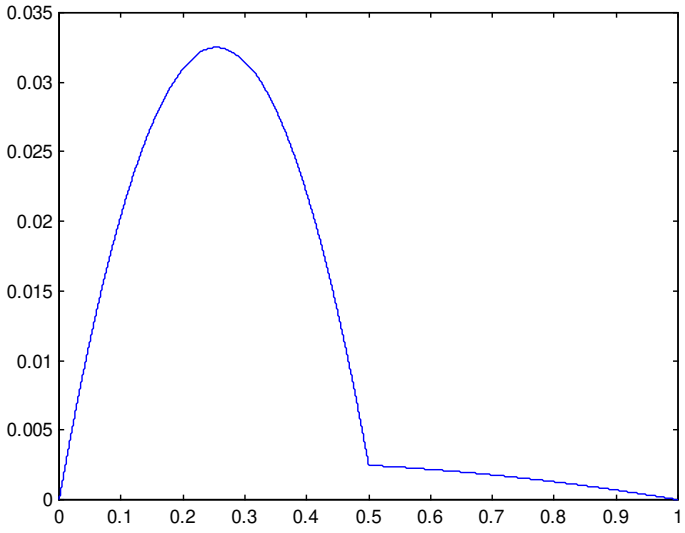
Amandine BROSSE
Blandine CHAUVIN
Emilie MORDACQUE

1A groupe 8
Trinôme 84
Ahiko Assemien

Calcul scientifique – Mini-projet n°3

Résolution d'un problème avec coefficient discontinu

Comparaison des méthodes numériques LPDEM et EFG



11 mai 2005

ENTPE

Introduction

But du projet : comparer plusieurs méthodes numériques pour la résolution d'un problème avec coefficient discontinu

Méthodes numériques étudiées : le schéma LPDEM et le schéma EFG (éléments finis généralisés)

On considère le problème (1) :

$$(1)\ \begin{cases} -(b(x)u')' = 1 \quad si\ 0 < x < 1 \quad (E) \\ u(0) = 0 \\ u(1) = 0 \end{cases} \qquad avec \quad b(x) = \begin{cases} 1 & si\ 0 < x < \frac{1}{2} \\ 100 & si\ \frac{1}{2} < x < 1 \end{cases}$$

I. Résolution du problème par la méthode analytique

Le problème (1) comporte une équation différentielle linéaire (E) d'ordre 2 non résolue avec coefficient discontinu b(x).

On pose $I_1 = \left]0\,;\frac{1}{2}\right[$ et $I_2 = \left]\frac{1}{2}\,;1\right[$.

Résolution de (E) sur I_1

Sur l'intervalle I_1, on a :

(E) $\Leftrightarrow \quad -u'' = 1$

(E) $\Leftrightarrow \quad \exists\,(a_o, b_o) \in \Re^2 \ tel\ que\ \forall x \in I_1,\ u(x) = -\frac{1}{2}x^2 + a_o x + b_o$

Résolution de (E) sur I_2

Sur I_2 on a de même :

(E) $\Leftrightarrow \quad -100u''(x) = 1$

(E) $\Leftrightarrow \quad \exists\,(a_1, b_1) \in \Re^2 \ tel\ que\ \forall x \in I_2,\ u(x) = -\frac{1}{200}x^2 + a_1 x + b_1$

On cherche une solution u $\in$ $C^0([0;1])$ vérifiant les conditions aux limites. On utilise un raisonnement par analyse et synthèse.

Analyse

Supposons qu'il existe une fonction u solution de (1). Alors u est solution de (E) sur I_1 et I_2 donc :

$\exists (a_o, b_o) \in \Re^2$ *tel que* $\forall x \in I_1, u(x) = -\frac{1}{2}x^2 + a_o x + b_o$

et $\exists (a_1, b_1) \in \Re^2$ *tel que* $\forall x \in I_2, u(x) = -\frac{1}{200}x^2 + a_1 x + b_1$

- En écrivant la continuité de u en 0, il vient :
  $u(0) = 0 = \lim_{x \to 0^+} u(x) = b_0$

- Continuité de u en 1 :
  $u(1) = 0 = \lim_{x \to 1^-} u(x) = -\frac{1}{200} + a_1 + b_1 \quad \Leftrightarrow \quad a_1 + b_1 = \frac{1}{200}$

- Continuité de u en ½ :
  $\lim_{x \to \frac{1}{2}^+} u(x) = \lim_{x \to \frac{1}{2}^-} u(x) \quad \Leftrightarrow \quad \frac{a_1}{2} + b_1 - \frac{1}{800} = \frac{a_0}{2} - \frac{1}{8} \quad \Leftrightarrow \quad -\frac{a_0}{2} + \frac{a_1}{2} + b_1 = -\frac{99}{800}$

- Continuité de b*u' en ½ :
  En effet, $\forall x \in]0;1[, (b(x) \times u'(x))' = -1$. Comme –1 est intégrable et ses primitives continues, il vient $\forall x \in]0;1[, b \times u'$ est continue et on a en ½ :
  $\lim_{x \to \frac{1}{2}^+} b(x) \times u'(x) = \lim_{x \to \frac{1}{2}^-} b(x) \times u'(x) \quad \Leftrightarrow \quad -\frac{1}{2} + 100 a_1 = -\frac{1}{2} + a_0 \quad \Leftrightarrow \quad a_0 - 100 a_1 = 0$

On a donc le système :

$$\begin{cases} b_0 = 0 \\ a_1 + b_1 = \frac{1}{200} \\ -\frac{a_0}{2} + \frac{a_1}{2} + b_1 = -\frac{99}{800} \\ a_0 - 100 a_1 = 0 \end{cases}$$

On résout ce système et on obtient :

$$\begin{cases} a_0 = \frac{103}{404} \\ b_0 = 0 \\ a_1 = \frac{103}{40400} \\ b_1 = \frac{99}{40400} \end{cases}$$

Synthèse

Réciproquement on pose : $u : \left| \begin{array}{l} [0;1] \to \Re \\ x \quad \to \begin{cases} -\frac{1}{2}x^2 + \frac{103}{404}x & si\ 0 \le x \le \frac{1}{2} \\ -\frac{1}{200}x^2 + \frac{103}{40400}x + \frac{99}{40400} & si\ \frac{1}{2} < x \le 1 \end{cases} \end{array} \right.$

On a clairement u est solution de (E). Montrons que u est C^0 sur [0;1] et qu'elle vérifie les conditions aux limites.

Sur I_1 et I_2, u est une fonction polynomiale donc C^1. De plus, on a bien :

$$u(0)=0 \text{ et } u(1)=\frac{99}{40400}+\frac{103}{40400}-\frac{1}{200}=0$$

$$\text{et } \lim_{x\to\frac{1}{2}^+} u(x)=-\frac{1}{800}+\frac{103}{80800}+\frac{99}{40400}=\frac{1}{104}=-\frac{1}{8}+\frac{103}{808}=\lim_{x\to\frac{1}{2}^-} u(x)$$

Donc u est continue sur son ensemble de définition.

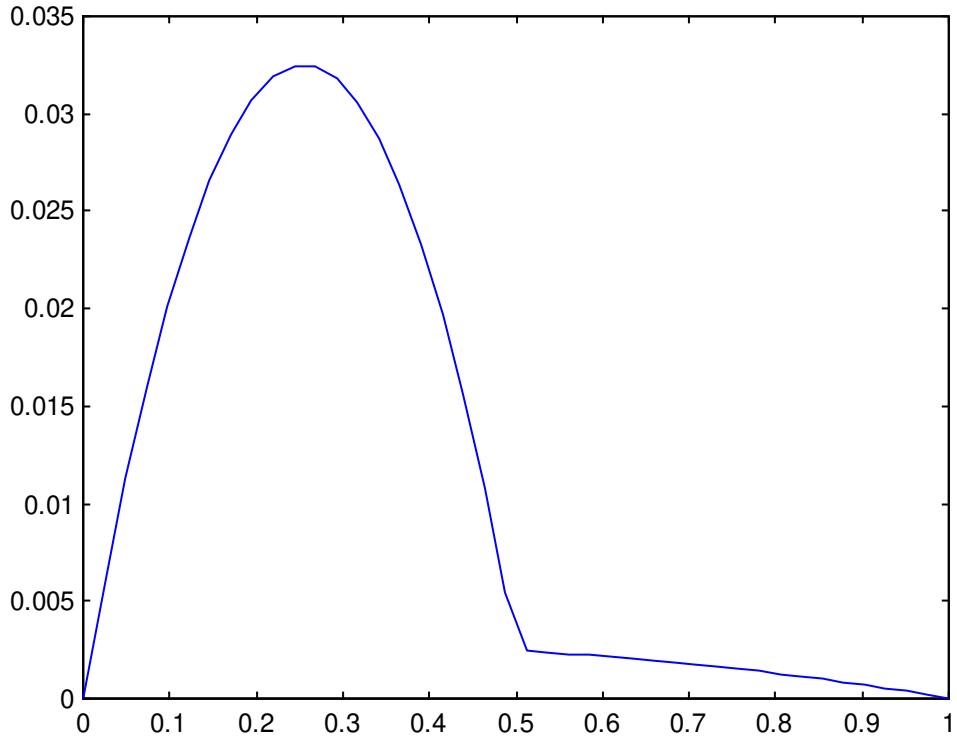
Montrons que b*u' est continue en ½ :

$$\lim_{\frac{1}{2}^-}(b\times u')=-\frac{1}{2}+\frac{103}{404}=-\frac{99}{404} \quad \text{et } \lim_{\frac{1}{2}^+}(b\times u')=-\frac{1}{2}-100\times\frac{103}{40400}=-\frac{99}{404}$$

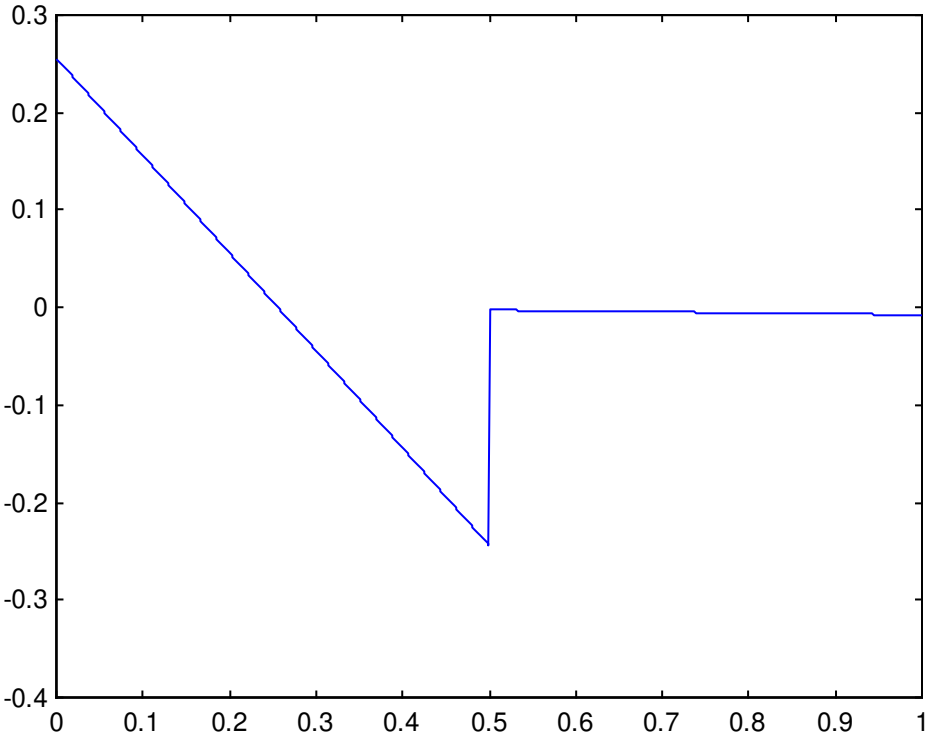
Conclusion

La fonction $$u:\left|\begin{array}{l}[0;1]\to\Re \\ x \quad \to\begin{cases}-\frac{1}{2}x^2+\frac{103}{404}x & si\ 0\le x<\frac{1}{2} \\ -\frac{1}{200}x^2+\frac{103}{40400}x+\frac{99}{40400} & si\ \frac{1}{2}<x\le 1\end{cases}\end{array}\right.$$ est l'unique solution du problème (1).

Tracé de la solution exacte



Tracé de sa dérivée



On remarque la discontinuité en ½.

II. Etablissement du schéma LPDEM

On considère maintenant le problème plus général :

(2) $\begin{cases} \forall x \in I : -(a(x)u')' = f'(x) \\ + \text{conditions aux bornes} \end{cases}$ où u(x) est l'inconnue et a(x) une fonction connue.

Introduction d'une fonction auxiliaire

Posons : $\boxed{\varphi : x \to a(x) * u'(x) + f(x)}$

Or a*u' et f sont dérivables sur l'ensemble I d'après (2) d'où φ est dérivable sur I et :

$\forall x \in I : \varphi'(x) = (a(x)u'(x))' + f'(x) = 0$ d'après (2).

$\Leftrightarrow \forall x \in I : \varphi(x) = K \ , \ K \in \Re$

Nouvelle expression de u' et intégrations

$$Si \ \forall x \in I, a(x) \neq 0 \ alors \ u'(x) = \frac{\varphi(x) - f(x)}{a(x)}$$

Soit h le pas de discrétisation. En intégrant u' sur [x;x+h] et en remplaçant φ, il vient :

$$\int_x^{x+h} u'(t)dt = \int_x^{x+h} \frac{\varphi(t)-f(t)}{a(t)}dt \quad \Leftrightarrow \quad u(x+h)-u(x) = K\int_x^{x+h} \frac{1}{a(t)}dt - \int_x^{x+h} \frac{f(t)}{a(t)}dt$$

En intégrant u' sur [x-h;x] :

$$\int_{x-h}^{x} u'(t)dt = \int_{x-h}^{x} \frac{\varphi(t)-f(t)}{a(t)}dt \quad \Leftrightarrow \quad u(x)-u(x-h) = K\int_{x-h}^{x} \frac{1}{a(t)}dt - \int_{x-h}^{x} \frac{f(t)}{a(t)}dt$$

On exprime K à partir de cette dernière équation :

$$K = \frac{1}{\int_{x-h}^{x} \frac{1}{a(t)}dt}\left(u(x)-u(x-h)+\int_{x-h}^{x} \frac{f(t)}{a(t)}dt\right)$$

On remplace l'expression de K dans la première équation trouvée :

$$u(x+h) = u(x) + \frac{\int_x^{x+h} \frac{1}{a(t)}dt}{\int_{x-h}^{x} \frac{1}{a(t)}dt}\left(u(x)-u(x-h)+\int_{x-h}^{x} \frac{f(t)}{a(t)}dt\right) - \int_x^{x+h} \frac{f(t)}{a(t)}dt$$

$$\Leftrightarrow \quad \left(\int_{x-h}^{x} \frac{1}{a(t)}dt\right)\times u(x+h) = \left(\int_x^{x+h} \frac{1}{a(t)}dt + \int_{x-h}^{x} \frac{1}{a(t)}dt\right)\times u(x) - \left(\int_x^{x+h} \frac{1}{a(t)}dt\right)\times u(x-h)$$

$$+ \int_x^{x+h} \frac{1}{a(t)}dt \times \int_{x-h}^{x} \frac{f(t)}{a(t)}dt - \int_{x-h}^{x} \frac{1}{a(t)}dt \times \int_x^{x+h} \frac{f(t)}{a(t)}dt$$

Schéma numérique

En posant $u(x+h)=u(x_{i+1})=u_{i+1}$, $u(x)=u(x_i)=u_i$ et $u(x-h)=u(x_{i-1})=u_{i-1}$, on obtient le schéma numérique LPDEM :

$$\left(\int_{x_i}^{x_{i+1}} \frac{1}{a(t)}dt\right)\times u_{i-1} - \left(\int_{x_{i-1}}^{x_{i+1}} \frac{1}{a(t)}dt\right)\times u_i + \left(\int_{x_{i-1}}^{x_i} \frac{1}{a(t)}dt\right)\times u_{i+1} = \int_{x_i}^{x_{i+1}} \frac{1}{a(t)}dt \times \int_{x_{i-1}}^{x_i} \frac{f(t)}{a(t)}dt - \int_{x_{i-1}}^{x_i} \frac{1}{a(t)}dt \times \int_{x_i}^{x_{i+1}} \frac{f(t)}{a(t)}dt$$

III. Résolution numérique de l'équation

On résout le problème (1) à l'aide des 2 méthodes :

- la méthode LPDEM avec le schéma trouvé dans la partie précédente
- la méthode des éléments finis généralisés (EFG) dont le schéma est $\forall i \in [1\,..\,N]$:

$$-\left(\frac{1}{\int_{x_{i-1}}^{x_i}\frac{1}{a(t)}dt}\right)\times u_{i-1}+\left(\frac{1}{\int_{x_i}^{x_{i+1}}\frac{1}{a(t)}dt}+\frac{1}{\int_{x_{i-1}}^{x_i}\frac{1}{a(t)}dt}\right)\times u_i-\left(\frac{1}{\int_{x_i}^{x_{i+1}}\frac{1}{a(t)}dt}\right)\times u_{i+1}=\frac{1}{h}\times\left(\int_{x_i}^{x_{i+1}}f(t)dt-\int_{x_{i-1}}^{x_i}f(t)dt\right)$$

Nous avons fait l'essentiel de notre code avec Matlab.

Entrée des données et des schémas

Les deux schémas donnent naissance à un système Ax=B avec x l'inconnue, et dans chaque cas, A est une **matrice tridiagonale**.
Il s'agit dans un premier temps de rentrer A et B, puis de résoudre le système.

Les deux schémas font intervenir des intégrales. Il aurait éventuellement fallu programmer un calcul d'intégrales car Matlab ne sait pas les calculer, contrairement à Maple. Toutefois, l'équation (E) faisant intervenir des constantes, nous nous sommes contentées d'entrer dans Matlab la valeur des coefficients des matrices « à la main » afin de nous concentrer sur la programmation de méthodes de résolution (cf. partie suivante).

Le pas h est considéré toujours impair afin de ne pas avoir à définir de valeurs en ½, point où le coefficient b(x) de l'équation (E) est discontinu.Cela est possible car Matlab fonctionne intrinsèquement avec des matrices et des vecteurs, ce qui est adéquat dans ce type de problème où il s'agit de discrétiser.

Résolution algorithmique du système obtenu Ax=B

Nous n'avons pas utilisé les fonctionnalités de Matlab permettant de déterminer directement x (x=A\B) mais nous avons programmé nos propres méthodes de résolution.

Nous avons programmé deux types de résolution du système linéaire Ax=B : une méthode directe (méthode de Gauss) et une méthode itérative (méthode S.O.R.).

Méthode de Gauss

Pour améliorer la précision des résultats obtenus par la méthode traditionnelle du pivot de Gauss (notamment dans le cas des matrices dont les coefficients sont petits), nous avons opté pour une méthode de Gauss avec pivot maximal. Cette méthode consiste, lors de chaque étape k de l'algorithme, à rechercher parmi les différents pivots possibles celui dont la valeur absolue est la plus grande et à permuter la ligne à laquelle il appartient avec la ligne L_k.

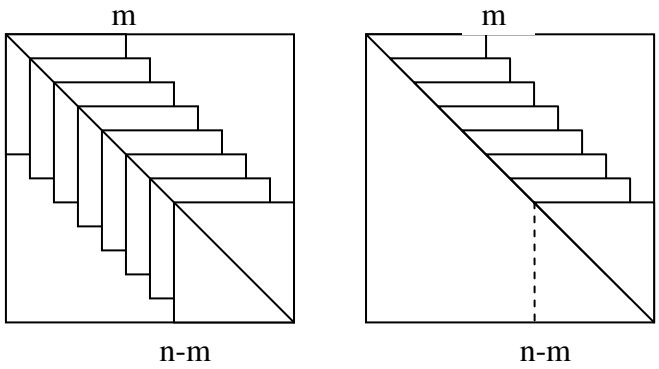
De plus, les matrices obtenues à partir des deux schémas LPDEM et EFG sont des matrices tridiagonales. Pour économiser des calculs inutiles sur les 0 de la matrice A, nous avons programmé une variante de la méthode de Gauss avec pivot maximal pour les matrices bandes (une matrice tridiagonale est une matrice bande de demi largeur 1).

Cependant, si une permutation est effectuée pour obtenir un pivot maximal, la matrice perd sa structure de matrice bande. Nous avons choisi de privilégier la précision des calculs sur le nombre d'opérations effectuées, étant donné que les matrices obtenues n'ont pas une taille trop importante. Tant qu'aucune permutation n'est effectuée, l'algorithme effectue une méthode de Gauss avec pivot maximal sur les coefficients de la bande non nulle. Dès qu'il est nécessaire d'effectuer une permutation pour obtenir le pivot maximal, le programme bascule sur une résolution par la méthode de Gauss avec pivot maximal pour des matrices quelconques.

Nous avons gardé en parallèle des programmes avec pivot de Gauss traditionnel et pivot de Gauss pour les matrices bandes afin de comparer les résultats.

Comparaison du nombre d'opérations entre les fonctions Gauss et GaussBand

Quand on applique le pivot de Gauss à une matrice bande (schéma de gauche), on obtient une matrice triangulaire supérieure "bande" (schéma de droite) :



	Fonction Gauss	Fonction GaussBand
Pivot à l'étape k	n-k divisions (n-k)(n-k+1) additions et multiplications	Si k≤n-m m divisions m(m+1) additions et multiplications Si k>n-m n-k divisions (n-k)(n-k+1) additions et multiplications
Total pivot	n(n-1)/2 divisions n(n²-1)/ 3 additions et multiplications	(n-m)m+m(m-1)/2 divisions (n-m)m(m+1)+m(m²-1)/3 additions et multiplications
Calcul de x_k	1 division n-k additions et multiplications	1 division si k≤n-m m additions et multiplications si k>n-m n-k additions et multiplications
Total X	n divisions n(n-1)/2 additions et multiplications	n divisions (n-m)m+m(m-1)/2 additions et multiplications
TOTAL	n(n+1)/2 divisions n(n+1)(2n+5)/6 additions et multiplications	(m+1)(2n-m)/2 divisions (n-m)m(m+2)+m(m-1)(2m+5)/6 additions et multiplications

Méthode S.O.R.

Pour optimiser la vitesse de convergence de la méthode S.O.R., nous calculons tout d'abord le ω_0 pour lequel cette vitesse de convergence est maximale. Dans le cas des matrices tridiagonales, ω_0 est donné par la formule :

$$\omega_0 = \frac{2}{1+\sqrt{1-\rho(L_1)}} \text{ si } \rho(L_1) \leq 1$$

On applique ensuite la méthode S.O.R. avec le paramètre ω_0 en fixant une valeur pour le critère d'arrêt et pour un vecteur X^0 donné. Comme, d'après les conditions aux limites, x_0^0 et x_N^0 sont nuls, nous avons décidé de prendre X^0 égal au vecteur nul.

De même que précédemment, nous avons gardé en parallèle un programme de la méthode S.O.R. où on peut entrer le paramètre ω au hasard.
Notre programme calcule dans chaque cas le nombre d'itérations effectuées, ce qui va permettre de vérifier la plus grande rapidité de la méthode optimisée.

Calcul des erreurs

Les erreurs sont calculées dans tous les cas en norme infinie discrète absolue.
Nous avons aussi programmé la détermination du point discrétisé en lequel l'erreur est maximale.

Programme de vérification avec Maple

Afin de vérifier que notre programme appliquant le pivot de Gauss nous donnait la bonne solution, nous avons utilisé sur Maple une fonction préexistante qui donne le résultat d'une équation linéaire de type Ax=B.
Pour cela nous avons programmé 2 petites fonctions *lpdem* et *efg* qui à partir des données de :

- n : nombre de points tel que le pas $h = \frac{1}{n+1}$
- a : la fonction a de l'équation (2)
- f : la fonction f de l'équation (2)
- u : la valeur de u(0)
- v : la valeur de u(1)

sortent les matrices A et le vecteur B correspondant respectivement aux schémas LPDEM et EFG.
Ces 2 programmes calculent les coefficients non nuls de la matrice (qui sont des intégrales) en évaluant numériquement (fonction *evalf ( )*) des intégrales théoriques (fonction *Int( )*).
Nous avons ainsi pu vérifier que ces programmes permettaient d'obtenir les mêmes matrices que le programme élaboré sur Matlab.

IV Résultats

Nous avons volontairement laissé un grand nombre de chiffres après la virgule pour observer les différences entre les méthodes de résolution.

's' est le critère d'arrêt de la méthode S.O.R.
'k' est le nombre d'itérations de la méthode S.O.R.

Nous rapportons dans ce document quelques résultats afin de comparer la méthode S.O.R. simple et la méthode S.O.R. optimisée et d'étudier l'influence du critère d'arrêt. Nous nous contentons de regarder ce que se passe pour h = 1/5 avec s = 0,01 puis $s = 10^{-14}$.
Nous prenons ensuite $s = 10^{-14}$ pour la méthode optimisée afin de pouvoir comparer les résultats. En effet, le temps de calcul de l'ordinateur reste raisonnable pour le pas le plus fin dans ce cas.

Comparaison des deux schémas numériques (avec la méthode de Gauss)

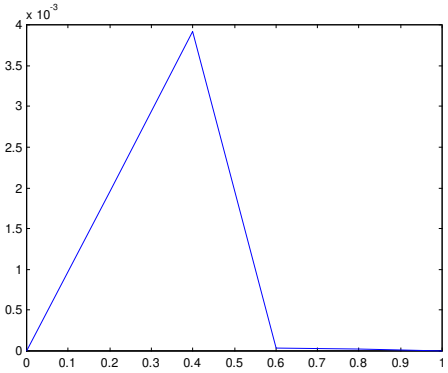
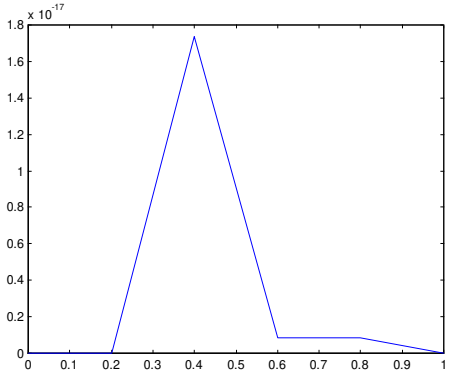
	EFG	LPDEM
h = 1/5	3.920792079207924e-003	1.734723475976807e-017
h = 1/21	2.646037198471979e-004	6.938893903907228e-018
h = 1/41	7.111025810724835e-005	3.469446951953614e-017
h = 1/81	1.844415503887698e-005	3.469446951953614e-017
h = 1/101	1.189213176299494e-005	5.551115123125783e-017
h = 1/151	5.338069833534919e-006	2.498001805406602e-016

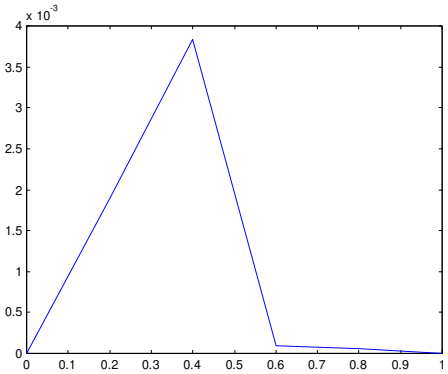
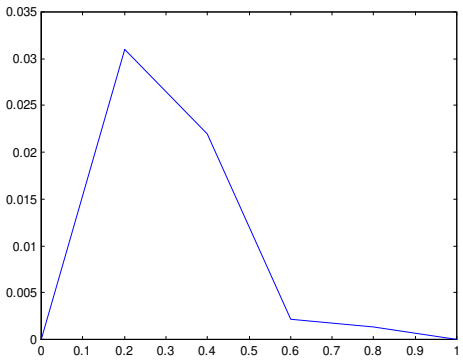
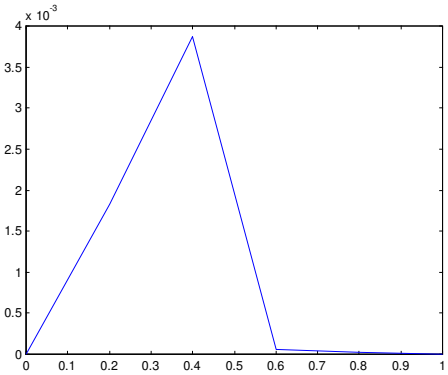
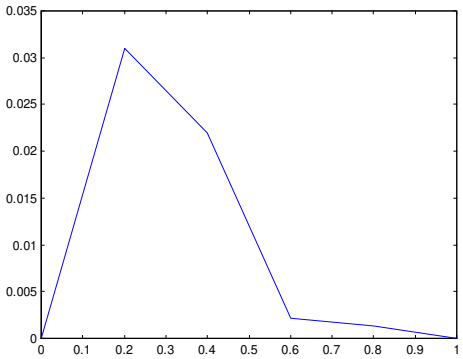
soit encore

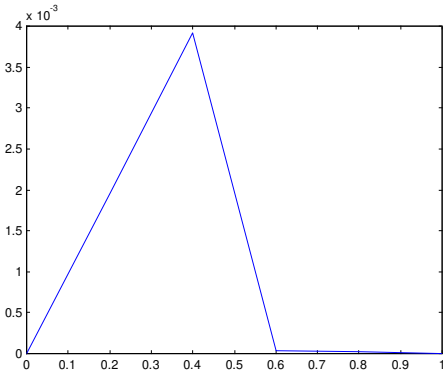
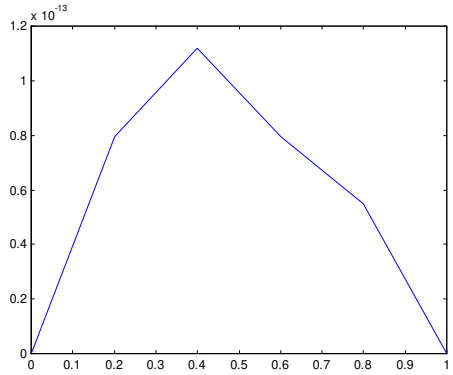
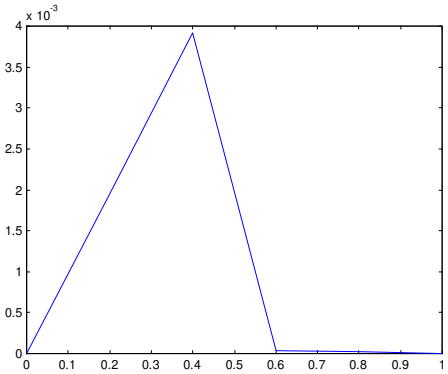
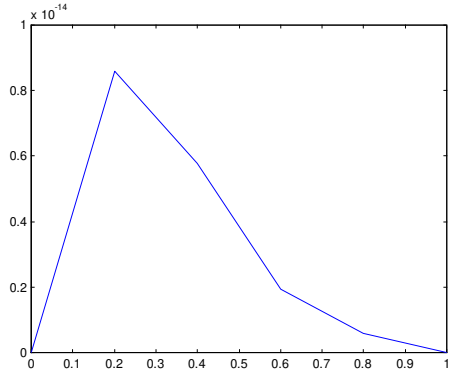
	EFG	**LPDEM**
h = 1/5	**3.92e-003**	**1.73e-017**
h = 1/21	**2.64e-004**	**6.94e-018**
h = 1/41	**7.11e-005**	**3.47e-017**
h = 1/81	**1.84e-005**	**3.47e-017**
h = 1/101	**1.194e-005**	**5.55e-017**
h = 1/151	**5.34e-006**	**2.50e-016**

Nous traitons ci-après les résultats détaillés.
Les graphiques représentent l'erreur absolue dans chaque cas.

h = 1/5

	EFG	LPDEM
Méthodes de Gauss		
max	x(2h)	x(2h)

SOR ω=0,1 s=0,01		
max	x(2h)	x(h)
SOR2 ω_0 s=0,01		
ω_0	1.176236931387601e+000	1.176236931387601e+000
max	x(2h)	x(h)

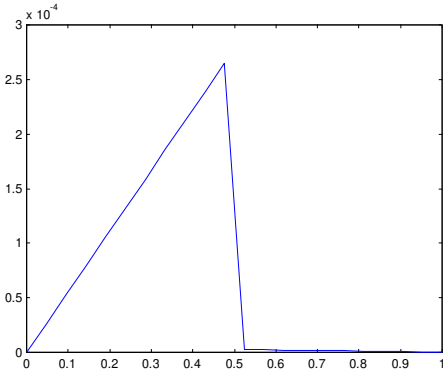
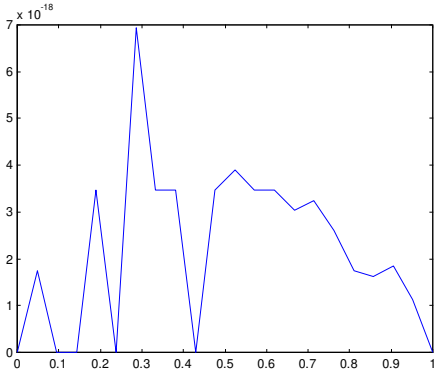
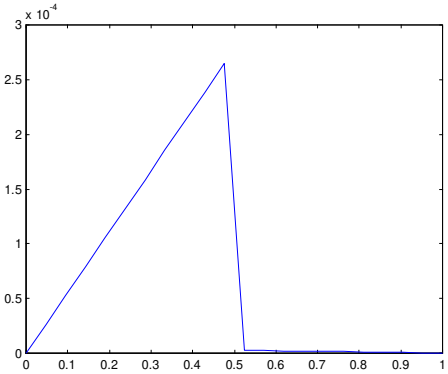
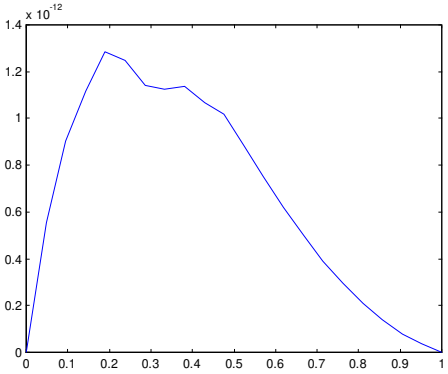
SOR ω=0,1 $s=10^{-14}$		
max	x(2h)	x(2h)
SOR2 ω_0 $s=10^{-14}$		
ω_0	1.176236931387601e+000	1.176236931387601e+000
max	x(2h)	x(h)

Méthode	EFG	LPDEM
Gauss	3.920792079207924e-003	1.734723475976807e-017
GaussMax	3.920792079207924e-003	1.734723475976807e-017
GaussBand	3.920792079207924e-003	1.734723475976807e-017
GaussMaxBand	3.920792079207924e-003	1.734723475976807e-017

s=0,01	SOR (ω=0,1)	3.841283066797235e-003	3.099009900990099e-002
	k_SOR	131	0
	SOR2 (ω_0)	3.869884591315424e-003	3.099009900990099e-002
	k_SOR2	4	0

$s=10^{-14}$	SOR (ω=0,1)	3.920792079207827e-003	1.119070114352638e-013
	k_SOR	1045	804
	SOR2 (ω_0)	3.920792079207886e-003	8.583411759133242e-015
	k_SOR2	21	18

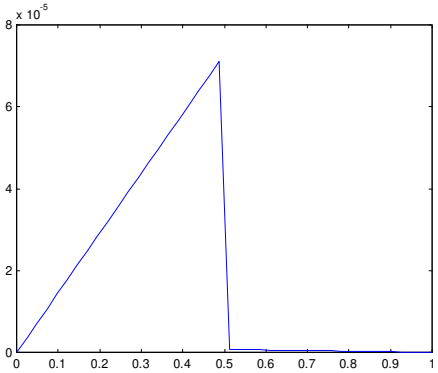
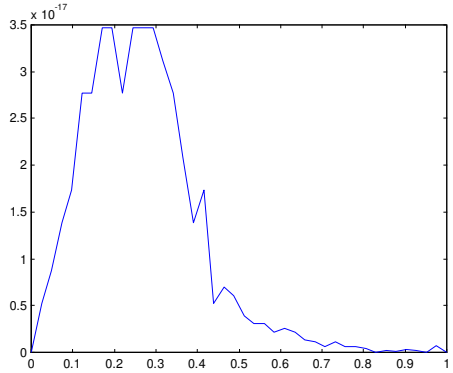
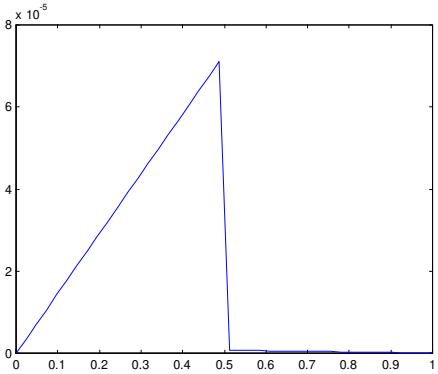
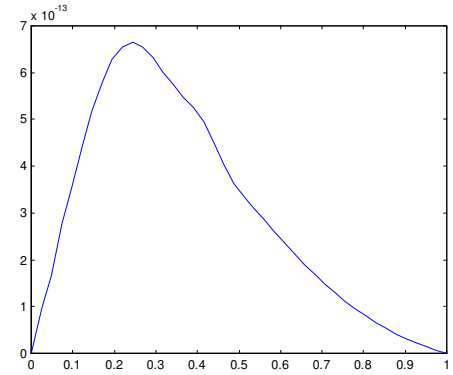
h = 1/21

	EFG	LPDEM
Méthodes de Gauss		
max	x(10h)	x(6h)
SOR2 ω_0 $s=10^{-14}$		
ω_0	1.729911594104093e+000	1.729911594104092e+000
max	x(10h)	x(4h)

Méthode	EFG	LPDEM
Gauss	2.646037198471979e-004	6.938893903907228e-018
GaussMax	2.646037198471979e-004	6.938893903907228e-018
GaussBand	2.646037198471979e-004	6.938893903907228e-018
GaussMaxBand	2.646037198471979e-004	6.938893903907228e-018

$s=10^{-14}$	SOR2 (ω_0)	2.646037198471649e-004	1.286162149005676e-012
	k_SOR2	114	80

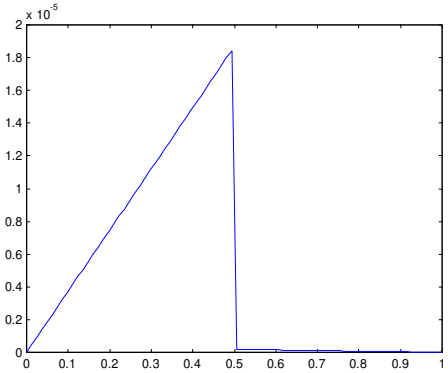
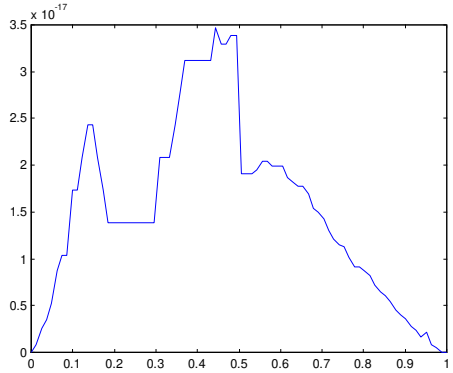
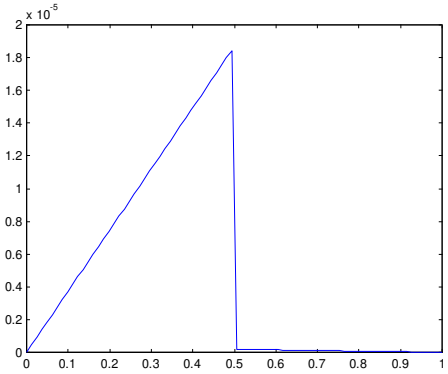
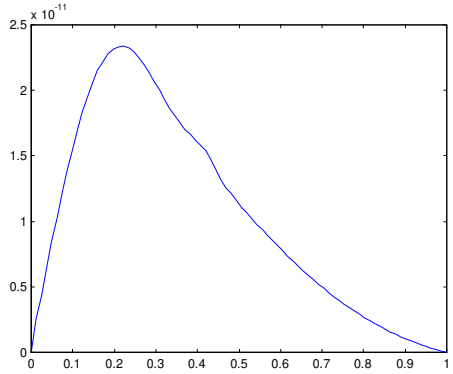
h = 1/41

	EFG	LPDEM
Méthodes de Gauss		
max	x(20h)	x(7h)
SOR2 ω_0 $s=10^{-14}$		
ω_0	1.854629910831040e+000	1.854629910831008e+000
max	x(20h)	x(10h)

Méthode	EFG	LPDEM
Gauss	7.111025810724835e-005	3.469446951953614e-017
GaussMax	7.111025810724835e-005	3.469446951953614e-017
GaussBand	7.111025810724835e-005	3.469446951953614e-017
GaussMaxBand	7.111025810724835e-005	3.469446951953614e-017

$s=10^{-14}$	SOR2 (ω_0)	7.111025810718763e-005	5.225903043637459e-012
	k_SOR2	229	152

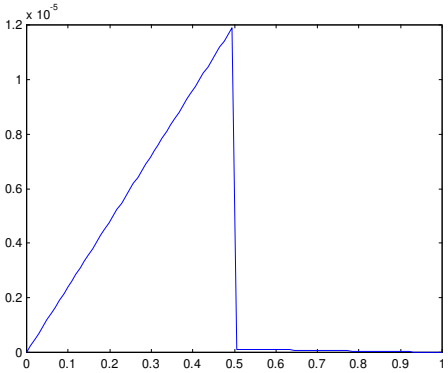
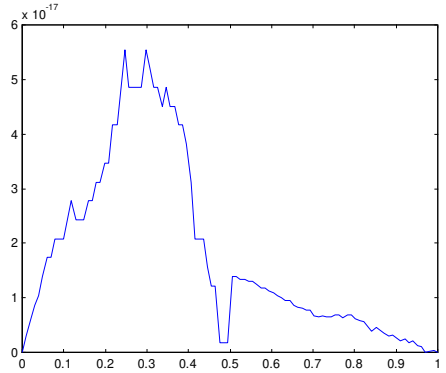
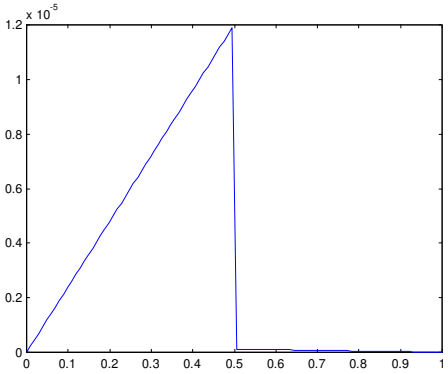
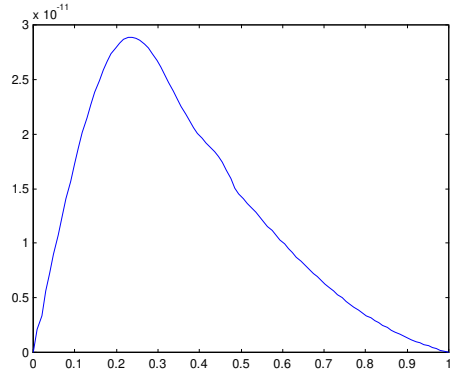
h = 1/81

	EFG	LPDEM
Méthodes de Gauss		
max	x(40h)	x(36h)
SOR2 ω_0 $s=10^{-14}$		
ω_0	1.924482197803286e+000	1.924482197803286e+000
max	x(40h)	x(18h)

Méthode	EFG	LPDEM
Gauss	1.844415503887698e-005	3.469446951953614e-017
GaussMax	1.844415503887698e-005	3.469446951953614e-017
GaussBand	1.844415503887698e-005	3.469446951953614e-017
GaussMaxBand	1.844415503887698e-005	3.469446951953614e-017

$s=10^{-14}$	SOR2 (ω_0)	1.844415503888218e-005	2.335853732660098e-011
	k_SOR2	465	288

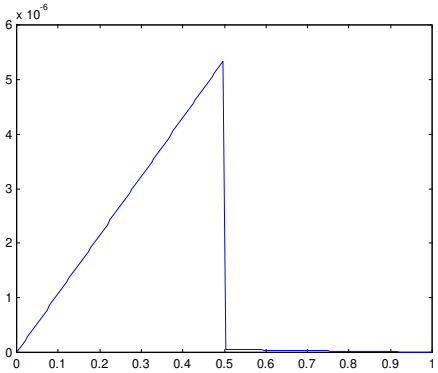
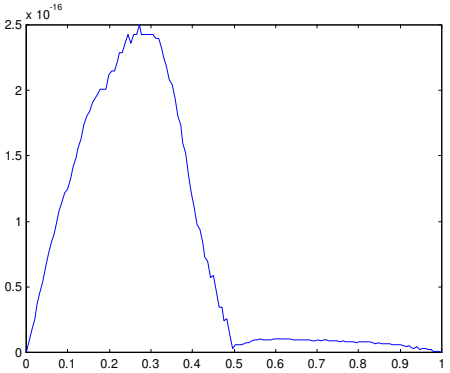
h = 1/101

	EFG	LPDEM
Méthodes de Gauss		
max	x(50h)	x(25h)
SOR2 ω_0 $s=10^{-14}$		
ω_0	1.939114805972248e+000	1.939114805972199e+000
max	x(50h)	x(24h)

Méthode	EFG	LPDEM
Gauss	1.189213176299494e-005	5.551115123125783e-017
GaussMax	1.189213176299494e-005	5.551115123125783e-017
GaussBand	1.189213176299494e-005	5.551115123125783e-017
GaussMaxBand	1.189213176299494e-005	5.551115123125783e-017

$s=10^{-14}$	SOR2 (ω_0)	1.189213176303397e-005	2.888222994101852e-011
	k_SOR2	570	356

h = 1/151

	EFG	LPDEM
Méthodes de Gauss		
max	x(75h)	x(41h)

Méthode	EFG	LPDEM
Gauss	5.338069833534919e-006	2.498001805406602e-016
GaussMax	5.338069833534919e-006	2.498001805406602e-016
GaussBand	5.338069833534919e-006	2.498001805406602e-016
GaussMaxBand	5.338069833534919e-006	2.498001805406602e-016

V Exploitation des résultats

Comparaison des schémas numériques

L'erreur en norme infinie discrète absolue commise par le schéma LPDEM est beaucoup plus faible que celle commise par le schéma EFG. **Le schéma LPDEM est beaucoup plus performant que le schéma EFG**.

Pour le schéma EFG, l'erreur semble être une fonction affine par morceaux avec une discontinuité en ½.

Pour le schéma LPDEM, la courbe de l'erreur obtenue avec la méthode SOR semble avoir une allure similaire à celle de la solution exacte : l'erreur est plus importante aux points où la solution exacte est plus grande. Avec la méthode de Gauss, la courbe d'erreur se rapproche proportionnellement de la courbe de la solution exacte, ce quand le pas devient de plus en plus petit.

Comparaison des méthodes de résolution

Méthodes de Gauss

On observe que les différentes méthodes de Gauss donnent le même résultat. Il s'avère en effet que les méthodes avec pivot maximal ne nécessitent aucune permutation, ainsi que nous avons pu le vérifier en introduisant un test dans la fonction GaussMaxBand. Dans notre cas, elles sont donc **équivalentes**.

En outre, on observe que l'erreur obtenue avec la méthode de Gauss ne diminue pas quand le pas diminue, contrairement aux attentes. Si elle est meilleure pour h = 1/21 que pour h = 1/5, elle est de moins en moins bonne ensuite lorsque le pas diminue. On suppose que ce résultat est lié à la méthode de résolution en elle-même.

Méthodes S.O.R.

On vérifie bien qu'en choisissant le ω_0, le nombre d'itérations est minimum : la vitesse de convergence maximale.
D'autre part, on vérifie que pour un même critère d'arrêt, le nombre d'itérations augmente quand le pas diminue.
Enfin, le nombre d'itérations est plus important pour le schéma EFG que pour le schéma LPDEM, ce qui semble cohérent avec la plus grande performance de ce second schéma.

Méthode de Gauss / méthode S.O.R.

Pour la SOR, la qualité du résultat dépend du critère d'arrêt.
C'est pourquoi la méthode S.O.R. est plus performante que la méthode de Gauss tant que le critère d'arrêt est assez faible. Pour le schéma EFG, les erreurs restant de l'ordre de 10^{-3} à 10^{-5}, le critère d'arrêt de 10^{-14} s'avère concluant. Tandis que pour le schéma LPDEM, les erreurs sont de l'ordre de 10^{-17}, donc le critère d'arrêt de 10^{-14} n'est pas pertinent. Toutefois, le temps de calcul pour un critère d'arrêt inférieur à 10^{-17} est beaucoup trop long.

VI Annexe : code

Matlab

Fonction ‘solution_exacte’

But : déterminer le vecteur correspondant à la solution_exacte, fonction du pas h

```
% Calcul de la solution exacte
% Paramètres d'entrée : valeur du pas h, indice de début i_0, indice de fin
i_N, indice de milieu i_milieu, valeurs aux limites u_debut et u_fin
% Paramètre de sortie : vecteur ligne u_exacte


function u_exacte=solution_exacte(h,i_0,i_milieu,i_N,u_debut,u_fin)


% Sur [0;1/2[, u(x)=(103/404-x/2)*x
% Sur [1/2;0[, u(x)=-x^2/200+103*x/40400+99/40400

u_exacte(i_0)=u_debut;
u_exacte(i_N)=u_fin;

for i=(i_0+1):i_milieu
   x=(i-1)*h;
   u_exacte(i)=(103/404-x/2)*x;
end

for i=(i_milieu+1):(i_N-1)
   x=(i-1)*h;
   u_exacte(i)=-x^2/200+103*x/40400+99/40400;
end
```

Fonction ‘coef_lpdem’

But : déterminer les coefficients c_lpdem et d_lpdem du schéma LPDEM qui interviennent dans la matrice A_lpdem et le second membre B_lpdem correspondants

```
% Calcul des coefficients dans le cas de l'énoncé pour le schéma LPDEM


function [c_lpdem,d_lpdem]=coef_lpdem(i_0,i_milieu,i_N,h,b)


% Définition des coefficients c dépendant de b : c(i)=int(dt/b(t),x(i),x(i+1))

for i=i_0:(i_milieu-1)
   c_lpdem(i)=h/b(i);
end
for i=(i_milieu+1):(i_N-1)
   c_lpdem(i)=h/b(i);
end

c_lpdem(i_milieu)=(b(i_milieu+1)+b(i_milieu))/2/b(i_milieu+1)*h;


% Définition des coefficients dépendant de f : e(i)=int(tdt/b(t),x(i),x(i+1))
```

```
for i=i_0:(i_milieu-1)
   e_lpdem(i)=((i-1)*h^2+h^2/2)/b(i);
end
for i=(i_milieu+1):(i_N-1)
   e_lpdem(i)=((i-1)*h^2+h^2/2)/b(i);
end

e_lpdem(i_milieu)=(99-396*(i_milieu-1)^2*h^2+4*h^2+8*(i_milieu-1)*h^2)/800;


% Définition du second membre d(i)=c(i)*e(i-1)-c(i-1)*e(i)

for i=(i_0+1):(i_N-1)
   d_lpdem(i)=c_lpdem(i)*e_lpdem(i-1)-c_lpdem(i-1)*e_lpdem(i);
end
```

Fonction ‘coef_efg’

But : déterminer les coefficients c_efg et d_efg du schéma EFG qui interviennent dans la matrice A_efg et le second membre B_efg correspondants

```
% Calcul des coefficients dans le cas de l'énoncé pour le schéma EFG


function [c_efg,d_efg]=coef_efg(i_0,i_milieu,i_N,h,b)


% Définition des coefficients c dépendant de b :
c(i)=1/int(dt/b(t),x(i),x(i+1))

for i=i_0:(i_milieu-1)
   c_efg(i)=b(i)/h;
end
for i=(i_milieu+1):(i_N-1)
   c_efg(i)=b(i)/h;
end

c_efg(i_milieu)=2*b(i_milieu+1)/(b(i_milieu+1)+b(i_milieu))/h;


% Définition du second membre dépendant de f : 1/h[int(f(t)dt,x(i),x(i+1))-
int(f(t)dt,x(i_1),x(i))]

for i=(i_0+1):(i_N-1)
   d_efg(i)=h;
end
```

Fonction ‘matrice_lpdem’

But : remplir la matrice A_lpdem et le vecteur B_lpdem à l’aide des coefficients c_lpdem et d_lpdem déterminés précédemment

```
% Le but de cette fonction est de créer une matrice à partir d'un schéma
numérique.

% On s'intéresse ici au schéma LPDEM.
% paramètres d'entrée :
   % l'intervalle de travail ie un début et une fin : debut et fin
   % le pas h
   % les coefficients qui dépendront de la fonction a(t) : b
```

```
   % la valeur  du second membre qui dépendra de f(t) : d
   % et aussi des valeurs particulières de u : u_debut et u_fin
% paramètres de sortie : matrice A_lpdem et vecteur ligne B_lpdem


function [A_lpdem,B_lpdem]=matrice_lpdem(debut,fin,h,u_debut,u_fin,c,d)

nombre_de_lignes=(fin-debut)/h+1;
nombre_de_colonnes=(fin-debut)/h+1;

A_lpdem=zeros(nombre_de_lignes,nombre_de_colonnes);
B_lpdem=zeros(nombre_de_lignes,1);

A_lpdem(1,1)=1;
B_lpdem(1,1)=u_debut;

A_lpdem(nombre_de_lignes,nombre_de_colonnes)=1;
B_lpdem(nombre_de_lignes,1)=u_fin;

for i=2:(nombre_de_lignes-1)
   A_lpdem(i,i-1)=c(i);
   A_lpdem(i,i)=-c(i)-c(i-1);
   A_lpdem(i,i+1)=c(i-1);
   B_lpdem(i)=d(i);
end
```

Fonction 'matrice_efg'

But : remplir la matrice A_efg et le vecteur B_efg à l'aide des coefficients c_efg et d_efg déterminés précédemment

```
% Le but de cette fonction est de créer une matrice à partir d'un schéma
numérique.

% On s'intéresse ici au schéma 'éléments finis généralisés'.
% paramètres d'entrée :
   % l'intervalle de travail ie un début et une fin : debut et fin
   % le pas h
   % les coefficients qui dépendront de la fonction a(t) : b
   % la valeur  du second membre qui dépendra de f(t) : d
   % et aussi des valeurs particulières de u : u_debut et u_fin
% paramètres de sortie : matrice A_efg et vecteur ligne B_efg


function [A_efg,B_efg]=matrice_efg(debut,fin,h,u_debut,u_fin,c,d)

nombre_de_lignes=(fin-debut)/h+1;
nombre_de_colonnes=(fin-debut)/h+1;

A_efg=zeros(nombre_de_lignes,nombre_de_colonnes);
B_efg=zeros(nombre_de_lignes,1);

A_efg(1,1)=1;
B_efg(1,1)=u_debut;

A_efg(nombre_de_lignes,nombre_de_colonnes)=1;
B_efg(nombre_de_lignes,1)=u_fin;

for i=2:(nombre_de_lignes-1)
   A_efg(i,i-1)=-c(i-1);
   A_efg(i,i)=c(i-1)+c(i);
   A_efg(i,i+1)=-c(i);
```

```
   B_efg(i)=d(i);
end
```

Fonction 'Gauss'

But : calculer la solution x du système Ax=B par la méthode de Gauss simple

```
%Méthode de Gauss
%Arguments d'entrée:matrice A et vecteur B
%Argument de sortie:vecteur solution x

function[X]=Gauss(A,B)

n=size(A);
n=n(1);
AG=A;
BG=B;

%Passage de la matrice A à une matrice triangulaire supérieure par
%combinaisons linéaires des lignes
for k=1:1:(n-1)
    for i=(k+1):1:n
        r=AG(i,k)/AG(k,k);
        for j=(k+1):1:n
            AG(i,j)=AG(i,j)-r*AG(k,j);
        end
        BG(i)=BG(i)-r*BG(k);
    end
end

%Calcul des coefficients du vecteur solution
 X(n)=BG(n)/AG(n,n);

 for q=1:1:(n-1)
     s=BG(n-q);
     for j=(n-q+1):1:n
         s=s-AG(n-q,j)*X(j);
         X(n-q)=s/AG(n-q,n-q);
     end
 end
```

Fonction 'GaussMax'

But : calculer la solution x du système Ax=B par la méthode de Gauss avec pivot maximal

```
%Méthode de Gauss avec pivot maximal
%Arguments d'entrée:matrice A et vecteur B
%Argument de sortie:vecteur solution x

function[X]=GaussMax(A,B)

n=size(A);
n=n(1);

AG=A;
BG=B;

%Passage de la matrice A à une matrice triangulaire supérieure
for k=1:1:(n-1)

    %Recherche du pivot maximal
```

```
    p=k;
    for i=(k+1):1:n
        if abs(A(k,i))>abs(A(k,p))
            p=i;
        end
    end
    %Permutation de la ligne k et de la ligne contenant le pivot maximal
    aux=AG(k,:);
    AG(k,:)=AG(p,:);
    AG(p,:)=aux;

    %Application du pivot
    for i=(k+1):1:n
        r=AG(i,k)/AG(k,k);
        for j=(k+1):1:n
            AG(i,j)=AG(i,j)-r*AG(k,j);
        end
        BG(i)=BG(i)-r*BG(k);

    end

end


%Calcul des coefficients du vecteur solution X en remontant le système
 X(n)=BG(n)/AG(n,n);

 for q=1:1:(n-1)
     s=BG(n-q);
     for j=(n-q+1):1:n
         s=s-AG(n-q,j)*X(j);
         X(n-q)=s/AG(n-q,n-q);
     end
 end
```

Fonction 'GaussBand'

But : calculer la solution x du système Ax=B par la méthode de Gauss en prenant en compte le caractère matrice bande

```
function[X]=GaussBand(A,B,m)

n=size(A);
n=n(1);

AG=A;
BG=B;
%Passage de la matrice A à une matrice triangulaire supérieure
%Les calculs ne sont effectués que sur les coefficients de la bande non nulle
for k=1:1:(n-m)

    for i=(k+1):1:(k+m)
        r=AG(i,k)/AG(k,k);
        for j=(k+1):1:(k+m)
            AG(i,j)=AG(i,j)-r*AG(k,j);
        end
        BG(i)=BG(i)-r*BG(k);

    end
end

for k=(n-m+1):1:(n-1)
```

```
    for i=(k+1):1:n
        r=AG(i,k)/AG(k,k);
        for j=(k+1):1:n
            AG(i,j)=AG(i,j)-r*AG(k,j);
            BG(i)=BG(i)-r*BG(k);
        end
    end
end



%Remontée:calcul des coefficients du vecteur solution X
 X(n)=BG(n)/AG(n,n);

for q=1:1:(m-1)
     s=BG(n-q);
     for j=(n-q+1):1:n
         s=s-AG(n-q,j)*X(j);
         X(n-q)=s/AG(n-q,n-q);
     end
end
for q=m:1:(n-1)
         s=BG(n-q);
         for j=(n-q+1):1:(n-q+m)
             s=s-AG(n-q,j)*X(j);
             X(n-q)=s/AG(n-q,n-q);
         end
end
```

Fonction 'GaussMaxBand'

But : calculer la solution x du système Ax=B par la méthode de Gauss avec pivot maximal en prenant en compte le caractère matrice bande

```
%Méthode de Gauss avec pivot maximal pour matrice bande
%Résolution du système AX=B
%Arguments d'entrée: matrice-bande A
%                    demi-largeur de A m
%                    vecteur B
%Argument de sortie:vecteur solution X

function[X,bool]=GaussBand(A,B,m)

n=size(A);
n=n(1);

AG=A;
BG=B;
%La variable booléenne bool indique si la matrice a une structure bande
bool=1;
%La variable k indique l'étape de la méthode de Gauss
k=1;

%Recherche du pivot maximal dans le cas d'une matrice bande
while k<(n-m+1)
    p=k;
    u=k+1;
    while u<n
        %Si le pivot initial n'est pas le pivot maximal,
        %il faut effectuer une permutation
        %la matrice n'a donc plus une structure de matrice-bande
```

```
        %on sort de la boucle
        if abs(AG(k,u))>abs(AG(k,p))
            p=u;
            bool=0;
            break;
        end
        if bool==0
            break
        else
            u=u+1;
        end
    end
    if bool==0
        break
    end

    %Application du pivot à l'étape k pour une matrice-bande

    for i=(k+1):1:(k+m)
        r=AG(i,k)/AG(k,k);
        for j=(k+1):1:(k+m)
            AG(i,j)=AG(i,j)-r*AG(k,j);
        end
        BG(i)=BG(i)-r*BG(k);
    end
    k=k+1;
end


%Pivot maximal de l'étape k à n-1 pour une matrice quelconque
if bool==0|k>(n-m)
    while k<n
        %Recherche du pivot maximal
        p=k;
        for i=(k+1):1:n
            if abs(AG(k,i))>abs(AG(k,p))
                p=i;
            end
        end
        %Permutation de la ligne k et de celle contenant le pivot maximal
        aux=AG(k,:);
        AG(k,:)=AG(p,:);
        AG(p,:)=aux;
        %Application du pivot à l'étape k
        for i=(k+1):1:n
            r=AG(i,k)/AG(k,k);
            for j=(k+1):1:n
                AG(i,j)=AG(i,j)-r*AG(k,j);
            end
            BG(i)=BG(i)-r*BG(k);
        end
        k=k+1;
    end

end


%Remontée:calcul des coefficients du vecteur solution X
 X(n)=BG(n)/AG(n,n);

 if bool
     %Dans le cas où il n'y a pas eu de permutation
     for q=1:1:(m-1)
          s=BG(n-q);
```

```
          for j=(n-q+1):1:n
              s=s-AG(n-q,j)*X(j);
              X(n-q)=s/AG(n-q,n-q);
          end
     end
     for q=m:1:(n-1)
              s=BG(n-q);
              for j=(n-q+1):1:(n-q+m)
                  s=s-AG(n-q,j)*X(j);
                  X(n-q)=s/AG(n-q,n-q);
              end
     end
 else
     %Dans le cas où il y a eu une (ou plusieurs) permutation(s)
     for q=1:1:(n-1)
         s=BG(n-q);
         for j=(n-q+1):1:n
             s=s-AG(n-q,j)*X(j);
             X(n-q)=s/AG(n-q,n-q);
         end
     end
 end
```

Fonction 'SOR'

But : calculer la solution x du système Ax=B par la méthode S.O.R. simple où le paramètre ω est choisi entre 0 et 2

```
%Méthode S.O.R
%Arguments d'entrée:matrice A
%                   vecteur B
%                   paramètre w
%                   critère d'arrêt s
%                   vecteur Xo
%Arguments de sortie: vecteur solution X
%                     nombre d'itérations k

function[X,k]=SOR(A,B,w,s,Xo)


X=Xo;
n=size(A);
n=n(1);
%Initialisation du critère
h=max(abs(A*Xo-B));
%Initialisation du compteur
k=0;

while h>s   %Poursuite des itérations tant que le critère d'arrêt n'est pas
vérifié
%Calcul des composantes du vecteur Xk
for i=1:1:n
        p=0;
        q=0;
        for j=1:1:i-1
            p=p+A(i,j)/A(i,i)*X(j);
        end
        for j=i+1:1:n
            q=q+A(i,j)/A(i,i)*X(j);
        end
        X(i)=(1-w)*X(i)-w*p-w*q+w*B(i)/A(i,i);
end
```

```
%Incrémentation du compteur
k=k+1;
%Calcul de la valeur du critère
    h=max(abs(A*X-B));
end
```

Fonction 'RaySpect'

But : calculer le rayon spectral d'une matrice

```
%Calcul du rayon spectral
%Argument d'entrée:matrice A
%Argument de sortie:rayon spectral r

function[r]=RaySpect(A)
%Calcul des valeurs propres de A
v=eig(A);
%Calcul du module des valeurs propres
v=abs(v);
%Recherche du maximum des modules des valeurs propres
r=max(v);
```

Fonction 'LSOR'

But : calculer la matrice d'itération de la méthode S.O.R.

```
%Calcul de la matrice d'itération de la méthode SOR
%Arguments d'entrée: matrice A
%                    paramètre w de la méthode SOR
%Argument de sortie: matrice d'itération de la méthode SOR Lw

function[Lw]=LSOR(A,w)
D=diag(diag(A));
E=- triu(A,1);
F=- tril(A,-1);
M=(1/w).*D-E;
N=(1/w-1).*D+F;
n=size(A);
n=n(1);
I=eye(n);
G=I-w.*D^(-1)*E;
H=(1-w).*I+w.*D^(-1)*F;
Lw=G^(-1)*H;
```

Fonction 'w0'

But : calculer la valeur ω_0 du paramètre de la méthode S.O.R. assurant une convergence optimale dans le cas d'une matrice tridiagonale

```
%Calcul de w0, valeur du paramètre de la méthode SOR assurant une
%convergence optimale dans le cas d'une matrice tridiagonale
%Argument d'entrée:matrice tridiagonale A
%Argument de sortie:paramètre w0

function[w]=w0(A)
%Calcul de la matrice L1
L1=LSOR(A,1);
%Calcul du rayon spectral de L1
x=RaySpect(L1);
%Calcul de w0 dans la cas où 1>=x (domaine de validité de la formule)
```

```
if x>1
    'Erreur: Le rayon spectral de L1 est strict supérieur à 1'
else
    w=2/(1+sqrt(1-x));
end
```

Fonction 'SOR2'

But : calculer la solution x du système Ax=B par la méthode S.O.R. avec le paramètre optimal ω_0

```
%Méthode S.O.R
%Arguments d'entrée:matrice A
%                   vecteur B
%                   critère d'arrêt s
%                   vecteur Xo
%Arguments de sortie: k nombre d'itérations effectuées
%                     vecteur solution X

function[X,k,w]=SOR2(A,B,s,Xo)


X=Xo;
n=size(A);
n=n(1);
w=w0(A);
%Initialisation du critère
h=max(abs(A*Xo-B));

%Initialisation du compteur d'itérations
k=0;

while h>s   %Poursuite des itérations tant que le critère d'arrêt n'est pas
vérifié
%Calcul des composantes du vecteur Xk
for i=1:1:n
        p=0;
        q=0;
        for j=1:1:i-1
            p=p+A(i,j)/A(i,i)*X(j);
        end
        for j=i+1:1:n
            q=q+A(i,j)/A(i,i)*X(j);
        end
        X(i)=(1-w)*X(i)-w*p-w*q+w*B(i)/A(i,i);
end
    %Calcul de la valeur du critère
    h=max(abs(A*X-B));

    %Incrémentation du compteur
    k=k+1;
end
```

Fonction 'erreur'

But : calculer les erreurs en norme infinie discrète absolue

```
% Calcul de l'erreur
% Norme infinie discrète absolue
% Paramètre d'entrée : vecteur ligne u_exacte et vecteur ligne u_methode
% Paramètre de sortie : valeur erreur_methode
```

```
function erreur_methode=erreur(u_exacte,u_methode,i_0,i_N,debut,fin,h)


% Il faut d'abord calculer la valeur absolue de l'erreur pour chaque point
discrétisé.

for i=i_0:i_N
   erreur_methode_i(i)=abs(u_methode(i)-u_exacte(i));
end

% Puis il faut prendre le max sur i de l'erreur absolue.

erreur_methode=max(erreur_methode_i);
```

Fonction 'resultat_h'

But : entrer les données de l'énoncé et regrouper toutes les fonctions précédentes en une seule qui les exécute au fur et à mesure

```
% Paramètres de sortie : matrice erreur_h contenant les erreurs obtenues par
les 2 schémas numériques, et avec différentes méthodes de résolution
%                       matrice iterations contenant - les valeurs de w0
%                                                    - le nombre d'itérations
de la méthode SOR quelconque dans chaque cas
%                                                    - le nombre d'itérations
de la méthode SOR optimisée dans chaque cas
%                                                    - l'information de
permutation ou non de la méthode Gaussband


function [erreur_h,iterations]=resultat_h(h,w,s)


Xo=zeros(1/h+1,1);

% DONNEES DE L'ENONCE
debut=0;
fin=1;
u_debut=0;
u_fin=0;


% CALCUL DES INDICES D'ITERATION
i_0=1;
i_N=(fin-debut)/h+1;
i_milieu=((i_N-i_0)-1)/2+1;


% DEROULEMENT DES CALCULS

% Création du vecteur de la solution exacte
u_exacte=solution_exacte(h,i_0,i_milieu,i_N,u_debut,u_fin);

% Définition de b, coefficient discontinu de l'équation différentielle de
départ
for i=i_0:i_milieu
   b(i)=1;
end
for i=(i_milieu+1):i_N
   b(i)=100;
```

```
end

% Définition des coefficients des schémas numériques
[c_efg,d_efg]=coef_efg(i_0,i_milieu,i_N,h,b);
[c_lpdem,d_lpdem]=coef_lpdem(i_0,i_milieu,i_N,h,b);

% Calcul de la matrice A des coefficients et du second membre B (Au=B)
[A_efg,B_efg]=matrice_efg(debut,fin,h,u_debut,u_fin,c_efg,d_efg);
[A_lpdem,B_lpdem]=matrice_lpdem(debut,fin,h,u_debut,u_fin,c_lpdem,d_lpdem);


% Calcul du vecteur de la solution approchée par différentes méthodes de
résolution
u_efg_1=Gauss(A_efg,B_efg);
u_lpdem_1=Gauss(A_lpdem,B_lpdem);

u_efg_2=GaussMax(A_efg,B_efg);
u_lpdem_2=GaussMax(A_lpdem,B_lpdem);

u_efg_3=GaussBand(A_efg,B_efg,1);
u_lpdem_3=GaussBand(A_lpdem,B_lpdem,1);

[u_efg_4,bool_efg]=GaussMaxBand(A_efg,B_efg,1);
[u_lpdem_4,bool_lpdem]=GaussMaxBand(A_lpdem,B_lpdem,1);

% k est le nombre d'itérations
[u_efg_5,k_efg_w]=SOR(A_efg,B_efg,w,s,Xo);
[u_lpdem_5,k_lpdem_w]=SOR(A_lpdem,B_lpdem,w,s,Xo);

[u_efg_6,k_efg_w0,w0_efg]=SOR2(A_efg,B_efg,s,Xo);
[u_lpdem_6,k_lpdem_w0,w0_lpdem]=SOR2(A_lpdem,B_lpdem,s,Xo);


% Calcul de l'erreur en norme infinie discrète

erreur_efg(1)=erreur(u_exacte,u_efg_1,i_0,i_N,debut,fin,h);
erreur_efg(2)=erreur(u_exacte,u_efg_2,i_0,i_N,debut,fin,h);
erreur_efg(3)=erreur(u_exacte,u_efg_3,i_0,i_N,debut,fin,h);
erreur_efg(4)=erreur(u_exacte,u_efg_4,i_0,i_N,debut,fin,h);
erreur_efg(5)=erreur(u_exacte,u_efg_5,i_0,i_N,debut,fin,h);
erreur_efg(6)=erreur(u_exacte,u_efg_6,i_0,i_N,debut,fin,h);

erreur_lpdem(1)=erreur(u_exacte,u_lpdem_1,i_0,i_N,debut,fin,h);
erreur_lpdem(2)=erreur(u_exacte,u_lpdem_2,i_0,i_N,debut,fin,h);
erreur_lpdem(3)=erreur(u_exacte,u_lpdem_3,i_0,i_N,debut,fin,h);
erreur_lpdem(4)=erreur(u_exacte,u_lpdem_4,i_0,i_N,debut,fin,h);
erreur_lpdem(5)=erreur(u_exacte,u_lpdem_5,i_0,i_N,debut,fin,h);
erreur_lpdem(6)=erreur(u_exacte,u_lpdem_6,i_0,i_N,debut,fin,h);

erreur_efg=erreur_efg';
erreur_lpdem=erreur_lpdem';



erreur_h=[erreur_efg,erreur_lpdem];

iterations=[w0_efg,w0_lpdem;k_efg_w,k_lpdem_w;k_efg_w0,k_lpdem_w0;
bool_efg,bool_lpdem];
```

Maple

```
> with(linalg):
```

Définition de la matrice A et du second membre B de la méthode EFG

```
> efg:=proc(n::integer,a,f,u,v)
local A,b,c,d;
A:=matrix(n+2,n+2,0);
b:=vector(n+2);
b[1]:=u; b[n+2]:=v;
A[1,1]:=1; A[n+2,n+2]:=1;
for i from 2 to n+1 do
c:=evalf(Int(1/a,(i-2)/(n+1)..(i-1)/(n+1)));
d:=evalf(Int(1/a,(i-1)/(n+1)..i/(n+1)));
A[i,i-1]:=-1/c:
A[i,i]:=1/c+1/d:
A[i,i+1]:=-1/d:
b[i]:=(n+1)*(int(f(x),x=(i-1)/(n+1)..i/(n+1))- int(f(x),x=(i-2)/(n+1)..(i-
1)/(n+1))):
od:
RETURN(evalm(A),evalm(b));
end:
```

Définition de la matrice A et du second membre B de la méthode LPDEM

```
> lpdem:=proc(n::integer,a,f,u,v)
local A,b,c,d;
A:=matrix(n+2,n+2,0);
b:=vector(n+2);
b[1]:=u; b[n+2]:=v;
A[1,1]:=1; A[n+2,n+2]:=1;
for i from 2 to n+1 do
c:=evalf(Int(1/a,(i-2)/(n+1)..(i-1)/(n+1)));
d:=evalf(Int(1/a,(i-1)/(n+1)..i/(n+1)));
A[i,i-1]:=d:
A[i,i]:=-evalf(Int(1/a,(i-2)/(n+1)..i/(n+1))):
A[i,i+1]:=c:
b[i]:=d*evalf(Int(f/a,(i-2)/(n+1)..(i-1)/(n+1)))- c*evalf(Int(f/a,(i-
1)/(n+1)..i/(n+1))):
od:
RETURN(evalm(A),evalm(b));
end:
```

Définition du coefficient discontinu b(x)

```
> b:=x->if 0<x and x<1/2 then 1 else 100 fi; h:=x->x:
```

Appel des solutions approchées obtenues par les deux schémas

```
> linsolve(efg(4,b,h,0,0)[1],efg(4,b,h,0,0)[2])
> linsolve(efg(20,b,h,0,0)[1],efg(20,b,h,0,0)[2]);
> linsolve(efg(40,b,h,0,0)[1],efg(40,b,h,0,0)[2]);

> linsolve(lpdem(4,b,h,0,0)[1],lpdem(4,b,h,0,0)[2]);
> linsolve(lpdem(20,b,h,0,0)[1],lpdem(20,b,h,0,0)[2]);
> linsolve(lpdem(40,b,h,0,0)[1],lpdem(40,b,h,0,0)[2]);
```