Rapport MT44
TP3

Nicolas Fleurot
Tony Duong

21 juin 2015

Table des matières

Introduction

Dans ce TP, on s'intéresse au calcul numérique d'intégrales. Deux approches de la problématique de l'intégration sont proposées, les méthodes d'intégration dîtes classiques et les méthodes gaussiennes.

Bien que profondément différentes dans leur construction, ces deux approches s'appuient sur les acquis de l'interpolation vue au TP1. En effet, ces méthodes d'intégration sont très utiles pour calculer numériquement l'intégrale d'une fonction interpolant des points de mesure par exemple.

Ce TP commence donc par la mise en oeuvre des méthodes d'intégration classiques et se finira sur les méthodes d'intégration gaussienne.

Partie 1 : Mise en oeuvre des méthodes d'intégration classiques

Introduction

Soient A et B deux réels tels que $A < B$, f une fonction de $[A; B]$ dans $\mathbb{R}$. On désire obtenir une approximation de l'intégrale de f sur $[A; B]$ par les méthodes classiques d'intégration (méthodes des rectangles, des milieux, des trapèzes et de Simpson) en prenant N sous-intervalles (N entier naturel non nul). On considère le pas de discrétisation défini par $h = \frac{(B-A)}{N}$.

Question 1.a

Rappel

Écrire la fonction Matlab *integ_ classique(type, A, B, N, f)* qui renvoie la valeur I de l'intégrale (supposée existée) à partir du type d'intégration choisi parmi les méthodes classiques, de la fonction f , de l'intervalle d'intégration $[A, B]$ et du nombre N de points. La fonction f , passée en paramètre, désignera une chaîne ; elle représente alors une fonction "mathématique" ordinaire.

Théorie

Soit f une fonction régulière sur $[A, B]$ (on supposera f de classe C^4) et n et N deux entiers naturels.

- On découpe $[A, B]$ en sous-intervalles à pas constant h (h appartient à $\mathbb{R}^{+*}$), notés $[x_i, x_{i+1}]$, avec $x_0 = A$ et $x_N = V$ et pour tout i appartient à $0, ..., N-1$, $x_{i+1} - x_i = h$
- Sur chaque sous-intervalle $[x_i, x_{i+1}]$, on considère la fonction $p_{i,n}$ qui interpole f en des points $x_{i,0}, ..., x_{i,n}$ en nombre suffisant. A cette fonction, on applique les résultats de l'intégration sur un intervalle élémentaire.
- On obtient alors, la valeur approchée de intégrale de A vers B de $f(x)dx$ est somme de N valeurs approchées élémentaires

$$\int_A^B f(x)\,\mathrm{d}x \simeq I_R^N = h \sum_{i=0}^{N-1} f(x_i) \tag{1}$$

$$\int_A^B f(x)\,\mathrm{d}x \simeq I_M^N = h \sum_{i=0}^{N-1} f\left(x_i + \frac{h}{2}\right) \tag{2}$$

$$\int_A^B f(x)\,\mathrm{d}x \simeq I_T^N = \frac{h}{2}(f(A) + f(B)) + h \sum_{i=1}^{N-1} f(x_i) \tag{3}$$

$$\int_A^B f(x)\,\mathrm{d}x \simeq I_S^N = \frac{h}{6}\left[f(A) + f(B) + 2\sum_{i=1}^{N-1} f(x_i) + 4\sum_{i=0}^{N-1} f\left(x_i + \frac{h}{2}\right)\right] \tag{4}$$

Source

Listing 1 – integ_classique

```
function [ In ] = integ_classique( type, A, B, N, f)
%INTEG_CLASSIQUE calcule l'integration numerique en N points de la fonction f (chaine de caractere)
    entre
%A et B (B > A) pour un type donne.
%Type :
%0 -> methode des rectangles
%1 -> methode des points milieux
%2 -> methode des trapezes
%3 -> methode de simpson

switch type
    case 0
        In = integ_rectangle(A, B, N, f);
    case 1
        In = integ_milieu(A, B, N, f);
    case 2
        In = integ_trapeze(A, B, N, f);
    case 3
        In = integ_simpson(A, B, N, f);
end

end
```

Listing 2 – integ_rectangle

```
function [ In ] = integ_rectangle( A, B, N, f )
%INTEG_RECTANGLE Retourne le resultat de l'integration par methode des
%rectangles de la fonction f (chaine de caractere) dans l'intervalle [A,B]
%avec N points

    % On calcule la taille d'un intervalle
    h = (B-A) / N;
    % On transforme la chaine de caractere en fonction
    func = str2func(['(x)' f]);

    % On applique la methode des rectangles
    In = 0;
    for i=0:N-1
        In = In + func(A+i*h);
    end
    In = In * h;
end
```

Listing 3 – integ_milieux

```
function [ In ] = integ_milieu( A, B, N, f )
%INTEG_MILLIEU Retourne le resultat de l'integration par methode des points
%milieux de la fonction f (chaine de caractere) dans l'intervalle [A,B] avec
%N points

    % On calcule la taille d'un intervalle
    h = (B-A) / N;
    % On transforme la chaine de caractere en fonction
    func = str2func(['(x)' f]);

    % On applique la methode des points milieux
    In = 0;
    for i=0:N-1
        In = In + func(A+i*h + h / 2);
    end
    In = In * h;

end
```

Listing 4 – integ_trapezes

```
function [ In ] = integ_trapeze( A, B, N, f )
%INTEG_TRAPEZ Retourne le resultat de l'integration par methode des trapeze
```

```
%de la fonction f (chaine de caractere) dans l'interval [A, B] avec
%N points

    % On calcule la taille d'un interval
    h = (B-A) / N;
    % On transforme la chaine de caractere en fonction
    func = str2func(['(x)' f]);

    % On applique la methode des trapeze
    In = h/2 * (func(A) + func(B));
    for i=1:N-1
        In = In + func(A+i*h);
    end
    In = In * h;

end
```

Listing 5 – integ_simpson

```
function [ In ] = integ_simpson( A, B, N, f )
%INTEG_SIMPSON Retourne le resultat de l'integration par methode de simpson
%de la fonction f (chaine de caractere) dans l'interval [A,B] avec
%N points

    % On calcule la taille d'un intervalle
    h = (B-A) / N;
    % On transforme la chaine de caractere en fonction
    func = str2func(['(x)' f]);

    % On applique la methode de simpson
    In = 0;

    for i=0:N-1
        In = In + func(A+i*h + h/2);
    end

    In = In * 2;

    for i=1:N-1
        In = In + func(A + i*h);
    end

    In = (In * 2 + func(A) + func(B)) * h / 6;
end
```

Question 1.b

Rappel

Pour le calcul d'intégrales connues, par exemple :

$$I_1 = \int_0^{\frac{\pi}{2}} \sin(x)dx \tag{5}$$

$$I_2 = \int_0^1 \frac{1}{1+x^2}dx \tag{6}$$

$$I_3 = \int_0^1 x^3 + xdx \tag{7}$$

lancer l'exécution de *integ_classique()* pour différentes valeurs de N. Comparer les valeurs approchées obtenues à leur valeur exacte. Quelle méthode vous parait la plus performante ? Quel critère objectif vous permet-il de justifier cela ? Que remarquez-vous de particulier pour l'intégrale I_3 ? Les résultats obtenus sont-ils conformes aux calculs d'erreur de chacune des méthodes, établis en TD. Produire une (ou des) représentation(s) graphique(s) qui permette(nt) de visualiser les performances comparées des différentes méthodes.

Théorie

L'erreur méthodique globale pour chacune des méthodes est obtenue en sommant les N valeurs d'erreurs élémentaires.

On obtient alors,

$$
\begin{aligned}
E_R^N &= h\frac{B-A}{2}f^{(1)}(\eta) \text{ avec } \eta \in]A,B[&& (8)\\
E_M^N &= h^2\frac{B-A}{24}f^{(2)}(\eta) \text{ avec } \eta \in]A,B[&& (9)\\
E_T^N &= -h^2\frac{B-A}{12}f^{(2)}(\eta) \text{ avec } \eta \in]A,B[&& (10)\\
E_R^N &= -\left(\frac{h}{4}\right)^4\frac{B-A}{180}f^{(4)}(\eta) \text{ avec } \eta \in]A,B[&& (11)
\end{aligned}
$$

Test

Listing 6 – Evaluation de l'erreur

```
clear;
clc;

% definition de quelques constantes
% couleurs des courbes
color = ['r', 'g', 'b', 'c'];

% nombre de tests
N = 150;

% definition des fonctions a integrer ainsi que de leur intervalle
% d'integration
f = ['sin(x)    ';'1/(1+x^2)';'x^3+x     '];
A = [0, 0, 0];
B = [pi/2, 1, 1];

% Il nous faut une variable symbolique pour calculer l'integrale 'exacte'
% dans un interval donne
syms x;

% recupere le nombre de fonctions a integrer disponible
nbFunc = size(f);
nbFunc = nbFunc(1);

for n=1:nbFunc
    % On calcule l'integrale 'exacte'
    integ_reel = int(f(n, :), x, A(n), B(n));
    for i=0:3
        % On alloue de la memoire pour contenir nos resultats
        integ_calc = zeros(1, N);
        for j=1:N
            % On calcule la valeur absolue de la difference entre nos
            % integrales numeriques et l'integrale 'exacte' pour la n-ieme
            % fonction, avec la i-eme methode, avec j nombres de points
            integ_calc(j) = abs(integ_reel - integ_classique(i, A(n), B(n), j, f(n, :)));
        end
        % On dessine les courbes representant l'erreur en fonction du nombre
        % de points
        subplot(nbFunc, 1, n);
        plot(integ_calc, 'color', color(i+1));
        % Les warnings n'ont aucune importance
        legend('integ rectangle', 'integ millieu', 'integ trapeze', 'integ simpson');
        title(f(n, :));
        %axis([50 100 0 0.1]);
        hold on;
    end
end
```

Listing 7 – test_perform.m

```
clear;
clc;

```

```
% definition de quelques constantes
% couleurs des courbes
color = ['r', 'g', 'b', 'c'];

% nombre de tests
N = 150;

% definition des fonctions a integrer ainsi que de leur intervalle
% d'integration
f = ['sin(x)    ';'1/(1+x^2)';'x^3+x     '];
A = [0, 0, 0];
B = [pi/2, 1, 1];

% Il nous faut une variable symbolique pour calculer l'integrale 'exacte'
% dans un intervalle donne
syms x;

% recupere le nombre de fonctions a integrer disponible
nbFunc = size(f);
nbFunc = nbFunc(1);

% On alloue de la memoire pour stocker les valeurs temporels
timeSpend = zeros(nbFunc, 4, N);

% Nombre de fois ou une integration est repetee. Plus ce nombre est grand, plus
% le calcul du temps est exact.
Prec = 1;

for n=1:nbFunc;
    for i=0:3
        % Comme les calculs peuvent etre long (suivant la valeur de Prec)
        % on ajoute ecrit a l'ecran une indication de l'avancement
        disp(['func : ' f(n, :) ' type : ' num2str(i)]);
        for j=1:N
            % On reproduit chaque calcul Prec fois
            for k=1:Prec
                % On mesure le temps d'un calcul
                tic;
                integ_classique(i, A(n), B(n), j, f(n, :));
                % On fait la somme de toutes les mesures
                timeSpend(n, i+1, j) = timeSpend(n, i+1, j) + toc;
            end
            % On divise la somme des mesures par le nombre de mesures,
            % Concretement, on fait la moyenne des mesures pour se
            % rapprocher le plus possible de la realite. C'est a dire
            % eliminer le 'bruit' produit par tout les programmes alentours
            % lorsque matlab opere les calculs.
            timeSpend(n, i+1, j) = timeSpend(n, i+1, j) / Prec;
        end
    end
end

% Etrangement, plot ne peut pas utiliser de matrices (1, 1, N) comme une
% matrice (1, N). Il faut donc copier nos donnees vers une matrice de
% taille adequate
tabTime = zeros(1, N);
for i=1:nbFunc
    for j=1:4
        % On copie les donnees comme explique precedemment
        for n=1:N
            tabTime(n) = timeSpend(i, j, n);
        end
        % On trace le temps que met une integrale a etre calcule en
        % fonction du nombre de points
        subplot(nbFunc, 1, i);
        plot(tabTime, 'color', color(j));
        title(f(i, :));
        ylabel('secondes');
        xlabel('nombre de points');
        % Les warnings n'ont aucune importance
        legend('integ rectangle', 'integ millieu', 'integ trapeze', 'integ simpson');
        hold on;
    end
end
```

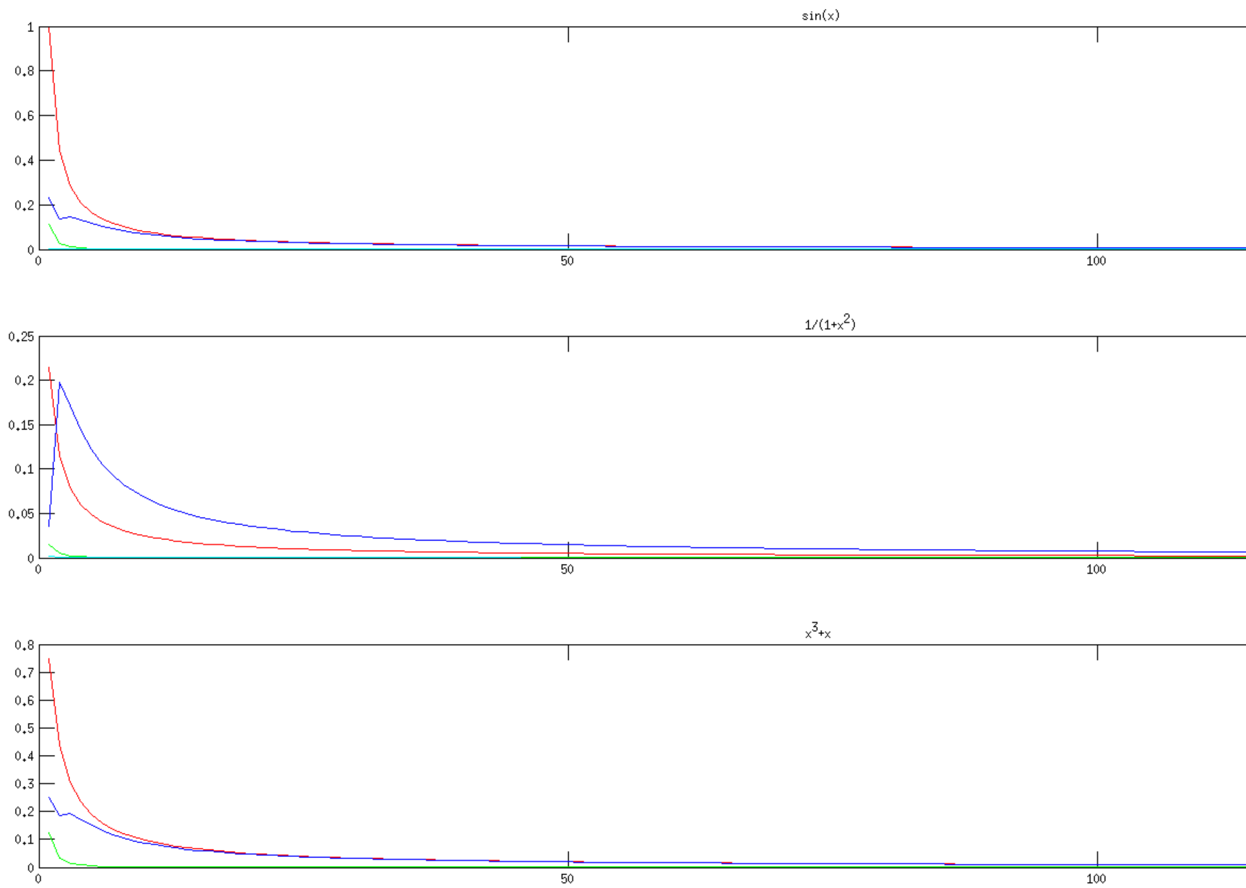

FIGURE 1 – Valeur absolue de l'erreur d'intégration

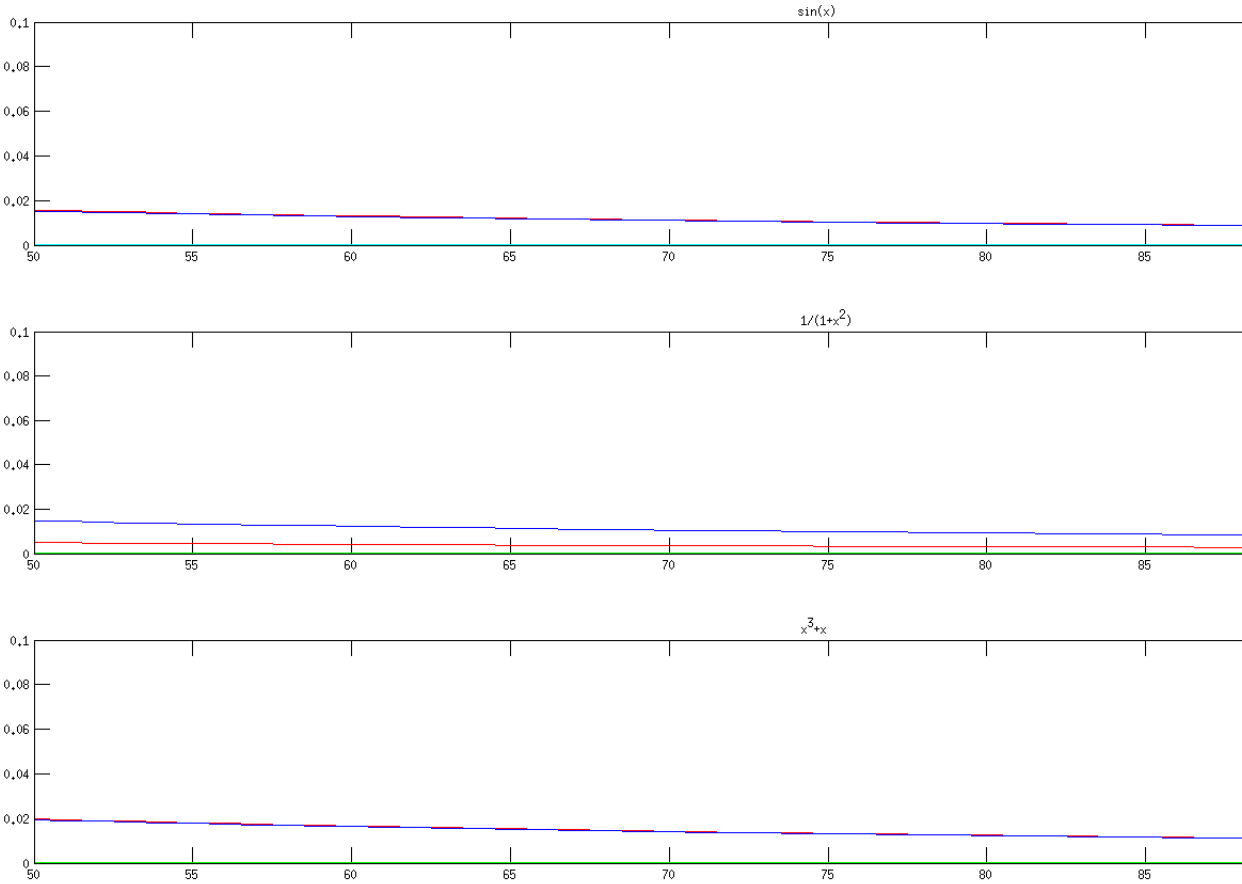

FIGURE 2 – Zoom sur l'erreur d'intégration entre 50 et 100 points

Test de performance temporelle

Nous remarquons que, bien que la méthode de Simpson soit la plus performante niveau résultat, elle prend légèrement plus de temps à le trouver en comparaison avec les trois autres méthodes.

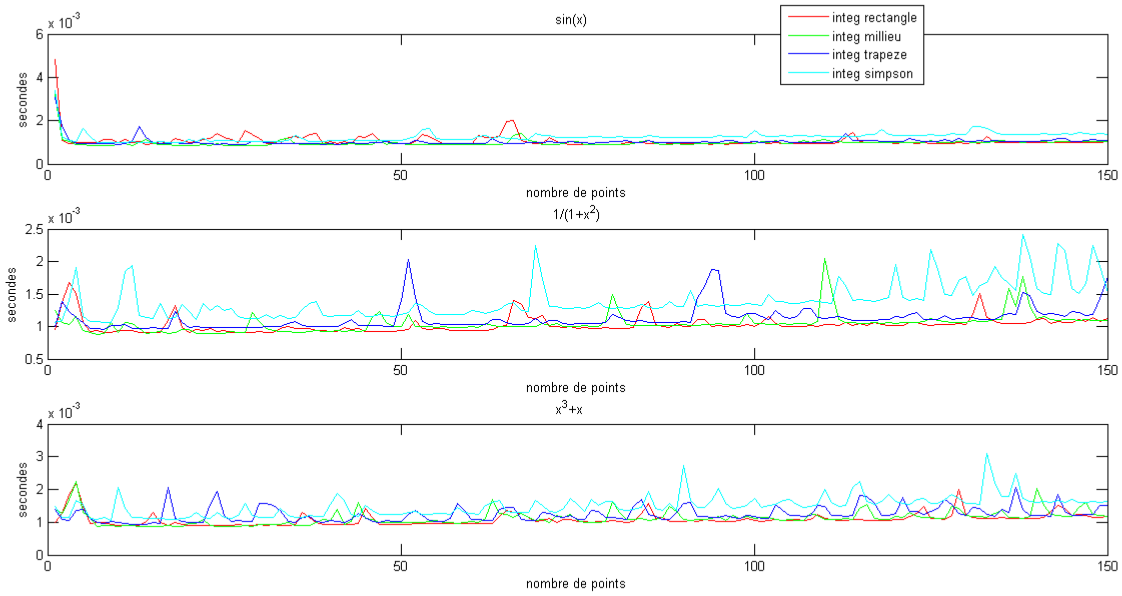

Figure 3 – Test de la performance temporelle des 4 méthodes

Partie 2 : Intégration gaussienne

Introduction

L'objet de cette partie est de préparer, sans qu'il soit nécessairement complet, un "kit d'intégration gaussienne". On se préoccupera de le rendre opérationnel pour les méthodes de Legendre et Tchebyschev ; les autres cas constitueront le "default" d'un switch a développer ultérieurement. La structure globale de la fonction d'intégration gaussienne doit exister ; l'utilisateur peut choisir la méthode a mettre en oeuvre. On ne réfléchira, qu'une fois le reste du TP terminé, a la reconnaissance de la forme d'intégration gaussienne à utiliser en process automatique !

Question 2.a

Rappel

Détermination des polynômes orthogonaux Construire la suite des n max (par exemple n max = 20) premiers polynômes orthogonaux de Legendre et Tchebyschev. On utilisera la relation de récurrence fournie en cours ; on rappelle que :

$$\begin{aligned} &L_0(x) = 1,\ L_1(x) = x, && \forall k \in \mathbb{N}^* L_{k+1} = \frac{2k+1}{k+1} x L_k(x) - \frac{k}{k+1} L_{k-1}(x) \\ &T_0(x) = 1,\ T_1(x) = x, && \forall k \in \mathbb{N}^* T_{k+1} = 2x T_k(x) - T_{k-1}(x) \end{aligned}$$

On stockera les résultats dans des matrices.

Théorie

En mathématiques, une suite de polynômes orthogonaux est une suite infinie de polynômes $p_0(x), p_1(x), p_2(x)$ ... à coefficients réels, dans laquelle chaque $p_n(x)$ est de degré n, et telle que les polynômes de la suite sont orthogonaux deux à deux pour un produit scalaire de fonctions donné.

Source

Listing 8 – Polynomes de Legendre

```
function [ legendre ] = polynome_legendre(k)
%POLYNOME_LEGENDRE Retourne un tableau contenant les polynomes de Legendre

% Nos polynomes dependent d'une variable symbolique
syms x;

% La polynome de degre 0 n'est pas symbolique (c'est 1), mais tous les autres le
% sont. Il faut donc un type de donnee capable de stocker aussi bien des
% symboliques que des reels
legendre = sym(zeros(1, k));

% On assigne les premiers degres
legendre(1) = 1;
if k >= 1
    legendre(2) = x;
end

% On calcule tous les polynomes du degre 3 jusqu'au degre k (si k < 3, la
% boucle est ignoree)
for i=3:k
    legendre(i) = (2*(i-2)+1) / (i-1) * x * legendre(i-1) - (i-2) / (i-1) * legendre(i-2);
```

```
end


end
```

Listing 9 – Polynomes de Tchebyschev

```
function [ tchebyschev ] = polynome_tchebyschev( k )
%POLYNOME_TCHEBYSCHEV Retourne un tableau contenant les polynomes de
%Tchebyschev

% Nos polynomes dependent d'une variable symbolique
syms x;

% La polynome de degre 0 n'est pas symbolique, mais tous les autres le
% sont. Il faut donc un type de donnee capable de stocker aussi bien des
% symboliques que des reels
tchebyschev = sym(zeros(1, k));

% On assigne les premiers degres
tchebyschev(1) = 1;
if k >= 1
    tchebyschev(2) = x;
end

% On calcule tous les polynomes du degre 3 jusqu'au degre k (si k < 3, la
% boucle est ignoree)
for i=3:k
    tchebyschev(i) = 2*x*tchebyschev(i-1)-tchebyschev(i-2);
end

end
```

Test

Listing 10 – Script test.m (affichage des polynômes orthogonaux de leurs racines et des coefficients d'intégration)

```
clear;
clc;

n_max=7;

poly_legendre = polynome_legendre(n_max);
poly_tchebyschev = polynome_tchebyschev(n_max);

zeros_legendre = zeros(n_max-1, n_max-1);
zeros_tchebyschev = zeros(n_max-1, n_max-1);

for i=2:n_max
   rootsPoly = double(roots(sym2poly((poly_legendre(i)))));
   zeros_legendre(:, i-1) = [rootsPoly; zeros(n_max-i, 1)];
   zeros_tchebyschev(:, i-1) = [roots(sym2poly(poly_tchebyschev(i))); zeros(n_max-i, 1)];

end

zeros_legendre = zeros_legendre';
zeros_tchebyschev = zeros_tchebyschev';
coefficients_legendre = coeff_legendre(poly_legendre, zeros_legendre);
coefficients_tchebyschev = coeff_tchebyschev(zeros_tchebyschev);

for i=1:n_max
    poly_legendre(i) = simplify(poly_legendre(i));
    poly_tchebyschev(i) = simplify(poly_tchebyschev(i));
end

disp('polynomes de Legendre')
disp(poly_legendre)
disp('racines des polynomes de Legendre')
disp(zeros_legendre)
disp('coefficients d''integration de Legendre')
disp(coefficients_legendre)

disp('polynomes de Tchebyschev')
```

```
disp(poly_tchebyschev)
disp('racines des polynomes de Tchebyschev')
disp(zeros_tchebyschev)
disp('coefficients d''integration de Tchebyschev')
disp(coefficients_tchebyschev)

%{
OUTPUT pour n = 5

polynomes de Legendre
[ 1, x, (3*x^2)/2 - 1/2, (x*(5*x^2 - 3))/2, (35*x^4)/8 - (15*x^2)/4 + 3/8, (x*(63*x^4 - 70*x^2 + 15)
    )/8, (231*x^6)/16 - (315*x^4)/16 + (105*x^2)/16 - 5/16]

racines des polynomes de Legendre
         0         0         0         0         0         0
    0.5774   -0.5774         0         0         0         0
         0    0.7746   -0.7746         0         0         0
   -0.8611    0.8611   -0.3400    0.3400         0         0
         0   -0.9062   -0.5385    0.9062    0.5385         0
   -0.9325   -0.6612    0.9325    0.6612   -0.2386    0.2386

coefficients d'integration de Legendre
    2.0000         0         0         0         0         0
    1.0000    1.0000         0         0         0         0
    0.8889    0.5556    0.5556         0         0         0
    0.3479    0.3479    0.6521    0.6521         0         0
    0.5689    0.2369    0.4786    0.2369    0.4786         0
    0.1713    0.3608    0.1713    0.3608    0.4679    0.4679

polynomes de Tchebyschev
[ 1, x, 2*x^2 - 1, x*(4*x^2 - 3), 8*x^4 - 8*x^2 + 1, x*(16*x^4 - 20*x^2 + 5), 32*x^6 - 48*x^4 + 18*x
    ^2 - 1]

racines des polynomes de Tchebyschev
         0         0         0         0         0         0
    0.7071   -0.7071         0         0         0         0
         0    0.8660   -0.8660         0         0         0
   -0.9239    0.9239   -0.3827    0.3827         0         0
         0   -0.9511   -0.5878    0.9511    0.5878         0
   -0.9659   -0.7071    0.9659    0.7071   -0.2588    0.2588

coefficients d'integration de Tchebyschev
    3.1416
    1.5708
    1.0472
    0.7854
    0.6283
    0.5236


%}
```

Question 2.b

Rappel

Pour les différentes familles de polynômes orthogonaux prises en compte au paragraphe précédent déterminer leurs zéros et les stocker dans une matrice appropriée.

Théorie

Les zéros des polynômes orthogonaux nous seront indispensables pour calculer l'intégrale numérique.
Pour l'intégration de Legendre, ces zéros ne sont pas connus explicitement mais calculés par une méthode numérique.
En revanche, pour l'intégration de Tchebyschev, les zéros sont connus explicitement et données par :

$$\forall i \in \{0, ..., n\}, x_i = \cos\left(\frac{2i + i}{n+1}\frac{\pi}{2}\right)$$

Source

Listing 11 – Zeros de Legendre

```
zeros_legendre = zeros(nb_points-1, nb_points-1);
poly_legendre = polynome_legendre(nb_points);
for i=2:nb_points
    rootsPoly = double(roots(sym2poly((poly_legendre(i)))));
    zeros_legendre(:, i-1) = [rootsPoly; zeros(nb_points-i, 1)];
end
```

Listing 12 – Zeros de Tchebyschev

```
zeros_tchebyschev = zeros(nb_points-1, nb_points-1);
poly_tchebyschev = polynome_tchebyschev(nb_points);
for i=2:nb_points
    zeros_tchebyschev(:, i-1) = [roots(sym2poly(poly_tchebyschev(i))); zeros(nb_points-i, 1)];
end
```

Test

Voir test de la question 2.a

Question 2.c

Rappel

Détermination des coefficients d'intégration.
On rappelle qu'en intégration gaussienne

$$\int_a^b f(x)dx = \int_a^b r(x)w(x)dx \simeq \sum_{i=0}^{n} D_i r(x_i)$$

où $[a, b]$ désigne l'intervalle conventionnel d'intégration pour la méthode étudiée, r la régularisée de la fonction à intégrer, w la fonction poids associée, D_i les coefficients cherchés et x_i les zéros déterminés à la question précédente. À partir des résultats fournis dans le polycopié, déterminer pour chaque méthode prise en compte, les coefficients nécessaires et les stocker.

Théorie

Pour Gauss-Legendre,

$$\forall i \in \{0, ..., n\}, W_i = \frac{2}{(n+1)L'_{n+1}(x_i)L_n(x_i)}$$

Pour Gauss-Tchebyschev,

$$\forall i \in \{0, ..., n\}, W_i = \frac{\pi}{n+1}$$

Source

Listing 13 – Coefficients d'intégration de Legendre

```
function [ w ] = coeff_legendre( polynomes , roots)
%COEFF_LEGENDRE Calcule les coefficients de Legendre

    n = 0;
    [x y] = size(roots);

    % On alloue un peu de memoire
    w = zeros(x,y);
    dec = y - 1;

    for i=1:x
        if i ~= 1
            % On calcule la derivee des polynomes orthogonaux
            ln1 = diff(polynomes(i+1),1);
            % Le tableau des polynomes sont des symboles, on les
            % transforme donc en fonctions
            ln2 = matlabFunction(polynomes(i));
            ln1 = matlabFunction(ln1);
```

```
            % On calcule le coefficient pour chaque xi
            for j=1:y-dec
                % On calcule xi
                xi = roots(n+1,j);
                % On applique la formule
                denominateur = (n+1)*ln1(xi)*ln2(xi);
                w(i,j)=2/denominateur;
            end
        else % Cas particulier quand i = 1
            ln1 = 1;
            ln2 = 1;
            for j=1:y-dec
               denominateur = (n+1)*ln1*ln2;
               w(i,j)=2/denominateur;
            end
        end
        dec = dec - 1;
        n = n+1;
    end
end
```

Listing 14 – Coefficients d'intégration de Tchebyschev

```
function [ w ] = coeff_tchebyschev( roots)

    n = 0;
    [x y] = size(roots);

    % On alloue un peu de memoire
    w = zeros(x,1);

    % On applique la formule
    for i=1:x
        w(i) = pi/(n+1);
        n = n+1;
    end

end
```

Test

Voir test de la question 2.a

Question 2.d

Rappel

Intégrer les éléments antérieurs pour calculer une valeur approchée d'intégrales convenables par méthode gaussienne. On écrira une fonction *intégration_gaussienne(type, nb points, r)* qui renvoie une valeur approchée de $I = \int_a^b r(x)w(x)dx$, à partir de la donnée du type d'intégration gaussienne requis, du nombre de points à prendre en compte et de la fonction régularisée associée r passée comme une chaîne.

Théorie

Source

Listing 15 – Integration gaussienne

```
function [ In ] = integration_gaussienne( type, nb_points, r )
%INTEGRATION_GAUSSIENNE Calcule l'integration de la fonction symbolique
%regularise r avec nb_points points en utilisant la methode type,
%pour les valeurs :
%0 -> Legendre
%1 -> Tchebyschev

    In = 0;
    % On transforme la fonction symbolique en fonction de matlab
    r = matlabFunction(r);
    switch type
```

```
    case 0
        % Integration Legendre
        % On recupere les polynomes orthogonaux de Legendre pour le nombre de points
        % donne
        poly_legendre = polynome_legendre(nb_points);
        % On alloue un peu de memoire pour contenir les racines des
        % polynomes
        zeros_legendre = zeros(nb_points-1, nb_points-1);
        for i=2:nb_points
            % On calcule les zeros des polynomes de Legendre
            % solve n'est plus capable de trouver les racine a partir d'un
            % polynome de degre 9. On utilise donc roots qui prend en parametres
            % les coefficients du polynome, qui sont donnes par sym2poly
            rootsPoly = roots(sym2poly((poly_legendre(i))));
            zeros_legendre(:, i-1) = [rootsPoly; zeros(nb_points-i, 1)];
        end
        % On s'interesse a la transpose de la matrice
        zeros_legendre = zeros_legendre';
        % On calcule les coefficients de Legendre
        coefficients_legendre = coeff_legendre(poly_legendre, zeros_legendre);

        for i=1:nb_points-1
            % On recupere la ieme racine des polynomes de Legendre
            xi = double(zeros_legendre(nb_points-1, i));
            % On applique la formule
            In = In + coefficients_legendre(nb_points-1, i)*r(xi);
        end
    case 1
        % Integration Tchebyschev
        % On alloue un peu de memoire
        zeros_tchebyschev = zeros(nb_points-1, nb_points-1);
        % On calcule les polynomes de Tchebyschev
        poly_tchebyschev = polynome_tchebyschev(nb_points);
        % On calcule les racines des polynomes. On n'utilise pas solve commepour les meme raison
            que pour
        % les racines de Legendre
        for i=2:nb_points
            zeros_tchebyschev(:, i-1) = [roots(sym2poly(poly_tchebyschev(i))); zeros(nb_points-i,
                1)];
        end
        % On s'interesse a la transpose de la matrice
        zeros_tchebyschev = zeros_tchebyschev';
        % On calcule les coefficients de Tchebyschev
        coefficients_tchebyschev = coeff_tchebyschev(zeros_tchebyschev);

        for i=1:nb_points-1
            % On recupere nos xi
            xi = double(zeros_tchebyschev(nb_points-1, i));
            % On applique la formule
            In = In + coefficients_tchebyschev(nb_points-1)*r(xi) * sqrt(1-xi^2);
        end
    end
end
```

Test

Listing 16 – Test intégration de Gauss-Legendre

```

%{
integration_gaussienne(0,5, sin(x))

ans =

   2.1372e-15

int(sin(x), -1, 1)

ans =

0

integration_gaussienne(0, 5, exp(x))

ans =

```

```
    2.3504

int(exp(x), -1, 1)

ans =

    2.3504
%}
```

Listing 17 – Test intégration de Gauss-Tchebyschev

```

%{
integration_gaussienne(1, 5, sin(x))

ans =

  -5.5511e-16

int(sin(x), -1, 1)

ans =

0

integration_gaussienne(1, 5, sin(x))

ans =

  -5.5511e-16

integration_gaussienne(1, 5, exp(x))

ans =

    2.4352

int(exp(x), -1, 1)

ans =

    2.3504
%}
```

Remarque

L'intégration de sin(x) dans les deux tests devrait être nulle (fonction impaire sur [-1,1], or Matlab nous affiche une erreur de l'ordre de 10^{-15}. Ceci est dû à la propagation de l'erreur.

Question 2.e

Rappel

Pour une ou plusieurs intégrales calculables explicitement, évaluer l'erreur de calcul commise et visualiser cette erreur en fonction du nombre de points considérés.

Théorie

Pour Gauss-Legendre, l'erreur commise est donnée par :

$$E_{n+1} = r^{(2n+2)}(\xi)\frac{2^{2n+3}\left[(n+1)!\right]^4}{(2n+3)\left[(2n+2)!\right]^3}$$

Pour Gauss-Tchebyschev, l'erreur commise est donnée par :

$$E_{n+1} = r^{(2n+2)}(\xi)\frac{2\pi}{2^{2n+2}(2n+2)!}$$

Source

Listing 18 – erreur d'integration de Gauss

```
clear;
clc;

% definition de quelques constantes
% couleurs des courbes
color = ['r', 'b'];

% nombre de tests
N = 15;



% Il nous faut une variable symbolique pour calculer l'integrale 'exacte'
syms x;

% definition des fonctions a integrer
f = [sin(x);1/(1+x^2);exp(x)];

% Definition des titres
titles = ['sin(x)    ';'1/(1+x^2)';'exp(x)    '];

% recupere le nombre de fonctions a integrer disponible
nbFunc = size(f);
nbFunc = nbFunc(1);

for n=1:nbFunc
    % On calcule l'integrale 'exacte'
    integ_reel = int(f(n, :), x, -1, 1);
    for i=0:1
        % On alloue de la memoire pour contenir nos resultats
        integ_calc = zeros(1, N);
        for j=1:N
            % On calcule la valeur absolue de la difference entre nos
            % integrales numeriques et l'integrale 'exacte' pour la n-ieme
            % fonction, avec la i-eme methode, avec j nombres de points
            integ_calc(j) = abs(integ_reel - integration_gaussienne(i, j, f(n, :)));
        end
        % On dessine les courbes representant l'erreur en fonction du nombre
        % de points
        subplot(nbFunc, 1, n);
        plot(integ_calc, 'color', color(i+1));
        title(titles(n, :));
        % Les warnings n'ont aucune importance
        legend('gauss/legendre', 'gauss/tchebyschev');
        hold on;
    end
end
```

Test

Nous remarquons que les erreurs convergent rapidement vers 0. Il faut seulement 3 points pour arriver à une erreur minime pour la fonction e^x. La méthode de Gauss-Legendre est également légèrement plus performante que celui de Gauss-Tchebyschev au niveau des résultats.

De plus, pour la fonction $sin(x)$, l'erreur devrait être nulle, mais comme expliqué précédemment, la propagation de l'erreur est responsable de cet erreur minime.

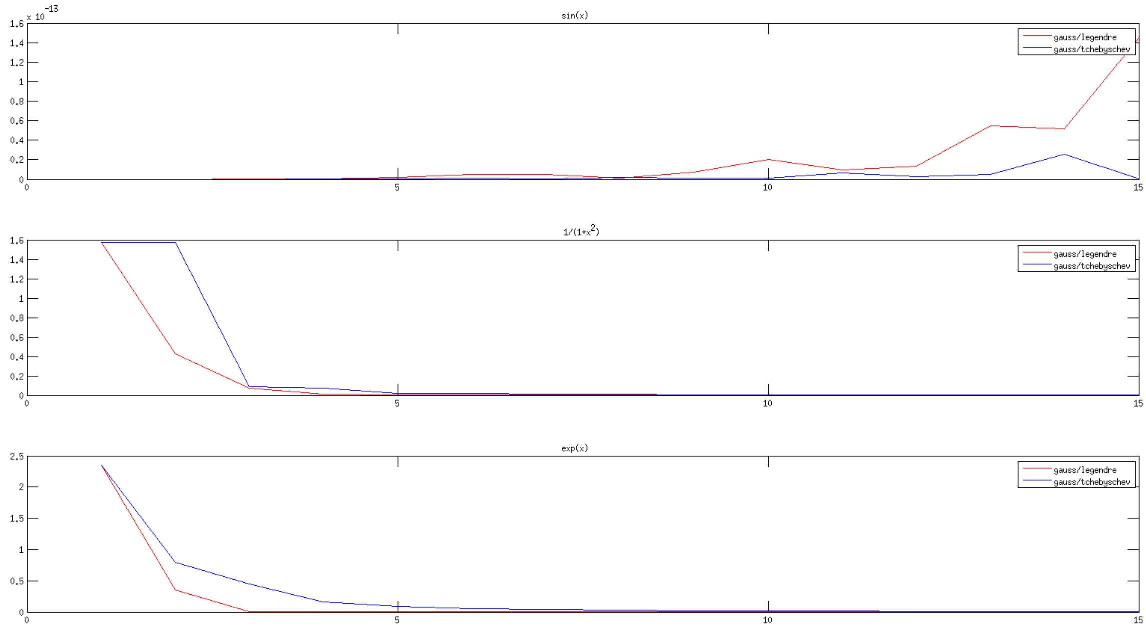

FIGURE 4 – Valeur absolue de l'erreur d'intégration