MDI224

Interpolation par splines cubiques

Travaux Pratique 1 - Deuxième semestre de 2011

Professeur: Roland BADEAU

Anthony CLERBOUT	Casier: 234
Tiago CHEDRAUOI SILVA	Casier: 214

Décembre 15, 2011

Table des matières

Table des figures

1 Résolution du système linéaire

1.1 Méthode de Jacobi

Jacobi converge ssi $\rho(D-1(L+U)) < 1$ d'après le cours. Or ici, $max(L) = max(U) = 1$ et $D-1$ est une matrice diagonale dont la plus grande valeur propre est $\frac{1}{2}$. Donc $\rho < 1$.

1.1.1 Implémentation

Pour voir la méthode de Jacobi, on a fait le code suivant :

Méthode de Jacobi

```
% % % % % % % % % % % % % % % % % % %
% 12/12/11
% Chedraoui Silva,Tiago
% Casier: 214
% CLERBOUT, Anthony
% Casier: 234
% TP1: interpolation par splines cubiques
% Description: Methode jacobi
% pour resoudre un systeme lineaire
% % % % % % % % % % % % % % % % % % %

function x = jacobi(A,b,x0,eps,maxit)

% Entree
% A : matrice
% b : vecteur
% x0: vecteur d'initialisation
% esp: critere de convergence
% maxit: nombre maximal d'iterations

% A in [N X N]
N = size(A);

% Iniatialization sortie
x = x0;

% Decomposition de A:
% A = M - K
% M = D
% K = L + U

% Pour rappeler:
% ----------
% | d -u -u|
% |-l  d -u|
% |-l -l  d|
% ----------
% Donc:
D = diag(diag(A));
K = A-D;

for i=1:maxit,

  xn=D\(b-K*x(:,i));

  x=[x xn];

  % sauvegarder les valeurs pour faire
  % le plot log(erreur) X iteres
  err=norm( x(:,i+1) - x(:,i) );

  % si l'erreur d'approximation est plus petite
  % que eps on doit arreter
  if (err <= eps)
    break;
  end;

  % sinon je doit repeter l'iteration jusqu'a convergence

end;
```

Pour voir le convergence de la méthode de Jacobi, on a pris, pour chaque itération, chaque valeur de x calculée jusqu'au moment oú le critére d'arrêt est atteint. Après, on a fait la comparaison de chaque x avec la valeur optimale (xex) en prenant le log de la différence :

1.1.2 Convergence

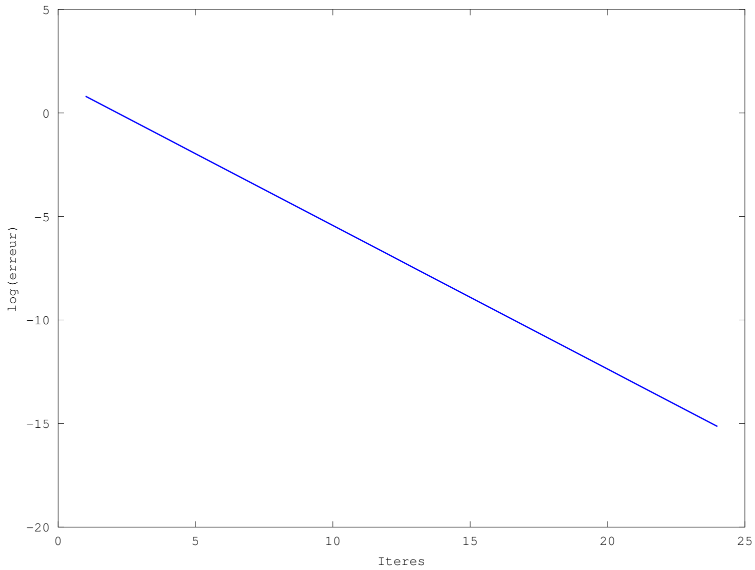


FIG. 1: Convergence de la méthode de Jacobi

En utilisant la fonction polyfit, nous avons trouvé le polynome : $p(x) = -0.693147x + 1.497866$

1.2 Méthode de relaxation

La çéthode de la relaxation converge ssi $(\frac{D}{w} - L) - (\frac{(1-w)}{w}D + U) < 1$ d'après le cours si A est tridiagonale et définie positive, alors la suite (x_k) converge effectivement vers l'unique solution de $Ax = b$.

1.2.1 Implémentation

Pour voir la méthode SOR (relaxation), on a fait le code suivant :

Méthode de Relaxation

```
% % % % % % % % % % % % % % % % % % %
% 12/12/11
% Chedraoui Silva,Tiago
% Casier: 214
% CLERBOUT, Anthony
% Casier: 234
```

```matlab
% TP1: interpolation par splines cubiques
% Description: Methode relaxation
% pour resoudre un systeme lineaire
% % % % % % % % % % % % % % % % % % %

function [x,rho] = relax(A,b,x0,w,eps,maxit)

% Entree
% A : matrice
% b : vecteur
% x0: vecteur d'initialisation
% w : parametre de relaxation
% esp: critere de convergence
% maxit: nombre maximal d'iterations

% A in [N X N]
N = size(A);

% Iniatialization sortie
x = x0;

% Decomposition de A:
% A = M - K
% M = D/w - L
% K = (1-w)D/w + U

% Pour rappeler:
% ----------
% | d -u -u|
% |-l  d -u|
% |-l -l  d|
% ----------

% Donc:
D = diag(diag(A));
U = (-1)*triu(A,1);
L = (-1)*tril(A,-1);

% Obs: Le deuxieme valeur de triu et tril
% est utilise por mettre des zeros au
% lieu de la diagonal principal

M = D/w - L;
K = (1.0-w)*D/w + U;

Rw = M\((1.0-w)*D/w + U);
rho = max(abs(eig(Rw)));

for i=1:maxit,

  xn = M\((K*x(:,i))+b);

  x = [x xn];

  % sauvegarder les valeurs pour faire
  % le plot log(erreur) X iteres
  err=norm( x(:,i+1) - x(:,i) );

  % si l'erreur d'approximation est plus petite
  % que eps on doit arreter
  if (err <= eps)
    break;
  end;

  % sinon je doit repeter l'iteration jusqu'a convergence

end;
```

Pour voir la convergence de la méthode SOR, on a pris, pour chaque itération, chaque valeur de x calculée jusqu'au moment l'algorithme atteint son critére d'arrêt(pour w=1.0). Après, on a fait la comparaison de chaque x avec la valeur optimal (xex) en prenant le log de la différence :

1.2.2 Convergence

Pour voir le taux, il existe deux méthodes. La première utilise les erreurs et la fonction polyfit de matlab. La deuxième le rayon spectral de la matrice R_w. Ainsi, on change les valeurs de w de manière à trouver le meilleur taux de convergence. Les graphes pour les deux approches sont représentés dans les graphes en 3 et 4 bas. On a trouvé un $w_{optimal} \approx 1.1$

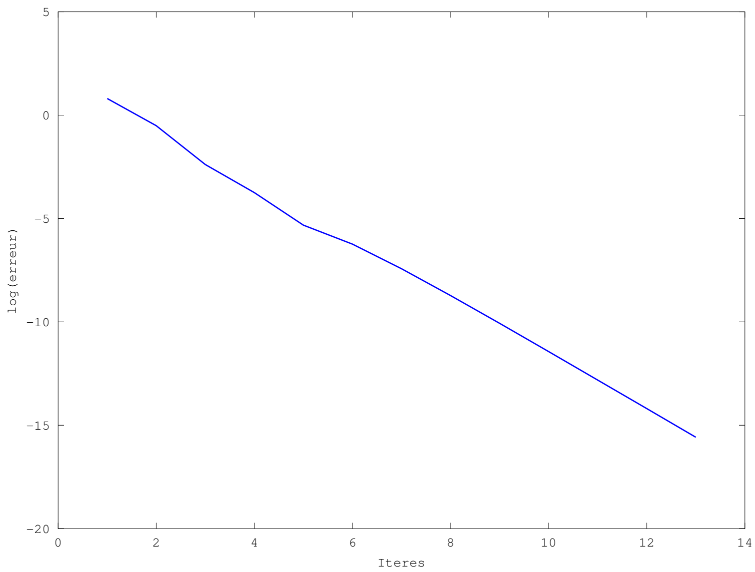
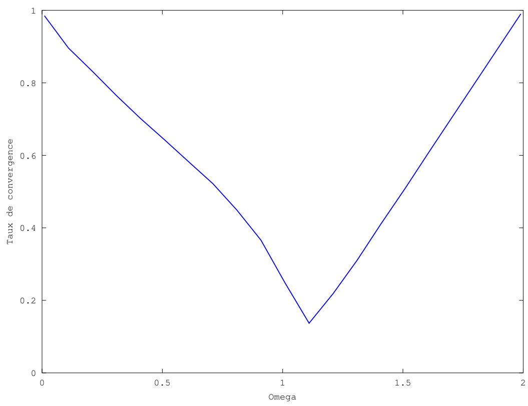
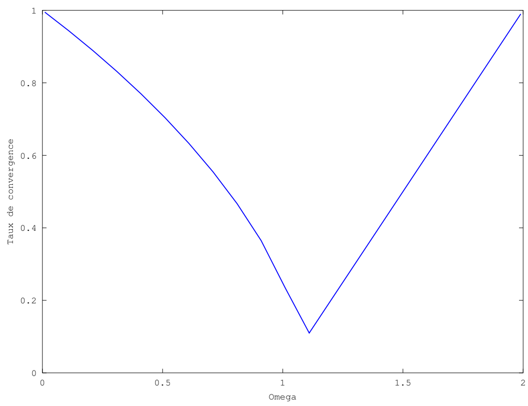


FIG. 2: Convergence de la méthode de Relaxation pour w=1.0



(a) Utilisant la fonction Polyfit et la suite d'erreurs



(b) Calcule du rayon spectral de la matrice R_w.

FIG. 3: Taux de convergence en fonction du paramètre de relaxation ω

1.3 Méthode de Cholesky

1.3.1 Implémentation

La fonction suivante était fournie par le problème.

Méthode de Cholesky

```
function x = cholesky(A,p,b)

N = size(A,1);
L = zeros(N);
for i=1:N,
    L(i,i) = sqrt(A(i,i) - sum(L(i,max(1,i-p):i-1).^2));
    for j=i+1:min(i+p,N);
        L(j,i) = (A(i,j) - sum(L(i,max(1,j-p):i-1).*L(j,max(1,j-p):i-1))) / L(i,i);
    end;
end;
x = L'\(L\b);
```

Pour le même exemple que précèdemmet, on a trouvé :

p	x
4	[1.00000 ;1.00000 ;1.00000 ;1.00000 ;1.00000]
3	[1.00000 ;1.00000 ;1.00000 ;1.00000 ;1.00000]
2	[1.00000 ;1.00000 ;1.00000 ;1.00000 ;1.00000]
1	[1.00000 ;1.00000 ;1.00000 ;1.00000 ;1.00000]

TAB. 1: Vérification fonction Cholesky

1.3.2 Complexité

La complexité de la fonction Cholesky est donnée par $O(\frac{N^2p}{2})$, pour les autres méthodes on a :

Jacobi Le calcul de x^{n+1} par Jacobi est donné par : $x^{n+1} = D^{-1}(L+U)x^n + D^{-1}b$. Comme la matrice A est composée seulement de termes non nuls dans la diagonale principale et dans deux diagonales secondaires, on a : $(L+U)$ qui est une matrice avec environ 2N termes et $(D)^{-1}$ est une matrice avec N termes :

Ainsi, le nombre de multiplications pour calculer $D^{-1}(L+U)x^n \in O(3N)$. De plus, $D^{-1}b \in O(N)$, que l'on peut donc négliger. Finalement, $Jacobi \in O(3N)$

Relaxation 'Le calcul de x^{n+1} par la méthode de la Relaxation est donné par : $x^{n+1} = (\frac{D}{w} - L)^{-1}(\frac{1-w}{w}D + U)x^n + (\frac{D}{w} - L)^{-1}b$. Comme la matrice A est composée seulement par des termes non nuls dans la diagonale principale et dans deux diagonales secondaires, on a : $(\frac{D}{w} - L)^{-1}$ qui est une matrice avec plus de $2N$ termes (on a trouvé cette valeur en utilisant Matlab pour la matrice A dans la section précédente) et $(\frac{1-w}{w}D + U)$ est alors une matrice avec 2N termes. Ainsi, $(\frac{D}{w} - L)^{-1}(\frac{1-w}{w}D + U)x^n \in O(5N)$ et $(\frac{D}{w} - L)^{-1}b \in O(3N)$. Finalement $Relax \in O(5N)$

Afin de calculer le temps d'exécution des méthodes de Jacobi, Relaxation et Cholesky, les codes suivants ont été crées.

Calcul temps Jacobi

```
%section 3.3.2: complexite%

for N=50:50:300,

  % pour chaque N on doit recree l'entree pour la methode
  A=4*eye(N) + diag(ones(1,N-1),1) + diag(ones(1,N-1),-1);
  A(1,1)=2; A(N,N)=2;
  xex = ones(N,1);
  b = A*xex;
  x0 = zeros(N,1);

  % On ira prendre le temps pour executer la methode
  tic ();
  x = jacobi(A,b,x0,eps,maxiter);
  elapsed_time = toc ();

  printf('Jacobi time for N = %d :\n',N);
  disp(elapsed_time);

end;
```

Calcul temps Relaxation

```
%section 3.3.2: complexite%


w=1.1; % parametre optimal (approximation)

for N=50:50:300,

  % pour chaque N on doit recree l'entree pour la methode
  A=4*eye(N) + diag(ones(1,N-1),1) + diag(ones(1,N-1),-1);
  A(1,1)=2; A(N,N)=2;
  xex = ones(N,1);
  b = A*xex;
  x0 = zeros(N,1);

  % On ira prendre le temps pour executer la methode
  tic ();
  [x,rho] = relax(A,b,x0,w,eps,maxiter);
  elapsed_time = toc ();

  printf('Relax time for N = %d :\n',N);
  disp(elapsed_time);

end;
```

Calcul temps Cholesky

```
%section 3.3.2: complexite%

p=1; % p<<N
for N=50:50:300,

  % pour chaque N on doit recree l'entree pour la methode
  A=4*eye(N) + diag(ones(1,N-1),1) + diag(ones(1,N-1),-1);
  A(1,1)=2; A(N,N)=2;
  xex = ones(N,1);
  b = A*xex;
  x0 = zeros(N,1);

  % On ira prendre le temps pour executer la fonction cholesky
  tic ();
  x = cholesky(A,p,b);
  elapsed_time = toc ();

  printf('CHOLESKY time for N = %d :\n',N);
  disp(elapsed_time);

end;
```

En exécutant les codes, on a trouvé :

En général la méthode Jacobi est la méthode la plus rapide pour un système tridiagonal, parce qu'elle fait moins de multiplications. Néanmoins, la méthode de relaxation fait moins d'itérations que Jacobi, c'est pour cela que quand N est petite la méthode relaxation est plus rapide que Jacobi ; mais lorsque N croit beaucoup, la méthode de Jacobi fait moins multiplications même avec plus d'interaction, ce qui le fait le plus vite.

N	50	100	150	200	250	300
Cholesky ($p = 1$)	0.011554	0.023093	0.036068	0.047571	0.061026	0.074059
Jacobi	0.0026100	0.0031020	0.0041549	0.0048039	0.0078550	0.0097679
Relaxation ($w = 1.1$)	0.0020840	0.0025171	0.0036389	0.0065949	0.0083960	0.018082

TAB. 2: Comparaison des temps entre les trois méthodes pour un système tridiagonal

2 Application

2.1 Calcul d'une spline d'interpolation

Méthode spline cubique

```
% % % % % % % % % % % % % % % % % % %
% 12/12/11
% Chedraoui Silva,Tiago
% Casier: 214
% Anthony CLERBOUT
% Casier: 234
% TP1: interpolation par splines cubiques
% Description: Calculer la spline cubique
% d'interpolation
% % % % % % % % % % % % % % % % % % %

function sp = sinterp(y,h)

% Entree
% y : vecteur

% y in [N X 1]
N = size(y,2);

% Sortie
%sp =  zeros(N);

%disp(N);
%disp(sp);

% Iniatialization matrice A (Ax=b)
A = zeros(N,N);
A(1,1)=2; A(N,N)=2;
A(1,2)=1; A(N,N-1)=1;

for i=2:N-1,
  A(i,i-1)=1;
  A(i,i)=4;
  A(i,i+1)=1;
end;

% Iniatialization vecteur b (Ax=b)
b = zeros(N,1);

% cas donne par C3

% s(n) - s(n-1)
b(N)=y(N)-y(N-1);

% s2-s1
b(1)=y(2)-y(1);

%
for i=2:N-1,
  b(i)=y(i+1)-y(i-1);
end;

b=(3/h)*b;


sp2=A\(b);
[sp2,rho] = relax(A,b,sp2,1,1e-6,100);

sp= sp2(:,(size(sp2,2)));
end;
```

2.2 Évaluation d'une fonction spline cubique

Méthode spline cubique

```
% % % % % % % % % % % % % % % % % % %
% 12/12/11
% Chedraoui Silva,Tiago
% Casier: 214
% Anthony CLERBOUT
% Casier: 234
% TP1: interpolation par splines cubiques
% Description: evaluer une fonction spline
% cubique aux points donnes par x
% % % % % % % % % % % % % % % % % % %

function y = speval(a,b,s,sp,x,h)

% Entree
% [a,b] : valeurs intervale
% s : valeurs splines
% sp :valeurs derive premier
% x :vecteur de points a value la fonction
% h: pas interaction

% Sortie

% x in [N X 1]
N = size(x,2);
N2 = size(s,2);

% Sortie
y =  zeros(N,1);

% Theorique: Poly page 28/43
% P(i)= diff1*L+diff2*M
% diff1= (t-t_i+1)^2/(h*h)
% diff2= (t-t_i)^2/(h*h)
% L = (s_i+(s_i' + 2 s_i/h)(t-t_i))
% M = (s_{i+1}+(s_{i+1}' - 2 s_{i+1}/h)(t-t_{i+1}))

for j=1:N,

  % Quel points sont les plus proches duquel je voudrais calculer?
  % t1 et t2 vont donner le index du vecteur
  % t11 et t22 vont donner les valeur temporel plus proche
  t1=1;
  t2=2;
  t11=a;
  t22=a+h;

  for k=1:N2-1,
    if(x(j)>(b-k*h))
      t1=N2-k;
      t2=N2-k+1;
      t11=b-k*h;
      t22=b+h*(1-k);
      break;
    end;
  end;

  diff1 = ((x(j)-t22)^2)/(h*h);
  diff2 = ((x(j)-t11)^2)/(h*h);
  L = (s(t1))+((sp(t1)+2*s(t1)/h)*(x(j)-t11));
  M = (s(t2))+((sp(t2)-2*s(t2)/h)*(x(j)-t22));

  y(j)= diff1*L+diff2*M;

end;
```

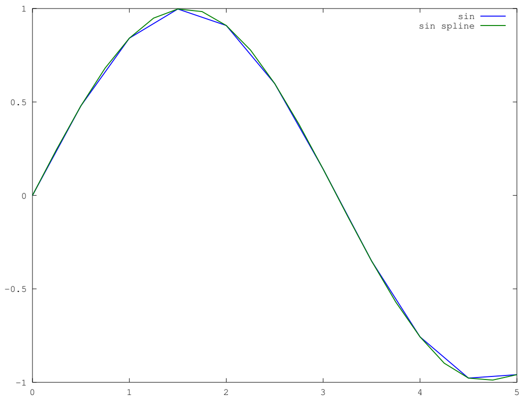
2.3 Application

En utilisant les fonctions mis en œuvre avant (relax et sinterp), on a utilisé les fonctions sinus et exponentielle pour vérifier le spline cubique. Les graphes suivants sont les valeurs interpolées par le spline cubique et l'erreur de cette interpolation est la comparaison entre la valeur calculée par l 'algorithme et la valeur rélle de la fonction.

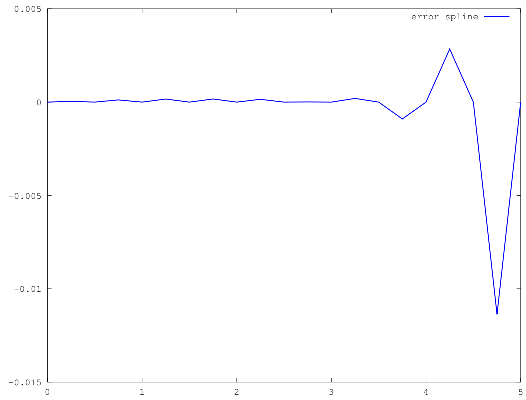
Une observation : les graphes des erreurs montrent que les points d'ordonnée nulle sont les points utilisées pour trouver les valeurs pendant l'interpolation, c'est-à-dire, que ce sont les points en commun entre la fonction réelle et la valeur de l'interpolation. Donc il n'existe pas d'erreur parce que c'est la même valeur. Une autre remarque est que plus la valeur de la fonction est grande, plus est l'erreur est importante.

Pour les graphes 6 et 9, on a augmenté le valeur du pas de la fonction original, cela veut dire, on a pris plus de points en utilisant la fonction original. Alors, la spline cubique ira être plus précise, comme on voit par le graphes des erreurs qui démontrent des valeurs trop petites.

Autre remarque c'est que on a, en utilisant les fonctions spinterp et relax, des erreurs avant la fonction speval. Donc, pour avoir une spline plus précise on pourrait aussi faire des améliorations dans la fonctions relax et spinterp.

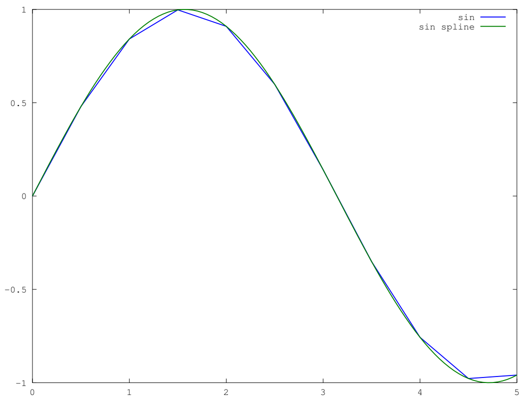


(a) Interpolation

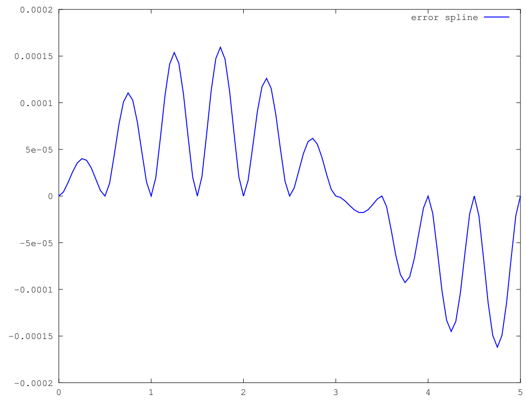


(b) Erreur interpolation.

FIG. 4: Fonction sinus pas original 0.5, pas spline 0.25

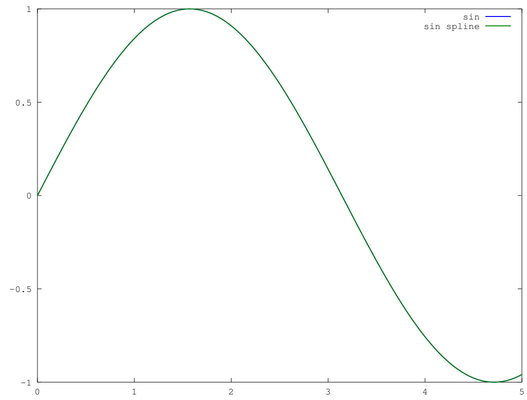


(a) Interpolation

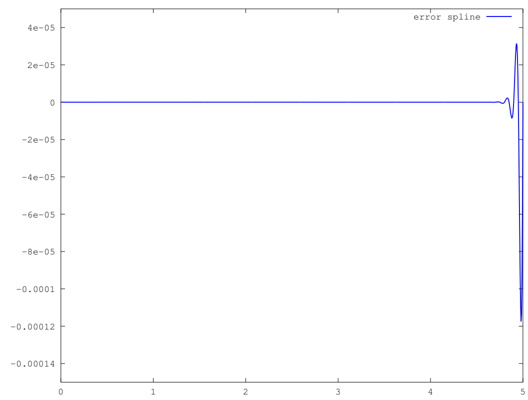


(b) Erreur interpolation.

FIG. 5: Fonction sinus pas original 0.5, pas spline 0.05

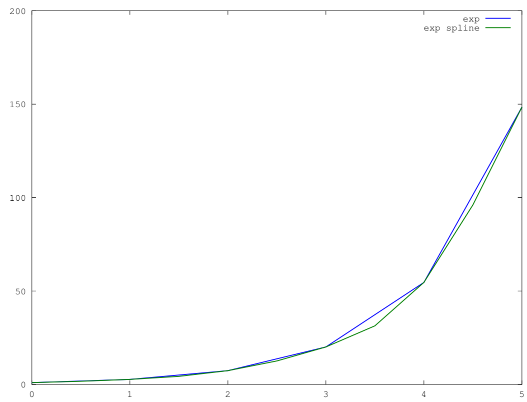


(a) Interpolation

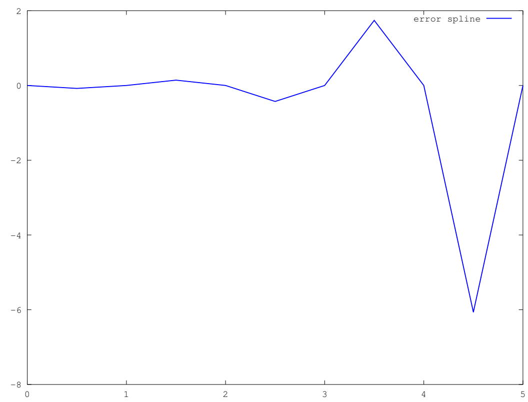


(b) Erreur interpolation.

FIG. 6: Fonction sinus pas original 0.05, pas spline 0.005

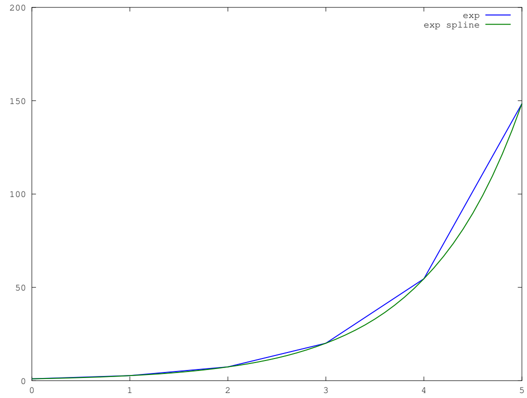


(a) Interpolation

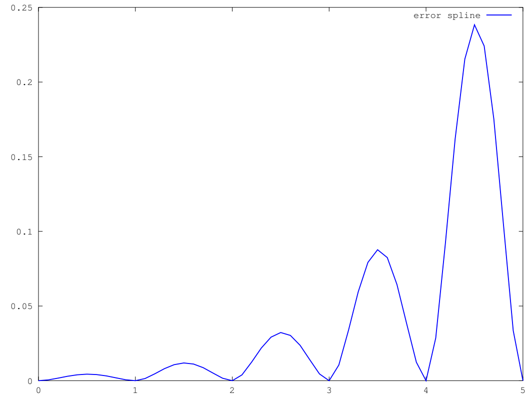


(b) Erreur interpolation.

FIG. 7: Fonction exp pas original 1, pas spline 0.5

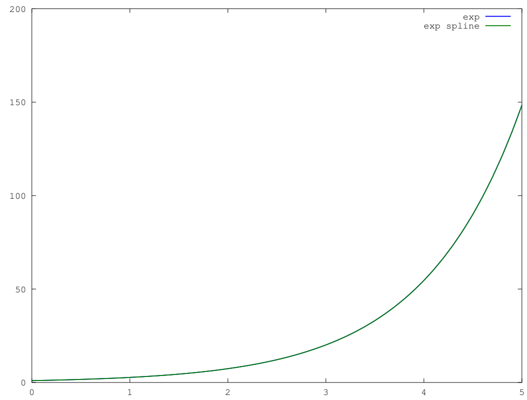


(a) Interpolation

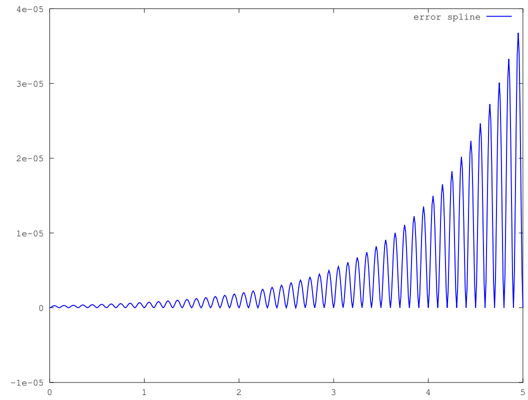


(b) Erreur interpolation.

FIG. 8: Fonction exp pas original 1, pas spline 0.1



(a) Interpolation



(b) Erreur interpolation.

FIG. 9: Fonction exp pas original 0.1, pas spline 0.01