= (X - x_0) \\ w_2 &= (X - x_0)(X - x_1) \\ \vdots &\vdots &\vdots \\ w_n &= (X - x_0)(X - x_1)\dots(X - x_{n-1}) \end{cases}$$

Les composantes de P_n dans cette base s'obtiennent alors par construction de la matrice (triangulaire inférieure) des différences divisées. C'est-dire

$$\boxed{P_n = \sum_{i=0}^{n} d_{i,i}\, w_i} \tag{7}$$

où $(d_{i,j})_{0\le j\le i\le n}$, sont constuits via la relation :

$$\begin{cases} d_{i,0} &= y_i, & 0 \le i \le n \\ d_{i,j} &= \dfrac{d_{i,j-1} - d_{i-1,j-1}}{x_i - x_{i-j}}, & 1 \le j \le n, \text{et } j \le i \le n \end{cases} \tag{8}$$

- On remarquera que dans cette écriture, les entrées de la matrice (i.e les $d_{i,j}$) sont construites colonnes par colonnes et de la première à la dernière. *Et dans chaque colonne on calcule l'élément diagonal* **ce qui guide sur la longueur de l'intervalle nécessaire pour les dénominateurs des entrées pour cette colone** $d_{i,i} = \dfrac{d_{i,i-1} - d_{i-1,i-1}}{x_i - x_0}$ *puis on calcule les éléments de la colonne qui sont situés sous la diagonale* $d_{i,j} = \dfrac{d_{i,j-1} - d_{i-1,j-1}}{x_i - x_{i-j}}$.
- On remarquera aussi qu'aux termes de la construction, seuls les éléments diagonaux sont utilisés dans l'expression du polynome recherché.
- Enfin pour simplifier la construction on peut se servir d'un tableau, appelé **table des différences divisées**:

i	x_i	$d_{.,0} := y_i$	$d_{.,1}$	$d_{.,2}$	$\dots$	$d_{.,j-1}$	$d_{.,j}$	$\dots$
0	x_0	y_0						
1	x_1	y_1	$d_{1,1}$					
2	x_2	y_2	$d_{2,1}$	$d_{2,2}$				
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$			
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\ddots$		
j	x_j	y_j	$d_{j,1}$	$d_{j,2}$		$d_{j,j-1}$	$d_{j,j}$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\dots$	$\vdots$	$\vdots$	$\dots$
$i-1$	x_{i-1}	y_{i-1}	$d_{i-1,1}$	$d_{i-1,2}$	$\dots$	$d_{i-1,j-1}$	$d_{i-1,j}$	$\dots$
i	x_i	y_i	$d_{i,1}$	$d_{i,2}$	$\dots$	$d_{i,j-1}$	$d_{i,j}$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\dots$	$\vdots$	$\vdots$	

Résumons cela à travers des scripts en `Python`. Commençons d'abord par la constriction des coefficients de la matrice des différences divisées:

Listing 3: Matrice des différences divisées

```
def MatriceDifferencesDivisee(x,y) :
    # x et y doivent etre des numpy array
    n = len(y)
    d = np.zeros((n,n))
    d[:,0] = 1. * y                        # On construit la èpremire colonne de d
    for j in range(1,n):
        d[j:n,j] = (d[j:n, j-1] - d[j-1:n-1, j-1])/(x[j:n] - x[0:n-j])
        #d[j:,j] = (d[j:, j-1] - d[j-1:-1, j-1])/(x[j:] - x[:n-j])  # on peut aussi ecrire comme ceci
    return d
```

Avec cette matrice on peut calculer les coefficients du polynôme.

Listing 4: Coefficients du polynôme pas différences divisées version non optimisée

```
def DifferencesDivisees(x, y) :
    # x et y doivent etre des numpy array
    # Attention Cet algorithme n'est pas optimisee
    d = MatriceDifferencesDivisee(x,y)  # calcul de la matrice des differences divisee
    a = np.diag(d)                      # extraction des coefficients du polynome
    return a
```

Mais puisque seuls les éléments diagonaux sont nécessaires, on peut se demander si l'on pouvait s'épargner le stockage des éléments extra-diagonaux. La réponse par l'affirmative est donnée par :

Listing 5: Coefficients du polynôme pas différences divisées

```
def DifferencesDivisees(x, y) :
    # x et y doivent etre des numpy array
    n = len(y)
    d = y.copy()
    for j in range(1,n):
        d[j:] = (d[j:] - d[j-1:-1])/(x[j:] - x[:n-j])
    return d
```

Une fois les coefficients disponibles on peut calculer le polynome d'interpolation, par une version adaptée de l'algorithme de Horner, que nous appellerons `HornerNewton`

Proposition 0.0.1 (Evaluation efficace du polynôme d'interpolation de Newton)**.**

Soit x un réel donné, $x_0, \ldots, x_n$ des réels deux à deux disticntes. Soit le Q polynôme défini par :

$$Q(x) = a_0 + \sum_{i=1}^{n} a_i \prod_{j=0}^{i-1} (x - x_j) = \sum_{i=0}^{n} a_i \, w_i(x)$$

la suite des polynômes $Q_0, Q_1, \ldots, Q_n$ définie par :

$$\left\{ \begin{array}{rcll} Q_n & = & a_n & \\ Q_k & = & a_k + (x - x_k)\, Q_{k+1} & \text{pour } k = n-1, \ldots, 0 \end{array} \right.$$

vérifie : $Q_0 = Q$.

Sous forme de script `Python`, cela donnerait :

Listing 6: Algorithme de Horner ajusté

```
def HornerNewton(d, x, xx):
    """
    On evalue  efficacement yy = P(xx) avec
    P(xx) = d[n-1]*(xx-x[n-1])..(xx - x[0]) + ... + d[1](xx - x[0]) + d[0]
    """
    # x et y doivent etre des numpy array
    n = len(d)
    yy = 0. * xx + d[n-1]   # ceci force yy a avoir la dimension de xx avec toutes ses entrees a d[n-1]
    for i in range(n-2, -1, -1): # en Python la borne droite n'est pas atteinte donc i s'arrete a 0
        yy = d[i] + (xx - x[i]) * yy
    return yy
```

On peut donc fournir le script ci-dessous pour l'interpolation polynomiale:

Listing 7: Interpolation de Newton

```
def InterpNewton(x, y, xx):
    x = np.asanyarray(x, dtype='float')     # force x a etre un numpy array
    y = np.asanyarray(y, dtype='float')     # idem
    xx = np.asanyarray(xx, dtype='float')   # idem
    d = DifferencesDivisees(x, y)
    return HornerNewton(d, x, xx)
```

Note 6 (Commentaires).

- Les scripts ci-dessous sont donnés à titre indicatif (avec en conséquence beaucoup moins de commentaires). Ils ne sont fournis ici que parceque déjà donnés en cours (dans le Groupe 5).
- Une correction du TP4 comportant des variantes mieux commentées est déjà disponible sur Dokéos. Vous pouvez donc vous y reporter.

Listing 8: Interpolation de Lagrange: schéma de Vandermonde

```
def MatriceInterpVdM(xi):
    # xi doit etre un numpy array
    n = len(xi)
    A = np.ones ((n, n))
    for k in range (1,n):
        A[:, k] = A[:, k -1] * xi
    return A

def CoefInterpVdM(AvdM,yi):
    # AvdM  : matrice de Vandermonde
    # yi : doit être numpy array
    p = np.linalg.solve (AvdM, yi)
    return p

def Horner(p,x):
    """ On suppose que le polynome s'ecrit P(x) = p[0] x^n + ...+ p[i] x^(n-i) + ... + p[n-1]x + p[n]↩
        """
    """ Des lors P(x) = (...(((p[0] x + p[1]) x + p[2])x + p[3])x + ...)x + p[n]) """
    x = np.asanyarray(x, dtype='float')
    # approche 1
    n = len(p)
    y = p[0]
    for i in range(n-1):
        y = y * x + p[i+1]
    return y
    """
    # approche 2
    y = 0 * x
    for ai in p:
        y *= x
        y += ai
    return y
    """

def InterpVdM (xi, yi, x ):
    """
    Interpolation par la methode de Vandermonde
    """
    xi, yi, x = np. asanyarray (xi, dtype = 'float'),\
    np.asanyarray (yi, dtype = 'float'), \
    np.asanyarray (x , dtype ='float')
    n = xi.size
    if yi.size != n:
        print ('Error  in  InterpVdm : x and y do not  have  the  same   size ')

    A = MatriceInterpVdM(xi)
    p = CoefInterpVdM(A,yi)
    y = Horner(p[::-1] , x)  # Ceci parce que p est obtenu pselon l'order decroissant des monomes.
                             # on l'ordonne donc dans le sens decroissant des monomes
    return y
```

Listing 9: Interpolation de Lagrange : schéma de Lagrange approche non optimisée

```
def k_baseLagrange(k,xi,yi,x):
    ''' k-eme element de la base de Lagrange '''
    x = np.asanyarray(x,dtype='float')
    xi = np.asanyarray(xi,dtype='float')
    n = len(xi)
    jk = [j for j in range(k)] + [j for j in range(k+1,n)]   # on exclut k de la liste [0,.., n]
    y = 1
    for i in jk:
        y *= (x - xi[i])/(xi[k] - xi[i])
    return y

def InterpLagrange(xi,yi,x):
    ''' Interpolation de Lagrange formule non barycentrique '''
    y = 0*x
    n = len(xi)
    for k in range(n):
        Lk = k_baseLagrange(k,xi,yi,x)
        y += yi[k] * Lk
    return y
```

Listing 10: Un utilitaire pour générer les noeuds d'interpolation

```
def NoeudsUniformes(a,b,N):
    # édfinition de N noeuds d'interpolation de uniformement repartis sur un intervalle [a,b]
    return np.linspace(a,b,N)


def NoeudsTchebyChev(a,b,N):
    # édfinition des N noeuds d'interpolation de Tchebychev sur un intervalle [a,b] , avec a < b
    j = np.arange(N)
    t = np.cos((2*j+1)*np.pi/(2*N))
    return a + (t+1) * (b - a)/2
```