TP 6 : Procédé d'orthonormalisation de Gram-Schmidt

Dans ce TP on met en oeuvre la méthode d'orthonormalisation de Gram-Schmidt originale et sa version modifiée, plus stable numériquement. On illustre l'instabilité de la méthode originale sur un exemple simple.

Le procédé de Gram-Schmidt est une méthode pour orthonormaliser une famille libre de vecteurs d'un espace vectoriel muni d'un produit scalaire. A partir d'une famille libre $(v_1, ..., v_n)$ on construit une famille orthonormale $(e_1, ..., e_n)$ qui engendre les mêmes espaces vectoriels successifs : pour tout j inférieur à n, $F_j = \text{Vect}(e_1, \dots, e_j) = \text{Vect}(v_1, \dots, v_j)$.

L'étape générale de l'algorithme consiste à soustraire au vecteur v_{j+1} sa projection orthogonale sur l'espace F_j. On s'appuie sur la famille orthonormale déjà construite pour le calcul de projection. Notons le projecteur sur la direction $\mathbf{u}$ par

$$\text{proj}_{\mathbf{u}}\, \mathbf{v} = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle} \mathbf{u}.$$

L'algorithme s'écrit :

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{v}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{||\mathbf{u}_1||} \\
\mathbf{u}_2 &= \mathbf{v}_2 - \text{proj}_{\mathbf{u}_1} \mathbf{v}_2, & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{||\mathbf{u}_2||} \\
\mathbf{u}_3 &= \mathbf{v}_3 - \text{proj}_{\mathbf{u}_1} \mathbf{v}_3 - \text{proj}_{\mathbf{u}_2} \mathbf{v}_3, & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{||\mathbf{u}_3||} \\
&\vdots \quad \vdots & &\vdots \\
\mathbf{u}_k &= \mathbf{v}_k - \sum_{j=1}^{k-1} \text{proj}_{\mathbf{u}_j} \mathbf{v}_k, & \mathbf{e}_k &= \frac{\mathbf{u}_k}{||\mathbf{u}_k||}
\end{aligned}$$

1. Implémenter ce procédé en créant une fonction `gramschmidt` prenant une matrice rectangulaire en entrée et, si la famille de vecteurs colonnes est libre, renvoyant une matrice de même taille avec les vecteurs orthonormaux issus du procédé ci-dessus. On veillera à tester si la matrice a au moins autant de lignes que de colonnes, et à afficher un message d'erreur si la famille n'est pas libre, c'est à dire si la norme d'un vecteur devient trop petite ($< 10^{-20}$ par exemple). Application : tester la procédure sur la matrice

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & -1 & 2 \\ 1 & 1 & 2 \end{pmatrix}$$

2. Cette méthode originelle est instable numériquement. Pour l'illustrer, considérer le cas

$$A = \begin{pmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{pmatrix}$$

avec par exemple $\varepsilon = 10^{-10}$. Quel est le résultat exact ? Quel résultat donne votre procédure d'orthonormalisation ? Est-ce une famille orthogonale ? Cette anomalie vient du fait que du fait des erreurs d'arrondi, les vecteurs $\mathbf{u}_k$ ne sont pas orthogonaux. Cependant le procédé peut être stabilisé en l'implémentant de la manière suivante : Plutôt que de calculer $\mathbf{u}_k$ par $\mathbf{u}_k = \mathbf{v}_k - \text{proj}_{\mathbf{u}_1} \mathbf{v}_k - \text{proj}_{\mathbf{u}_2} \mathbf{v}_k - \cdots - \text{proj}_{\mathbf{u}_{k-1}} \mathbf{v}_k$, on le calcule par

$$\begin{aligned}
\mathbf{u}_k^{(1)} &= \mathbf{v}_k - \text{proj}_{\mathbf{u}_1} \mathbf{v}_k, \\
\mathbf{u}_k^{(2)} &= \mathbf{u}_k^{(1)} - \text{proj}_{\mathbf{u}_2} \mathbf{u}_k^{(1)}, \\
&\vdots \\
\mathbf{u}_k^{(k-2)} &= \mathbf{u}_k^{(k-3)} - \text{proj}_{\mathbf{u}_{k-2}} \mathbf{u}_k^{(k-3)}, \\
\mathbf{u}_k &= \mathbf{u}_k^{(k-2)} - \text{proj}_{\mathbf{u}_{k-1}} \mathbf{u}_k^{(k-2)}.
\end{aligned}$$

Ce calcul donnerait le même résultat en arithmétique exacte mais est plus stable en arithmétique approchée.

3. Implémenter ce second algorithme en modifiant très légèrement la fonction précédente et comparer le résultat qu'il fournit sur l'exemple ci-dessus. Etonnant, non ?

2. On considère à présent l'application linéaire dérivée :

$$\begin{array}{rccc} \mathcal{D} & : & \mathbb{R}_n[X] & \to & \mathbb{R}_n[X] \\ & & P & \mapsto & P', \end{array}$$

associant à tout polynôme P de $\mathbb{R}_n[X]$ ayant pour coefficients a_k, $k = 0, \ldots, n$, le polynôme P' ayant pour fonction polynomiale $P'(x) = \sum_{k=1}^{n} k\, a_k\, x^{k-1}$. Reprendre la question précédente avec $\mathcal{D}$ en place de $\mathcal{R}$.

3. On considère enfin les deux applications composées $\mathcal{D} \circ \mathcal{R}$ et $\mathcal{R} \circ \mathcal{D}$. Que font-elles ? Sont-elles identiques ? Quel est leur lien avec les produits de matrices $M_{\mathcal{D}} M_{\mathcal{R}}$ et $M_{\mathcal{R}} M_{\mathcal{D}}$?

Exercice 3 (procédé d'orthonormalisation de Gram–Schmidt). On rappelle que, partant d'une famille $\mathcal{B} = \{x_1, \ldots, x_m\}$ de vecteurs linéairement indépendants de $\mathbb{R}^n$, avec m et n des entiers tels que $2 \leq m \leq n$, le procédé d'orthonormalisation de Gram–Schmidt permet de construire une famille $\mathcal{B}' = \{q_1, \ldots, q_m\}$ de vecteurs orthonormaux donnés par

$$q_1 = \frac{x_1}{\|x_1\|},$$

$$\widetilde{q}_{k+1} = x_{k+1} - \sum_{i=1}^{k} (x_{k+1}, q_i)\, q_i, \quad q_{k+1} = \frac{\widetilde{q}_{k+1}}{\|\widetilde{q}_{k+1}\|}, \quad k = 1, \ldots, m-1.$$

1. Écrire une fonction nommée `gramschmidt`, prenant comme paramètre d'entrée une matrice ayant pour colonnes les m vecteurs de la famille $\mathcal{B}$ et retournant en sortie une matrice ayant pour colonnes les m vecteurs de la famille $\mathcal{B}'$, obtenue en appliquant à $\mathcal{B}$ le procédé d'orthonormalisation de Gram–Schmidt. Penser à inclure une procédure vérifiant que la famille $\mathcal{B}$ fournie lors de l'appel de la fonction est bien libre.
2. On pose $\varepsilon = 10^{-8}$. Tester la fonction `gramschmidt` avec la famille

$$\mathcal{B} = \left\{ \begin{pmatrix} 1 \\ \varepsilon \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ \varepsilon \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ \varepsilon \end{pmatrix} \right\},$$

puis vérifier que les vecteurs obtenus sont bien orthogonaux deux à deux. Que constate-t-on ?
3. Pour pallier les défauts d'orthogonalité observés des vecteurs de la famille $\mathcal{B}'$ (qui sont dûs à l'accumulation des erreurs d'arrondi, les calculs étant faits en arithmétique à virgule flottante), il faut utiliser une version plus stable du procédé. Celle-ci consiste à opérer de la manière suivante :

$$q_1 = \frac{x_1}{\|x_1\|},$$

$$q_{k+1}^{(0)} = x_{k+1}, \; q_{k+1}^{(i)} = q_{k+1}^{(i-1)} - \left(q_{k+1}^{(i-1)}, q_i\right) q_i, \; i = 1, \ldots, k, \; q_{k+1} = \frac{q_{k+1}^{(k)}}{\|q_{k+1}^{(k)}\|}, \; k = 1, \ldots, m-1.$$

Mettre en œuvre, en modifiant la fonction déjà existante, cette variante pour obtenir une nouvelle fonction qu'on nommera `modgramschmidt` (pour procédé de Gram–Schmidt modifié). Effectuer ensuite l'orthonormalisation de la famille $\mathcal{B}$ donnée dans la question précédente.

On utilise fréquemment la commande disp pour afficher un message. On utilise également la commande disp pour afficher un résultat. On remarque que l'usage de la commande disp est un peu particulier. En effet un tableau doit être d'un type donnée. Puisque le tableau ne peuvent donc être des chaînes de caractères et des vecteurs remarque. On a donc recours à la commande num2str (num to string) pour convertir la valeur numérique à la chaîne de caractères

WikiMath

Des mathematiques, du Prolog, et du Python

13 mars 2014

Wiki

- Accueil
- Groupe

Python

- Général
- Modules

Aide

- Polices MathML

edit SideBar

Rechercher

-

Gram Schmidt

Sur cette page... (hide)

1. Motivations

Le procédé d'orthonormalisation consiste à prendre une base (b1,...,bn) de l'espace vectoriel considéré, et à la transformer en une base orthonormée (e1,...,en) :

- les vecteurs sont deux à deux orthogonaux,
- ils sont de norme égale à 1.

Ce procédé se généralise immédiatement à toute famille libre (b1,...,bk) qui peut ainsi être transformée en une famille (libre) orthonormée (e1,...,ek) engendrant le même espace.

En fait, on peut montrer que (e1,...,ek) est l'*unique* famille orthonormée engendrant le même espace que (b1,...,bk), telle que <ei,bi>>0 (produit scalaire strictement positif, *i.e.* les vecteurs "pointent dans la même direction").

Finalement, la méthode de *Gram-Schmidt* consiste à former une famille orthonormale en "redressant", puis en normant, progressivement chacun des vecteurs fournis, pour les rendre orthogonaux aux vecteurs déjà formés.

2. L'Algorithme

Le principe de cette orthonormalisation se comprend aisément :

1. On divise b1 par sa norme, ce qui donne e1 :

$$e1=b1||b1||$$

   Ainsi, notre premier vecteur de la nouvelle base sera bien normé.

1. On annule la composante de b2 suivant e1, pour que le résultat soit orthogonal à e1, puis on divise le résultat par sa norme :

$$e2=b2-<b2,e1>e1||b2-<b2,e1>e1||$$

   En effet, b2-<b2,e1>e1 est orthogonal à e1 (prendre leur produit scalaire pour s'en convaincre) : (e1,e2) forme bien une famille orthonormale.

1. On annule la composante de b3 suivant e1 et e2, pour que le vecteur résultant soit bien orthogonal à ces deux vecteurs, et on divise le résultat par sa propre norme :

$$e3=b3-<b3,e1>e1-<b3,e2>e2||b3-<b3,e1>e1-<b3,e2>e2e1||$$

1. On continue jusqu'à épuiser tous les vecteurs bi...

$$ei=bi-\sum_{j=1}^{i-1}<bi,ej>ej||bi-\sum_{j=1}^{i-1}<bi,ej>ej||$$

3. Gram-Schmidt et sympy

Montrons comment réaliser le procédé d'orthonormalisation de Gram-Schmidt avec **python-sympy**.

3.1 Préliminaires

Commençons par définir trois vecteurs à orthonormaliser :

```
>>> from sympy import *
>>> B1 = Matrix((2,1,2)).reshape(3,1)
>>> B2 = Matrix((12,-6,0)).reshape(3,1)
>>> B3 = Matrix((-3,-3,18)).reshape(3,1)
```

Avant toutes choses, vérifions que l'on a bien une base. Nous allons former une matrice avec ces trois vecteurs, puis nous assurer que le déterminant de cette matrice est bien non nul :

```
>>> M=B1.row_join(B2).row_join(B3)
>>> M
[2, 12, -3]
[1, -6, -3]
[2,  0, 18]
>>> M.det()
```

```
-540
```

Ces vecteurs forment donc bien une base !

3.2 Fonction prédéfinie

Pour orthonormaliser façon Gram-Schmidt en **sympy** :

```
>>> GramSchmidt([B1,B2,B3], orthog = True)
[ [2/3]
[1/3]
[2/3], [ 2/3]
[-2/3]
[-1/3], [-1/3]
[-2/3]
[ 2/3] ]
```

Si on a juste besoin d'orthogonaliser (pas normer), ne pas mettre le paramètre **orthog** :

```
>>> GramSchmidt([B1,B2,B3])
[ [2]
[1]
[2], [ 8]
[-8]
[-4], [ -5]
[-10]
[ 10] ]
```

Le produit scalaire entre vecteurs s'obtient avec la méthode **dot**, et la norme avec **norm**. On peut donc vérifier que l'on a bien une base orthogonale non normée :

```
>>> (u,v,w) = GramSchmidt([B1,B2,B3])
>>> u.dot(v)
0
>>> u.dot(w)
0
>>> v.dot(w)
0
>>> u.norm()
3
```

3.3 A la main

On a tout ce qu'il faut pour orthonormaliser à la main. Commençons par calculer le premier vecteur e1 de la nouvelle base (c'est b1 divisé par sa norme) :

```
>>> e1 = B1/B1.norm()
>>> e1
[2/3]
[1/3]
[2/3]
>>> e1.norm()
1
```

Continuons le procédé, en suivant l'algorithme ci-dessus. On vérifiera, à chaque étape, la norme, et le produit scalaire avec les autres vecteurs :

```
>>> e2 = B2-B2.dot(e1)*e1
>>> e2 = e2/e2.norm()
>>> e2
[ 2/3]
[-2/3]
[-1/3]
>>> e2.norm()
1
>>> e1.dot(e2)
0
>>> e3 = B3-B3.dot(e1)*e1-B3.dot(e2)*e2
>>> e3 = e3/e3.norm()
>>> e3
[-1/3]
[-2/3]
[ 2/3]
>>> e3.norm()
1
>>> e3.dot(e1)
0
>>> e3.dot(e2)
0
```

On a donc réussi, en appliquant le procédé d'orthonormalisation de Gram-Schmidt, à transformer la base

(212),(12-60),(-3-318)

en la base *orthonormée* :

(231323),(23-23-13),(-13-2323)

4. Travaux pratiques

1. Faire le même travail avec numpy.
2. Faire une fonction qui reçoit une liste de vecteurs en entrée, et qui les orthonormalise.
3. Faire en sorte que votre fonction puisse normaliser, ou non.

Avec la participation de Laurence Pilard, Karine Deschinkel, Michel Salomon, André Turberg et Husam Alustwani

Dernière modification : 21/10/2009 15:13
Design theundersigned | mod par CarlosAB
Powered by PmWiki

$$\text{proj}_u v = \frac{\langle u, v \rangle}{\langle u, u \rangle} u.$$

$$e_2 = \frac{u_2}{\|u_2\|} = v_2 - \text{proj}_{u_1} v_2.$$

$$= v_2 - \frac{\langle u_1, v_2 \rangle}{\langle u_1, u_1 \rangle} v_2$$

$$\frac{}{\|u_2\|}$$

$$u_1 = v_1$$

$$u_2 = v_2 - \text{proj}_{u_1} v_2.$$

$$\tilde{q}_1 = x_1$$

$$\tilde{q}_2 = x_2 - (x_2, q_1) q_1 \qquad q_2 = \frac{\tilde{q}_2}{\|q_2\|}$$

$g \circ f = 0 \quad \mathcal{L}(E, G)$ $\qquad rg(g \circ f) = rg(y).$

Soit $\beta \in Im(\beta)$.

d'où

$$f(1,1,1) = 3.$$

$$f(1,0,2) = 0$$

$$f(1,0,0) = 2.$$

$$f(1,1,1) = \underbrace{ax + by + cz}_{IR}.$$

$$a + b + c = 3. \qquad b =$$

$$a + 2c = 0 \qquad c = -1.$$

$$a = 2. \qquad 2 - 1 \quad b + 1 = 3.$$

$$b = 2.$$

$rg(g \circ f) \leq rg(g)$

TP6:

Méthode de la puissance:

① Critère d'arrêt:

Méthode de la puissance est une méthode itérative pour trouver une valeur approchée de la plus grande valeur propre en module.

~~Grâce~~ Pour que. $P_2 \longmapsto +\infty$.

$$q^{(P_2)} \longrightarrow V.$$

$$\frac{V^{(P_2)}}{\|V^{(P_2)}\|} \xrightarrow{P_2 \mapsto +\infty} \lambda \quad : \text{La plus grande valeur propre approchée en module}$$

$q^{(P_2)} \xrightarrow{P_2 \mapsto +\infty} V$: Le plus grand vecteur propre associé à λ.

càd $AV = \lambda V$.

Vecteur propre — Valeur propre

Pratiquement On peut Trouver les valeurs propres exactes d'une matrice avec $\text{cig}(A)$.

Si on a un tel algorithme, il converge

Pourquoi

On peut mettre fin aux itérations de l'algorithme ci dessous de la méthode puissance.

$$\left| \nu^{(p_2+1)} - \nu^{(p_2)} \right| < \varepsilon \qquad \forall p \geq 1$$

Tolérance $\varepsilon \ll 1$.

critère d'arrêt

```
// ProcÃ©dÃ© d'orthogonalisation de Gram-Schmidt
// Entree : une matrice rectangulaire
// Sortie : si la famille des vecteurs colonne est libre,
// une famille orthogonale engendrant le meme sev
//
function res=gramschmidt(A)
  [n,m]=size(A);
  if (m>n)
    printf("Une famille de %d vecteurs de taille %d ne peut pas Ãªtre
libre\n",m,n);
    return(A);
  end

  norme = A(:,1)'*A(:,1);
  if (norme<1e-20)
    printf("La famille est liÃ©e\n");
    return(A);
  end
  A(:,1)=A(:,1)/sqrt(norme);
  for j=2:m
    for k=1:j-1
      ps = A(:,j)'*A(:,k);
      A(:,j) = A(:,j) - ps*A(:,k);
    end
    norme = A(:,j)'*A(:,j); A(:,j)=A(:,j)/sqrt(norme);
    if (norme<1e-20)
      printf("La famille est liÃ©e\n");
      return(A);
    end
 end
res = A;
endfunction
```

ita = ita + 1;

z = A * V.

morn f (ita) = move(z, 2)

Panda (ita) = conj(V') * A * V.

res = abs(Panda(ita) - Panda(ita - 1))

End

end.

```
function [Lamda, V, itai] = puissance (A, q0, tol, nmax)
% calcul de la valeur propre approchée en module de la matrice
A
[n m] = size(A).
if (n ~= m)
[ error('la matrice n''est pas carée');
end.
itai = 1;
Z = A * q0;
normf(1) = norm (Z, 2);
V = Z / normf(1);
Lamda (1) = conj(V') * A * qV;
% Nbcons res = critère d'arrêt.
res = tol + 1;
while ((itai < nmax) & (res >= tol)).
```

$$u_{p_2}^{(1)} = v_{p_2} - \text{proj}_{u_1} v_{p_2}$$

$$u_{p_2}^{(2)} = u_{p_2}^{(1)} - \text{proj}_{u_2} u_{p_2}^{(1)} ;$$

$$\vdots$$

$$u_{p_2}^{(p_2-2)} = u_{p_2}^{(p_2-3)} - \text{proj}_{u_{p_2-2}} u_{p_2}^{(p_2-3)}$$

$$u_{p_2} = u_{p_2}^{(p_2-2)} - \text{proj}_{u_{p_2-1}} u_{p_2}^{(p_2-2)}$$

---

for $p_2 = 1, \ldots, m$

```
n = 20;
u = zeros(n,1);
u(1) = 0;
u(2) = 1;
For i = 3 : n
  u(i) = u(i-1) + u(i-2)
  end
u.
```

```
u = [ ]
u(1) = 0;
u(2) = 1;
n = 2;
while u(n) <= 50000
    n = n+1;
    u(n) = u(n-1) + u(n-2);
    n = n + 1
end.
u(n-1)
un
```

```
function u = fibo(n)
if n == 1
    u = 0;
else if n == 2
    u = 1;
else
  umA = 0 , umn = 1;
  For i = 3 : n.
  u = umA + umB;
  umA = umn;
  umn = u;
  end
  end.
```

$$V^{10} \simeq \frac{\sqrt{27}}{\Pi}$$

$$\Pi \simeq \frac{\sqrt{27}}{V^{10}}$$

```
n = 10;
u = 1 , v = 2.
For i = 1 : n.
u = (u + v) / 2. ;
V = sqrt(u*v);
end.
sqrt(27)/V
```

[xi ; ita, nbs, cinc] = dichotomie (f, 2, 3, 1-e10, 1000)

Question ⑥

TP

semilogy(cinc, 'r')

Par définition

$cinc(P_k - 1) = |x^{(P_k)} - x^{(P_k - 1)}| \leq \epsilon$

à l'itération P_k.

$\log|x^{(P_k)} - x^{(P_k - 1)}|$ ⟵ $P_k - 1$

$\log|x^{(P_k - 1)} - x^{(P_k - 2)}|$ ⟵ $P_k - 2$

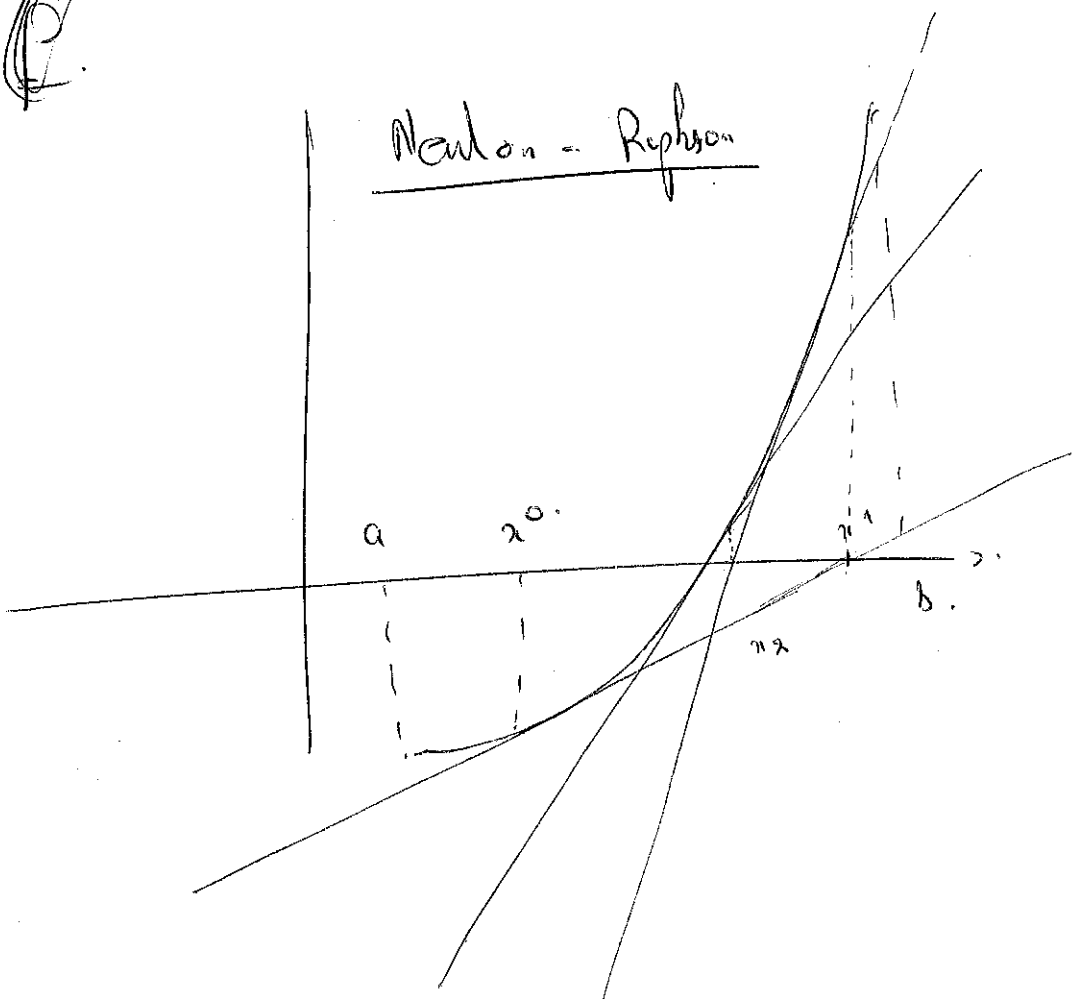
$$\frac{\log|x^{(P_k)} - x^{(P_k - 1)}| - \log|x^{(P_k - 1)} - x^{(P_k - 2)}|}{P_k - (P_k - 1)} = p$$

$$\log|x^{(P_k)} - x^{(P_k - 1)}| - \log|x^{(P_k - 1)} - x^{(P_k - 2)}| = p$$

$$\log\left|\frac{x^{(P_k)} - x^{(P_k - 1)}}{x^{P_k - 1} - x^{P_k - 2}}\right| = p \Rightarrow \underline{\log 10^p}$$

$p \log(10) =$

Question 6.



$$\begin{cases} x^{(k+1)} = x^{(k)} - \dfrac{f(x^{(k)})}{f'(x^{(k)})} & \forall k \geq 0 \\ x^{(0)} \text{ donné} \end{cases}$$

$$\boxed{f(x^{(k)}) + (x - x^{(k)})\, f'(x^{(k)}) = 0}$$

Rq $f(x_0) + (x - x_0) f'(x^0) = 0$

$$\left| \frac{x^{(p_2)} - x^{(p_1)}}{x^{p_1} - x^{p_2}} \right| = \underline{10^p} = c.$$

$$\left| x^{(p_2)} - x^{(p_1)} \right| = 10^p \left| x^{p_1} - x^{p_2} \right| \qquad p_2$$

$$\left| x^{p_1} - x^{p_2} \right| = 10^p \left| x^{p_2} - x^{p_3} \right|$$

$$\vdots$$

$$\left| x^{p_2} - x^{p_1} \right| = 10^p \left| x^{(1)} - x^{(0)} \right|.$$

Loi $\left| x^{(p_2)} - x^{(p_1)} \right| = \left(10^p\right)^{p_1} \left| x^{(1)} - x^{(0)} \right|$

graphiquement $p = \underline{-0,3}.$

$$c = 10^p = 10^{-0,3} = 0,5.$$

$$\left| x^{(p)} - x^{p_1} \right| \simeq \left(\frac{1}{2}\right)^{p_1} \left| x^1 - x^0 \right|.$$

```
function res = mid(n).
x0 = ones(n,1).
H = hilb(n)
b = H * x0.
x = inv(H) * b.
res. = nor(x-x0) / nor(x0);
return.

for h=1:50
  er(h)=mid(h);
disp('Affichage');
plot(er,'-b');

x0= rand(n,1).
```

Section :

Résolution d'équation non linéaire :

Position du problème

La question qu'on résout toujours, c'est résoudre le système $Ax = b$.
(Méthode directe, méthode itérative).

Formulation d'une équation linéaire

Question

$f(\lambda x) \neq \lambda x$.

Soit si $f(\lambda x) \neq \lambda x$ équation non linéaire

Exemple

Question : soit une fonction f associée à une équation non lin.

Ex q/c $f(x) = x^2 + 2x + 1$.

$f(x) = ax + b$. $a, b \in \mathbb{R}$

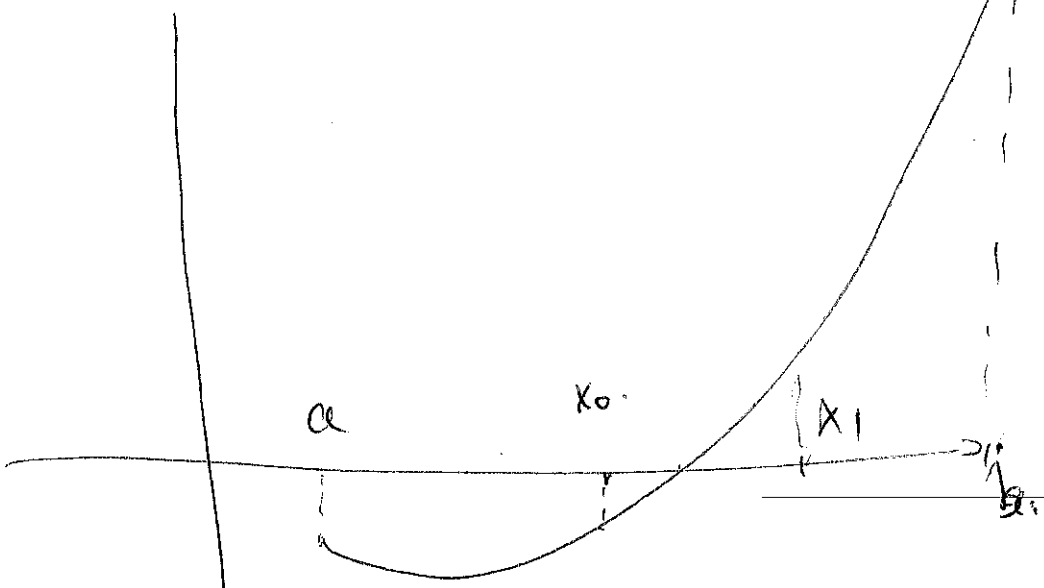
$f(x) = e^{ax}$. $a \in \mathbb{R}$.

val $f(\eta) = 0$. c'à d.

$\sim \exists \xi \in IR$ tq $f(\xi) = 0$.

Idées maîtres:

1) Méthode dict dichotomie (bisection).



$x_0 = \frac{a+b}{2} = \frac{a^{(0)} + b^{(0)}}{2}$. à l'itération 0

Soit

* Si $f(a) \cdot f(x_0) < 0$ alors $\xi \in [a^0, x^0]$.

ou

sinon Si $f(x_0) \cdot f(b) < 0$ alors $\xi \in [x^0, b^0]$.

$x_1 = x_0$ $a^{(1)} = x_0$ $b^{(1)} = b$.

$x_1 = \frac{a^{(1)} + b^{(1)}}{2} = \frac{x_0 + b}{2}$.

$f(x_0) \cdot f(x_1) < 0 \Rightarrow \xi \in [x_0, x_1]$.

$f(x_1) \cdot f(b) < 0 \Rightarrow \xi \in [x_1, b]$

$a^{(k)}$; $b^{(k+1)} = x^{(k)}$ &

$f(a^{(k)}) \cdot f($

Université Paris-Dauphine
Département MIDO
DE MI2E deuxième année

Méthodes numériques
année 2012/2013

feuille de travaux pratiques

Séance 9 : interpolation polynomiale et formules de quadrature

Le symbole ⋄ indique un exercice optionnel. Le travail demandé peut être effectué indifféremment avec MATLAB ou le logiciel libre GNU OCTAVE (`http://www.gnu.org/software/octave/`).

Exercice 1 (interpolation de Lagrange à l'aide des matrices de Vandermonde). Dans MATLAB et GNU OCTAVE, toute fonction polynomiale de degré $n \geq 0$, $p(x) = a_0 + a_1\,x + a_2\,x^2 + \cdots + a_n\,x^n$, est représentée par un tableau `p=[an,...,a2,a1,a0]` contenant [1] les $n+1$ coefficients du polynôme qui lui est associé. La commande `polyval(p,x)` permet alors d'évaluer n'importe quelle fonction polynomiale p en un point (ou groupe de points) x donné. Enfin, la commande `vander(x)` renvoie la matrice de Vandermonde associée à un ensemble de nœuds contenu dans le tableau `x`.

1. Écrire une fonction `lagrange`, ayant pour arguments une fonction continue f, les bornes d'un intervalle $[a,b]$ et un entier positif n, construisant le polynôme d'interpolation de Lagrange $\Pi_n f$ associé à une distribution uniforme de $n+1$ points dans l'intervalle $[a,b]$ par résolution du système linéaire de Vandermonde correspondant.
2. Utiliser cette fonction pour construire les polynômes d'interpolation de Lagrange $\Pi_n \sin$ de la fonction sinus à nœuds équirépartis sur l'intervalle $[0,3\pi]$, avec $n = 1,\ldots,5$. Comparer les graphes des polynômes obtenus avec celui de la fonction donnée sur ce même intervalle.
3. Évaluer de manière approchée l'erreur
$$E_n(\sin) = \max_{x\in[0,3\pi]} |\sin(x) - \Pi_n \sin(x)|$$
et représenter sur un graphe les valeurs obtenues en fonction de n, pour $n = 1,\ldots,12$. Que se passe-t-il pour $n > 12$?
4. En observant que $|\sin^{(n)}(x)| \leq 1$, $\forall n \in \mathbb{N}$ et $\forall x \in \mathbb{R}$, comparer les valeurs de l'erreur obtenues à la question précédente avec celles fournies par la majoration théorique
$$E_n(f) \leq \frac{1}{4(n+1)} \left(\frac{b-a}{n}\right)^{n+1} \max_{x\in[a,b]} |f^{(n+1)}(x)|.$$

Exercice 2 (phénomène de Runge et points de Tchebychev). On considère, sur l'intervalle $[-5,5]$, la fonction
$$f(x) = \frac{1}{1+x^2}.$$

1. Utiliser la fonction `lagrange` de l'exercice précédent pour construire le polynôme d'interpolation de Lagrange $\Pi_n f$ de degré n, avec $n = 2, 4, 8$ et 12, de f en des nœuds équirépartis sur $[-5,5]$ et comparer graphiquement les polynômes obtenus avec la fonction donnée.
2. Évaluer de manière approchée l'erreur
$$E_n(f) = \max_{x\in[-5,5]} |f(x) - \Pi_n f(x)|$$
et représenter sur un graphe les valeurs obtenues en fonction de n, pour $n = 1,\ldots,12$. Qu'observe-t-on ?

1. On notera que le premier élément du vecteur correspond au coefficient du terme de plus haut degré.

Interpoler une fonction en des nœuds équidistribués sur un intervalle $[a, b]$ n'est pas forcément le meilleur choix, comme le montre l'absence de convergence de l'interpolation de Lagrange constatée à la question précédente. Pour une interpolation de degré n, on peut montrer que l'erreur sera minimale si les nœuds d'interpolation $\{x_k\}_{0 \leq k \leq n}$ sont (à une transformation affine près) les racines du polynôme de Tchebychev T_{n+1}, c'est-à-dire si

$$x_k = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2k+1}{2(n+1)}\,\pi\right), \ \forall k \in \{0, \ldots, n\}.$$

3. Modifier la fonction `lagrange` pour que l'interpolation soit faite aux points de Tchebychev définis ci-dessus.
4. Reprendre alors les questions 1 et 2.

Exercice 3 ◇ (ordre de convergence de formules de quadrature composées). Soit f une fonction réelle continue sur l'intervalle $[0, 1]$. Le but de cet exercice est de mesurer l'efficacité de formules de quadrature de Newton–Cotes composées classiques pour l'approximation de l'intégrale définie

$$I(f) = \int_a^b f(x)\,\mathrm{d}x,$$

pour différents choix de fonction, et l'influence de la régularité de la fonction sur la vitesse de convergence de la méthode.

1. Écrire trois fonctions, ayant chacune pour paramètres d'entrée la fonction f, les bornes a et b, et un nombre $m \geq 1$ de subdivisions de l'intervalle $[a, b]$, calculant une approximation $I_{m,n}(f)$ de $I(f)$ respectivement par les formules de quadrature de la règle du point milieu (formule ouverte, $n = 0$), de la règle du trapèze (formule fermée, $n = 1$) et de la règle de Simpson (formule fermée, $n = 2$) composées.
2. Au moyen de la commande `semilogy`, tracer le graphe des courbes de l'erreur $|I(f) - I_{m,n}(f)|$ de chacune de ces trois formules en fonction du nombre de sous-intervalles pour $f(x) = e^x$, $a = 0$ et $b = 1$. Commenter les résultats.
3. Reprendre la question précédente pour $f(x) = \left|3\,x^4 - 1\right|$. Que constate-t-on ? Comment procéder pour retrouver les ordres de convergence théoriques des formules dans ce cas ?

Exercice 4 ◇ (méthode de Romberg). On considère l'utilisation de la règle du trapèze composée associée à une subdivision dyadique de l'intervalle $[a, b]$ pour le calcul de l'intégrale $I(f)$ de l'exercice précédent. En supposant la fonction f suffisamment régulière et en posant $H = \frac{b-a}{2^k}$, $k \geq 0$, on peut montrer à partir de la formule d'Euler–Maclaurin que

$$I_{2^k,1}(f) = I(f) + b_2\,H^2 + b_4\,H^4 + \ldots$$

À partir d'une estimation de l'intégrale pour une subdivision de taille $\frac{H}{2}$, on obtient, grâce au procédé d'extrapolation de Richardson, une meilleure approximation de $I(f)$, fournie par la quantité

$$R(k+1,1) = R(k+1,0) + \frac{1}{3}\,(R(k+1,0) - R(k,0)) = \frac{1}{3}\,(4\,R(k+1,0) - R(k,0)),$$

où l'on a posé $R(k,0) = I_{2^k,1}(f)$ et $R(k+1,0) = I_{2^{k+1},1}(f)$, et qui est d'ordre quatre en H. La méthode de Romberg pour l'évaluation approchée de $I(f)$ consiste simplement en l'application répétée de cette opération à partir de la valeur $k = 0$.

Écrire une fonction `romberg`, ayant pour arguments d'entrée une fonction f, les bornes a et b, un nombre d'extrapolations maximal N et une tolérance ε, renvoyant la valeur de l'approximation $R(k,k)$ telle que $|R(k,k) - R(k-1,k-1)| \leq \varepsilon$, avec $0 \leq k \leq N$, ou bien $k = N$. Pour cela, on construira un tableau des valeurs extrapolées $R(k,m)$, $0 \leq m \leq k \leq N$, dont les éléments satisfont la relation de récurrence

$$R(k,m) = \frac{1}{4^m - 1}\,(4^m\,R(k,m-1) - R(k-1,m-1)), 1 \leq m \leq k \leq N,$$

et dont la première colonne est telle que $R(k,0) = I_{2^k,1}(f)$, $k = 0, \ldots, N$.

$$\Pi_n(x) = \sum_{j=0}^{n} x^j a_j$$

$$\Pi(x_0) = y_0$$
$$\Pi(x_1) = y_1$$

$$\Pi_1(x_A) = a_0 + x_0^1 a_1 = y_0$$
$$a_0 + x_1^1 a_1 = y_0$$

$$\begin{pmatrix} 1 & x_0^1 \\ 1 & x_1^1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_0 \end{pmatrix}$$

$$\begin{cases} \Pi_n(x) = \sum_{j=0}^{n} x^j a_j & \forall x \in \mathbb{R} \\ \Pi_n(x_i) = y_i & i = 0, \dots n. \end{cases}$$

$$a_0 + a_1 x_i + a_2 x_i^2 + \dots + a_n x_i^n = y_i$$

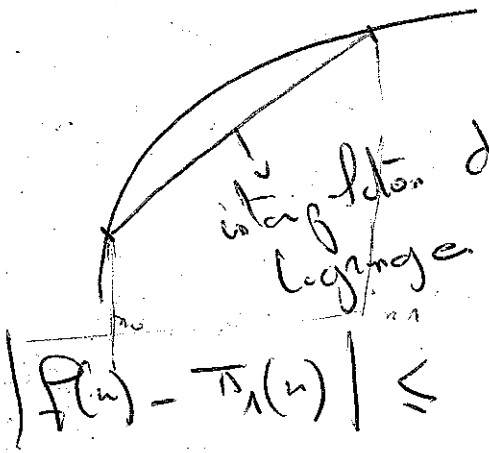
$$i = 0, \dots n.$$

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ & x_1 & & & \\ & \vdots & & & \\ & \vdots & & & \\ 1 & x_n & \dots & & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ \vdots \\ \vdots \\ y_n \end{pmatrix}$$

Van Vandermonde

$Ax = b$

$A \setminus b$



$$|f(x) - \Pi_1(x)| \leq \dots$$

DÉCLARATION DE CANDIDATURE

Mr BEY Mohamed-Amine

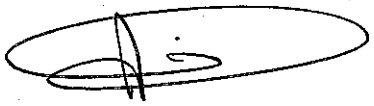
à

Monsieur le Président
de l'Université Paris-Est Marne-la-Vallée

J'ai l'honneur de solliciter mon recrutement ou renouvellement en qualité d'Attaché Temporaire d'Enseignement et de Recherche à l'université Paris-Est Marne-la-Vallée.

Fait à Paris le 15/04/2013

Signature



$Df(x) = g(x) - x$. Soit f une fonction

$Df \in (1, +\infty)$ qu'elconque.

$$\Pi_n(x) = \sum_{j=0}^{n} a_j x^j = f(x).$$
$$\forall x \in \mathbb{R}.$$

$x = (x_0, x_1, \ldots x_i \ldots x_n)$.

$$\Pi_n(x_i) = \sum_{j=0}^{n} a_j x_i^j = f(x_i) \quad \forall i \in \{0, \ldots n\}$$

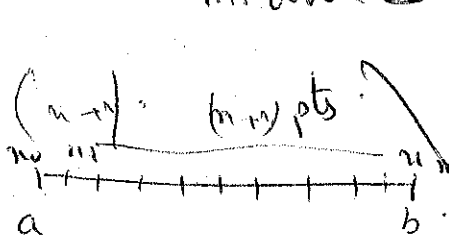
$$a_0 \underbrace{x_i^0}_{1} + a_1 x_i^1 + a_2 x_i^2 + a_3 x_i^3 + \ldots$$
$$+ \ldots a_n x_i^n = y_i$$
$$\forall i \in \{0, \ldots n\}$$

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

matrice Vander. ch

$[a = inv(V) \cdot y]$

$(n+1)$: $(n+1)$ pts



(a) matlab $= y$ $[a = V \backslash y]$ ou

```
nmax = 20;
for n=1:nmax.
e(n) = max(abs(sin(x) - polyval(Pograng(
                                @, sn, a, b, n))));
end
demiPogy(e);
```

$\log|E_n(sin)|$ $\log|\max E_n(sin)|$.

~~For~~ Mon script :

```
function p = Lagrange(func, a, b, n)
% construction du polynôme d'interpolation
de Lagrange de degré n.
xp = linspace(a, b, n+1)
p = vander(xp) \ feval(func, xp')
end
```

```
a = 0;   b = 3*pi;
x = linspace(a, b, 1000);
plot(x, sin(x), polyval(Lagrange(@sin, a, b, 1), x))
```

```
clear all; close all;
% question 1.
disp('Question 1.');
a=0; b=3*pi;
x=linspace(a,b,1000);
disp(['Trace des graphes sur l''intervalle [',num2str(a),',',num2str(b),']...']);
figure;
plot(x,sin(x),x,polyval(lagrange(@sin,a,b,1),x),x,polyval(lagrange(@sin,a,b,2),x),x,polyval(lagrange
(@sin,a,b,3),x),x,polyval(lagrange(@sin,a,b,4),x),x,polyval(lagrange(@sin,a,b,5),x));
legend('sin(x)','Pi_1sin(x)','Pi_2sin(x)','Pi_3sin(x)','Pi_4sin(x)','Pi_5sin(x)');
disp('Appuyer sur une touche pour la question suivante.');
pause
%%
% question 2.
disp('Question 2.');
disp('Trace du graphe de l''erreur d''interpolation E_n(sin) en fonction de n...');
nmax=20;
for n=1:nmax
   e(n)=max(abs(sin(x)-polyval(lagrange(@sin,a,b,n),x)));
end
figure;
semilogy(e);
legend('E_n(sin)');
disp('Appuyer sur une touche pour la question suivante.');
pause
%%
% question 3.
disp('Question 3.');
disp('Trace du graphe...');
for n=1:nmax
   m(n)=((b-a)/n)^(n+1)/(4*(n+1));
end
semilogy(1:nmax,e,1:nmax,m);
legend('E_n(sin)','majoration theorique de l''erreur d''interpolation');
```

```
function p=lagrange(func,a,b,n)
% Construction du polynome d'interpolation de Lagrange de degre n d'une fonction func en n+1 points
equidistribues dans l'intervalle [a,b]
xp=linspace(a,b,n+1);
p=polyfit(xp,feval(func,xp),n);
return
```

http://www.gnu.org/software/octave/.

3) function y = fct64(n)

```
if (n == 0),
    y = 1;
else if (n == 1),
    y = 1;
else
    y = fct64(n-1) + fct64(n-2);
end
```

F(16) = 97

fct64(16).

2)

```
V(1) = 0;
V(2) = 1;
i = 2.
while (V(i) <= 50000)
    V(i+1) = V(i) + V(i-1)
    i = i + 1;
end
V(i-1),
```

1) function V = fctFin(n)

```
V = zeros(1, n);
for i = 1:n
if (i == 1)
    V(i) = 0;
else if (i == 2)
    V(i) = 1;
else
V(i) = V(i-1) + V(i-2);
end
end
```

```
function [zero,iter,res,inc]=newton(func,dfunc,x0,tol,nmax)
% Recherche d'un zero d'une fonction func derivable par la methode de Newton a partir du point x0.
% En cas de succes, la fonction retourne l'approximation du zero obtenue, le numero de l'iteration a
laquelle cette approximation a ete calculee, la valeur de la fonction func en ce point et un vecteur
contenant la valeur absolue de la difference entre deux approximations successives (increments).
% Si la recherche echoue, un message d'erreur est affiche.
% Note : la fonction func et sa derivee dfunc prennent en entree un argument scalaire et renvoient un
scalaire en sortie.
x=x0;
iter=0;
diff=tol+1;
while (abs(diff)>=tol&iter<=nmax)
   iter=iter+1;
   fx=feval(func,x); dfx=feval(dfunc,x);
   diff=-fx/dfx;
   x=x+diff;
   inc(iter)=abs(diff);
end
if iter>nmax
   fprintf(['Le nombre maximum d''iterations a ete atteint sans convergence avec la tolerance desiree.
\n']);
end
zero=x;
res=feval(func,x);
return
```

```
clear all; close all;
% question 1.
disp('Question 1.');
disp(['La valeur de la racine carree de 2 fournie par la commande sqrt(2) est ',num2str(sqrt
(2),15),'.']);
disp(' ');
f=inline('x^2-2','x');
a=1; b=2; tol=1e-10;
[xi,iter,res,inc]=dichotomie(f,1,2,tol,1000);
disp(['L''approximation de cette valeur obtenue par la methode de dichotomie sur l''intervalle
[',num2str(a),',',num2str(b),'] (avec une tolerance egale a ',num2str(tol),' pour le critere d''arret)
est ',num2str(xi,15),', apres ',num2str(iter),' iterations.']);
disp(' ');
[xi,iter,res,inc]=regulafalsi(f,1,2,tol,tol,1000);
disp(['L''approximation de cette valeur obtenue par la methode de la fausse position sur l''intervalle
[',num2str(a),',',num2str(b),'] (avec une tolerance egale a ',num2str(tol),' pour le critere d''arret)
est ',num2str(xi,15),', apres ',num2str(iter),' iterations.']);
disp(' ');
df=inline('2*x','x');
[zero,iter,res,inc]=newton(f,df,(a+b)/2,tol,1000);
disp(['L''approximation de cette valeur obtenue par la methode de Newton a partir du point initial
',num2str((a+b)/2),' (avec une tolerance egale a ',num2str(tol),' pour le critere d''arret) est
',num2str(xi,15),', apres ',num2str(iter),' iterations.']);
disp(' ');
[xi,iter,res,inc]=secante(f,1,2,1e-10,1000);
disp(['L''approximation de cette valeur obtenue par la methode de la secante a partir des points
initiaux ',num2str(a),' et ',num2str(b),' (avec une tolerance egale a ',num2str(tol),' pour le critere
d''arret) est ',num2str(xi,15),', apres ',num2str(iter),' iterations.']);
disp(' ');
disp('Appuyer sur une touche pour la question suivante.');
pause
%%
% question 2.a.
disp('Question 2.a.');
disp('D''apres les resultats du cours, on a convergence d''une methode de point fixe associee a une
fonction g a valeurs d''un intervalle [a,b] dans lui-meme s''il existe une constante 0<K<1 telle que |
g''(x)|<=K pour tout x dans [a,b] (la fonction g etant alors contractante sur [a,b]).');
a=1; b=2;
x=linspace(a,b,100);
g1=inline('2+x-x.^2','x'); dg1=inline('1-2*x','x');
figure;
plot(x,g1(x),x,abs(dg1(x)));
grid on;
legend(['graphe de g1 sur [',num2str(a),',',num2str(b),']'],['graphe de |g1''| sur [',num2str
(a),',',num2str(b),']']);
g2=inline('2./x','x'); dg2=inline('-2./x.^2','x');
figure;
plot(x,g2(x),x,abs(dg2(x)));
grid on;
legend(['graphe de g2 sur [',num2str(a),',',num2str(b),']'],['graphe de |g2''| sur [',num2str
(a),',',num2str(b),']']);
g3=inline('(x+2)./(x+1)','x'); dg3=inline('-1./(x+1).^2','x');
figure;
plot(x,g3(x),x,abs(dg3(x)));
grid on;
legend(['graphe de g3 sur [',num2str(a),',',num2str(b),']'],['graphe de |g3''| sur [',num2str
(a),',',num2str(b),']']);
disp(' ');
disp(['D''apres les graphes des fonctions g1, g2, g3 et de leurs derivees sur l''intervalle [',num2str
(a),',',num2str(b),'], on en deduit que seule g3 verifie les hypotheses requises et conduit a une
methode convergente.']);
disp(' ');
disp('Appuyer sur une touche pour la question suivante.');
pause
%%
% question 2.b.
disp('Question 2.b.');
x=1/2;
for i=1:19
    x(i+1)=g1(x(i));
end
disp(['Les vingts premiers termes de la suite construite par la methode de point fixe associee a g1
```

```
sont : ',num2str(x,15),'.']);
disp(' ');
for i=1:19
    x(i+1)=g2(x(i));
end
disp(['Les vingts premiers termes de la suite construite par la methode de point fixe associee a g2
sont : ',num2str(x,15),'.']);
disp(' ');
for i=1:19
    x(i+1)=g3(x(i));
end
disp(['Les vingts premiers termes de la suite construite par la methode de point fixe associee a g3
sont : ',num2str(x,15),'.']);
disp(' ');
disp('L''affirmation de la reponse precedente se trouve donc confirmee.');
```

TP 6 : Procédé d'orthonormalisation de Gram-Schmidt

Dans ce TP on met en oeuvre la méthode d'orthonormalisation de Gram-Schmidt originale et sa version modifiée, plus stable numériquement. On illustre l'instabilité de la méthode originale sur un exemple simple.

Le procédé de Gram-Schmidt est une méthode pour orthonormaliser une famille libre de vecteurs d'un espace vectoriel muni d'un produit scalaire. A partir d'une famille libre $(v_1, ..., v_n)$ on construit une famille orthonormale $(e_1, ..., e_n)$ qui engendre les mêmes espaces vectoriels successifs : pour tout j inférieur à n, $F_j = \text{Vect}(e_1, \ldots, e_j) = \text{Vect}(v_1, \ldots, v_j)$.

L'étape générale de l'algorithme consiste à soustraire au vecteur v_{j+1} sa projection orthogonale sur l'espace F_j. On s'appuie sur la famille orthonormale déjà construite pour le calcul de projection. Notons le projecteur sur la direction $\mathbf{u}$ par

$$\text{proj}_{\mathbf{u}}\, \mathbf{v} = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle} \mathbf{u}.$$

L'algorithme s'écrit :

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{v}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{||\mathbf{u}_1||} \\
\mathbf{u}_2 &= \mathbf{v}_2 - \text{proj}_{\mathbf{u}_1} \mathbf{v}_2, & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{||\mathbf{u}_2||} \\
\mathbf{u}_3 &= \mathbf{v}_3 - \text{proj}_{\mathbf{u}_1} \mathbf{v}_3 - \text{proj}_{\mathbf{u}_2} \mathbf{v}_3, & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{||\mathbf{u}_3||} \\
&\vdots \quad \vdots & &\vdots \\
\mathbf{u}_k &= \mathbf{v}_k - \sum_{j=1}^{k-1} \text{proj}_{\mathbf{u}_j} \mathbf{v}_k, & \mathbf{e}_k &= \frac{\mathbf{u}_k}{||\mathbf{u}_k||}
\end{aligned}$$

1. Implémenter ce procédé en créant une fonction `gramschmidt` prenant une matrice rectangulaire en entrée et, si la famille de vecteurs colonnes est libre, renvoyant une matrice de même taille avec les vecteurs orthonormaux issus du procédé ci-dessus. On veillera à tester si la matrice a au moins autant de lignes que de colonnes, et à afficher un message d'erreur si la famille n'est pas libre, c'est à dire si la norme d'un vecteur devient trop petite ($< 10^{-20}$ par exemple). Application : tester la procédure sur la matrice

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & -1 & 2 \\ 1 & 1 & 2 \end{pmatrix}$$

2. Cette méthode originelle est instable numériquement. Pour l'illustrer, considérer le cas

$$A = \begin{pmatrix} 1 & 1 & 1 \\ \varepsilon & 0 & 0 \\ 0 & \varepsilon & 0 \\ 0 & 0 & \varepsilon \end{pmatrix}$$

avec par exemple $\varepsilon = 10^{-10}$. Quel est le résultat exact ? Quel résultat donne votre procédure d'orthonormalisation ? Est-ce une famille orthogonale ? Cette anomalie vient du fait que du fait des erreurs d'arrondi, les vecteurs $\mathbf{u}_k$ ne sont pas orthogonaux. Cependant le procédé peut être stabilisé en l'implémentant de la manière suivante : Plutôt que de calculer $\mathbf{u}_k$ par $\mathbf{u}_k = \mathbf{v}_k - \text{proj}_{\mathbf{u}_1} \mathbf{v}_k - \text{proj}_{\mathbf{u}_2} \mathbf{v}_k - \cdots - \text{proj}_{\mathbf{u}_{k-1}} \mathbf{v}_k$, on le calcule par

$$\begin{aligned}
\mathbf{u}_k^{(1)} &= \mathbf{v}_k - \text{proj}_{\mathbf{u}_1} \mathbf{v}_k, \\
\mathbf{u}_k^{(2)} &= \mathbf{u}_k^{(1)} - \text{proj}_{\mathbf{u}_2} \mathbf{u}_k^{(1)}, \\
&\vdots \\
\mathbf{u}_k^{(k-2)} &= \mathbf{u}_k^{(k-3)} - \text{proj}_{\mathbf{u}_{k-2}} \mathbf{u}_k^{(k-3)}, \\
\mathbf{u}_k &= \mathbf{u}_k^{(k-2)} - \text{proj}_{\mathbf{u}_{k-1}} \mathbf{u}_k^{(k-2)}.
\end{aligned}$$

Ce calcul donnerait le même résultat en arithmétique exacte mais est plus stable en arithmétique approchée.

3. Implémenter ce second algorithme en modifiant très légèrement la fonction précédente et comparer le résultat qu'il fournit sur l'exemple ci-dessus. Etonnant, non ?

Le cbfe manquait vat dire
qu'ne Pegcè enan sn Ps.
donée dlentéo donc un Pegcè
enan sn Ps. nsulld.

```
function [zero,iter,res,inc]=dichotomie(func,a,b,tol,nmax)
% Recherche d'un zero dans l'intervalle [a,b] d'une fonction reelle func par la methode de dichotomie.
% En cas de succes, la fonction retourne l'approximation du zero obtenue, le numero de l'iteration a
laquelle cette approximation a ete calculee, la valeur de la fonction func en ce point et un vecteur
contenant la valeur absolue de la difference entre deux approximations successives (increments).
% Si la recherche echoue, un message d'erreur est affiche.
% Note : la fonction func prend en entree un argument reel et retourne un reel en sortie.
fa=feval(func,a); fb=feval(func,b);
if fa*fb>0
   error('Le signe de la fonction doit differer en chaque extremite de l''intervalle.');
elseif fa==0
   zero=a; res=0; iter=0;
   return
elseif fb==0
   zero=b; res=0; iter=0;
   return
end
iter=0;
inc=(a+b)*0.5;
err=(b-a)*0.5;
while (err>=tol&iter<=nmax)
   iter=iter+1;
   x=inc(end);
   fx=feval(func,x);
   if fx*fa>0
      a=x;
      fa=fx;
   elseif fx*fa<0
      b=x;
      fb=fx;
   else
      zero=x; inc(end)=[]; res=0;
      return
   end
   err=0.5*err;
   inc(end+1)=(a+b)*0.5;
   inc(end-1)=abs(inc(end)-x);
end
if iter>nmax
   fprintf('Le nombre maximum d''iterations a ete atteint sans convergence avec la tolerance desiree.
\n');
end
zero=inc(end);
inc(end)=[];
res=feval(func,zero);
return
```

```
function [zero,iter,res,inc]=regulafalsi(func,a,b,tolab,tolf,nmax)
% Recherche d'un zero dans l'intervalle [a,b] d'une fonction reelle func par la methode de la fausse
position.
% En cas de succes, la fonction retourne l'approximation du zero obtenue, le numero de l'iteration a
laquelle cette approximation a ete calculee, la valeur de la fonction func en ce point et un vecteur
contenant la valeur absolue de la difference entre deux approximations successives (increments).
% Si la recherche echoue, un message d'erreur est affiche.
% Note : la fonction func prend en entree un argument reel et retourne un reel en sortie.
fa=feval(func,a); fb=feval(func,b);
if fa*fb>0
   error('Le signe de la fonction doit differer en chaque extremite de l''intervalle.');
elseif fa==0
   zero=a; res=0; iter=0;
   return
elseif fb==0
   zero=b; res=0; iter=0;
   return
end
iter=0;
inc=a-fa*(a-b)/(fa-fb);
fx=feval(func,inc);
res=abs(fx);
while (min(inc(end)-a,b-inc(end))>=tolab&res>=tolf&iter<=nmax)
   iter=iter+1;
   x=inc(end);
   if fx*fa>0
      a=x;
      fa=fx;
   elseif fx*fa<0
      b=x;
      fb=fx;
   else
      zero=x; inc(end)=[]; res=0;
      return
   end
   inc(end+1)=a-fa*(a-b)/(fa-fb);
   inc(end-1)=abs(inc(end)-x);
   fx=feval(func,inc(end));
   res=abs(fx);
end
if iter>nmax
   fprintf('Le nombre maximum d''iterations a ete atteint sans convergence avec la tolerance desiree.
\n');
end
zero=inc(end);
inc(end)=[];
res=feval(func,zero);
return
```

```
function [zero,iter,res,inc]=secante(func,x0,xm1,tol,nmax)
% Recherche d'un zero d'une fonction func par la methode de la secante a partir des points x0 et xm1.
% En cas de succes, la fonction retourne l'approximation du zero obtenue, le numero de l'iteration a
laquelle cette approximation a ete calculee, la valeur de la fonction func en ce point et un vecteur
contenant la valeur absolue de la difference entre deux approximations successives (increments).
% Si la recherche echoue, un message d'erreur est affiche.
% Note : la fonction func prend en entree un argument scalaire et renvoient un scalaire en sortie.
fxm1=feval(func,xm1); fx0=feval(func,x0);
iter=0;
diff=tol+1;
while (abs(diff)>=tol&iter<=nmax)
   iter=iter+1;
   diff=-fx0*(x0-xm1)/(fx0-fxm1);
   x=x0+diff;
   inc(iter)=abs(diff);
   xm1=x0; x0=x;
   fxm1=fx0; fx0=feval(func,x);
end
if iter>nmax
   fprintf(['Le nombre maximum d''iterations a ete atteint sans convergence avec la tolerance desiree.
\n']);
end
zero=x;
res=fx0;
return
```

```
clear all; close all;
% question 2.
disp('Question 2.');
f=inline('exp(x)','x');
I=exp(1)-exp(0);
disp(['La valeur de l''integrale I(f) est ',num2str(I,15),'.']);
disp('Trace des graphes de l''erreur de quadrature en fonction du nombre de sous-intervalles m...');
mmax=20;
for m=1:mmax
   err_mpr(m)=abs(I-pointmilieu_composite(0,1,m,f));
   err_tr(m)=abs(I-trapeze_composite(0,1,m,f));
   err_sr(m)=abs(I-simpson_composite(0,1,m,f));
end
figure;
semilogy(1:mmax,err_mpr,1:mmax,err_tr,1:mmax,err_sr);
legend('erreur pour la formule du point milieu composite','erreur pour la formule du trapeze
composite','erreur pour la formule de Simpson composite');
disp('Appuyer sur une touche pour la question suivante.');
pause
%%
% question 3.
disp('Question 3.');
f=inline('abs(3*x.^4-1)','x');
I=(4*3^(-0.25)-1)*2/5;
disp(['La valeur de l''integrale I(f) est ',num2str(I,15),'.']);
disp('Trace des graphes de l''erreur de quadrature en fonction du nombre de sous-intervalles m...');
mmax=20;
for m=1:mmax
   err_mpr(m)=abs(I-pointmilieu_composite(0,1,m,f));
   err_tr(m)=abs(I-trapeze_composite(0,1,m,f));
   err_sr(m)=abs(I-simpson_composite(0,1,m,f));
end
figure;
semilogy(1:mmax,err_mpr,1:mmax,err_tr,1:mmax,err_sr);
legend('erreur pour la formule du point milieu composite','erreur pour la formule du trapeze
composite','erreur pour la formule de Simpson composite');
disp('On ne retrouve pas les ordres de convergence theoriques car la fonction n''est que continue (sa
derivee premiere presente en effet une discontinuite en 3^(-1/4)). Il convient dans de decomposer
l''intervalle d''integration en deux sous-invervalles, [0,3^(-1/4)] et [3^(-1/4),1], sur lesquels la
fonction est reguliere.');
mmax=20;
for m=1:mmax
   err_mpr(m)=abs(I-(pointmilieu_composite(0,3^(-0.25),m,f)+pointmilieu_composite(3^(-0.25),1,m,f)));
   err_tr(m)=abs(I-(trapeze_composite(0,3^(-0.25),m,f)+trapeze_composite(3^(-0.25),1,m,f)));
   err_sr(m)=abs(I-(simpson_composite(0,3^(-0.25),m,f)+simpson_composite(3^(-0.25),1,m,f)));
end
figure;
semilogy(2*(1:mmax),err_mpr,2*(1:mmax),err_tr,2*(1:mmax),err_sr);
legend('erreur pour la formule du point milieu composite','erreur pour la formule du trapeze
composite','erreur pour la formule de Simpson composite');
```

```
function integrale=pointmilieu_composite(a,b,m,func)
% Calcul d'une approximation de l'integrale de la fonction func sur l'intervalle [a,b] par la formule
de quadrature du point milieu composite
H=(b-a)/m;
x=[a:H/2:b];
y=eval(func);
integrale=H*sum(y(2:2:2*m-1));
return
```

```
function integrale=simpson_composite(a,b,m,func)
% Calcul d'une approximation de l'integrale de la fonction func sur l'intervalle [a,b] par la formule
de quadrature de Simpson composite
H=(b-a)/m;
x=[a:H/2:b];
y=eval(func);
integrale=H/6*(y(1)+2*sum(y(3:2:2*m-1))+4*sum(y(2:2:2*m))+y(2*m+1));
return
```

```
function integrale=trapeze_composite(a,b,m,func)
% Calcul d'une approximation de l'integrale de la fonction func sur l'intervalle [a,b] par la formule
de quadrature du trapeze composite
H=(b-a)/m;
x=[a:H:b];
y=eval(func);
integrale=H/2*(y(1)+2*sum(y(2:m))+y(m+1));
return
```

```
clear all; close all;
% question 1.
disp('Question 1.');
f=inline('1./(1+x.^2)','x');
a=5; b=-5;
x=linspace(a,b,1000);
disp(['Trace des graphes sur l''intervalle [',num2str(a),',',num2str(b),']...']);
figure;
plot(x,f(x),x,polyval(lagrange(f,a,b,2),x),x,polyval(lagrange(f,a,b,4),x),x,polyval(lagrange
(f,a,b,8),x),x,polyval(lagrange(f,a,b,12),x));
legend('f','Pi_2f','Pi_4f','Pi_8f','Pi_{12}f');
disp('Appuyer sur une touche pour la question suivante.');
pause
%%
% question 2.
disp('Question 2.');
disp('Trace du graphe de l''erreur d''interpolation E_n(f) en fonction de n...');
nmax=20;
for n=1:nmax
   e(n)=max(abs(f(x)-polyval(lagrange(f,a,b,n),x)));
end
figure;
semilogy(e);
legend('E_n(f)');
disp('Appuyer sur une touche pour la question suivante.');
pause
%%
% question 3.b.
disp('Question 3.b.');
disp('On interpole a present aux points de Tchebychev.');
disp(['Trace des graphes sur l''intervalle [',num2str(a),',',num2str(b),']...']);
figure;
plot(x,f(x),x,polyval(lagrange_tchebychev(f,a,b,2),x),x,polyval(lagrange_tchebychev
(f,a,b,4),x),x,polyval(lagrange_tchebychev(f,a,b,8),x),x,polyval(lagrange_tchebychev(f,a,b,12),x));
legend('f','Pi_2f','Pi_4f','Pi_8f','Pi_{12}f');
disp('Trace du graphe de l''erreur d''interpolation E_n(f) en fonction de n...');
for n=1:nmax
   e(n)=max(abs(f(x)-polyval(lagrange_tchebychev(f,a,b,n),x)));
end
figure;
semilogy(e);
legend('E_n(f)');
```

```
function p=lagrange(func,a,b,n)
% Construction du polynome d'interpolation de Lagrange de degre n d'une fonction func en n+1 points
equidistribues dans l'intervalle [a,b]
xp=linspace(a,b,n+1);
p=polyfit(xp,feval(func,xp),n);
return
```

```
function p=lagrange_tchebychev(func,a,b,n)
% Construction du polynome d'interpolation de Lagrange de degre n d'une fonction func en n+1 points de
Tchebychev dans l'intervalle [a,b]
xp=(a+b)/2+(b-a)*cos(pi*(2*(1:n+1)-ones(1,n+1))/(2*(n+1)))/2;
p=polyfit(xp,feval(func,xp),n);
return
```

```
f = inline('x/2 - sin(x) + pi/6 - sqrt(3)/2','x');
x = [-pi/2 : 0.1 : pi];
plot(x, f(x))
grid on;
legend('graphe de la fonction f sin[-pi/2, pi]');
```

```
[xi, ita, nb, cv] = dichotomie(f, -pi/2, pi, 1e-10, 1000)
while (cv >= 1 & ita <= nmax)
    ita = ita + 1
    x = (a+b)/2
    fx = feval(fund, x);
    if fx * f(a) > 0
        a = x
        fa = fx
    elseif fx * fa < 0
        b = x
        fb = fx
    else
        zero = x  cv(cv) = [ ]
        nb = 0;
    end
end
er = 0.5 * (b-a)
```

2. Montrer par des développements de Taylor avec reste intégral qu'il existe une fonction h continue sur un voisinage de ξ telle que si $x^{(0)} \in B(\xi, R)$, alors

$$x^{(k+1)} = h(x^{(k)})(x^{(k)} - \xi), \ \forall k \in \mathbb{N} \text{ tel que } x^{(k)} \neq \xi.$$

3. Montrer que la méthode est localement convergente en ξ et que son ordre de convergence est au moins égal à deux.
4. Quel est l'avantage de cette méthode par rapport à celle de Newton–Raphson ?

Exercice 10 ◇ (accélération de la convergence d'une méthode de point fixe par le procédé Δ^2 d'Aitken). Soit I un intervalle fermé de $\mathbb{R}$ et g une fonction de I dans I, deux fois continûment dérivable sur I et admettant un point fixe ξ. On étudie une technique visant à accélérer la convergence de méthodes d'approximations successives du type

$$x^{(k+1)} = g(x^{(k)}), \ k \geq 0, \ x^{(0)} \neq \xi \text{ donné},$$

que l'on suppose convergente avec

$$\|g'\|_\infty = \sup_{x \in I} |g'(x)| < 1.$$

1. On pose $e^{(k)} = x^{(k)} - \xi$, $\forall k \in \mathbb{N}$. Déterminer $\lambda = \lim\limits_{k \to +\infty} \dfrac{e^{(k+1)}}{e^{(k)}}$.
2. On pose ensuite $\lambda^{(k)} = \frac{x^{(k+1)} - x^{(k)}}{x^{(k)} - x^{(k-1)}}$, $\forall k \geq 1$. Montrer que $\lim\limits_{k \to +\infty} \lambda^{(k)} = \lambda$.
3. On introduit la suite auxiliaire du procédé Δ^2 d'Aitken, définie pour tout $k \geq 1$ par

$$z^{(k)} = x^{(k-1)} - \frac{\left(x^{(k)} - x^{(k-1)}\right)^2}{x^{(k+1)} - 2\,x^{(k)} + x^{(k-1)}}.$$

   Vérifier que

$$z^{(k)} = \frac{x^{(k+1)} - \lambda^{(k)} x^{(k)}}{1 - \lambda^{(k)}},$$

   puis exprimer le quotient $\dfrac{z^{(k)} - \xi}{x^{(k)} - \xi}$ en fonction de $e^{(k+1)}$, $e^{(k)}$ et $\lambda^{(k)}$.
4. En déduire que les approximations $z^{(k)}$ convergent plus vite que les approximations $x^{(k)}$, c'est-à-dire que

$$\lim_{k \to +\infty} \frac{z^{(k)} - \xi}{x^{(k)} - \xi} = 0.$$

5. Montrer que l'application du procédé d'Aitken toute les trois itérations à la méthode de point fixe initiale conduit à la construction de la suite $(\overline{x}^{(k)})_{n \in \mathbb{N}}$, définie par

$$\overline{x}^{(k+1)} = \overline{x}^{(k)} - \frac{\left(g(\overline{x}^{(k)}) - \overline{x}^{(k)}\right)^2}{g(g(\overline{x}^{(k)})) - 2\,g(\overline{x}^{(k)}) + \overline{x}^{(k)}} = \psi(\overline{x}^{(k)}),$$

   $\overline{x}^{(0)}$ étant choisi égal à $x^{(0)}$.
6. Prouver que la fonction ψ est dérivable et que $\psi'(\xi) = 0$. En déduire alors que la méthode basée sur le procédé d'Aitken converge quadratiquement dès que $x^{(0)}$ suffisamment proche de ξ. A-t-on besoin de supposer que la méthode de point fixe initiale est convergente ?