TP7

Exercice 1. Calcul approché de π.

Soit f la fonction définie par $(\forall x \in [0,1])\ f(x) = \dfrac{4}{1+x^2}$. On note I_n la valeur approchée de

$$I = \int_0^1 f(x)\,dx$$

à l'aide de la méthode des trapèzes, avec n trapèzes.

Créez un répertoire `Exo1` et placez-vous à l'intérieur.

1. Dans un fichier `rect.f90`, programmer une fonction nommée `approx` qui à partir d'un entier `niter` renvoie un réel égal à la valeur de I_n. Compilez.
2. Dans un fichier `exo1.f90`, écrivez un programme qui appelle la fonction `approx` pour $n = 10, 100, 1000$ et comparer avec la valeur théorique de l'intégrale [1]. L'erreur en valeur absolue sera imprimée à l'écran.

Exercice 2. Retour sur l'interpolation.

Introduction

Nous avons vu lors du TP6 le phénomène de Runge et une stratégie permettant de limiter cet effet : l'approximations par des fonctions polynomiales par morceau. Nous allons étudier lors de ce TP une autre stratégie, qui consiste à utiliser une interpolation polynomiale sur des points d'interpolation non équirépartis.

On peut montrer que l'erreur d'interpolation est minimale en choisissant les **racines des polynômes de Tchebychev** comme point d'interpolation. Bien qu'il existe une expression analytique de ces racines, nous allons calculer ces racines par **dichotomie**. La difficulté est alors d'identifier au préalable des intervalles dans lequel on sait qu'il n'y a qu'une seule racine. Nous utiliserons alors **la méthode de Sturm**, qui permet de connaître le nombre de racines dans un intervalle donné, pour identifier de tels intervalles.

Ce TP se divise donc en 5 parties :

Partie 1 : Préliminaires : codage de fonctions élémentaires sur les polynômes.

Partie 2 : Introduction et implémentation de la méthode de Sturm.

Partie 3 : Implémentation de la méthode de dichotomie et recherche des racines d'un polynôme.

Partie 4 : Construction des polynômes de Tchebychev et recherche des racines.

1. Pi=3,141592653589793238

Partie 5 : Comparaison des interpolations réalisées avec des points équirépartis et avec les racines du polynôme de Tchebychev correspondant.

1 Préliminaires

Dans le répertoire `TP8`, créer un répertoire `Exo3` .

Dans ce TP, le polynôme $P(x) = \sum_{i=0}^{n} a_i X^i$ sera représenté par un type dérivé `polynome` composé :

- d'un entier `degre` qui représente le degré du polynôme
- d'un tableau de réels `coefficients` à une dimension contenant 101 cases, représentant les coefficients du polynôme (on suppose que le degré du polynôme est au maximum 100).

Exemple : Pour P, l'entier `degre` vaut n et le tableau `coefficients` est

$$(\backslash a_0, a_1, ..., a_n, 0, ..., 0\backslash)$$

Création du type polynômes et première fonctions associées

Dans un fichier `modpoly.f90`, créez un module `modpoly` contenant le type dérivé `polynome`. Tous les sous-programmes suivants sont à écrire dans ce même fichier.

a- Ecrire une subroutine `cree_monome`, qui modifie la valeur d'un coefficient d'un polynôme donné.
- un polynôme `Poly`,
- le degré n du monome à modifier dans `Poly`,
- la valeur du coefficent a_n associé au monome de degré n.

Remarque : si le degré du monome à modifier est supérieur au degré de `Poly`, on augmente la valeur du degré de `Poly`.

b- Ecrire une fonction `copie_pol` qui prend comme paramètres un polynôme A et qui retourne un polynôme B tel que $B = A$.

c- Ecrire une fonction `initialize_pol` qui prend comme paramètre un entier n et qui retourne un polynôme de degré n dont tous les coefficients seront initialisés à 0.

d- Ecrire une subroutine `print_pol` qui prend comme arguments :
- un polynôme,
- une chaine de caractère `ch`,

et qui affiche à l'écran sur une même ligne `ch` le degré du polynôme et ses coefficients (attention à ne pas écrire les 100 réels du tableau `coeff`).

Dans un fichier `prog.f90`, écrire un programme utilisant le module `modpoly` et vérifiez que tous vos sous-programmes tournent correctement.

Opération sur les polynômes

On travaille toujours dans le module `modpoly`.

a- Ecrire une fonction `add_pol` qui prend comme paramètres deux polynômes A et B, et qui retourne un polynôme $C = A + B$. Attention à gérer les cas où les termes de plus au degré s'annulent l'un l'autre.

b- Ecrire sur papier l'algorithme permettant de multiplier deux polynômes.

c- Ecrire de la même façon une subroutine `mult_pol` qui effectue le produit de 2 polynômes.

d- Ecrire une fonction `derivpol` qui dérive un polynome. Elle prendra en paramètre le polynôme et retournera sa dérivée.

Testez tous vos sous-programmes.

Division euclidienne de deux polynômes

a- Ecrivez sur papier l'algorithme effectuant la division euclidienne de deux polynômes

b- En se servant des sous-programmes écrits dans les questions précédentes, écrire une subroutine `divisioneucl` qui effectue la division euclidienne de 2 polynômes. Cette subroutine prend comme paramètres :
- deux polynômes,
- le reste de la division,
- le quotient de la division.

2 Introduction et implémentation de la méthode de Sturm

Méthode de Sturm. Introduisons succinctement la méthode de Sturm de comptage des racines d'un polynôme réel dans un intervalle.

Soit $P \in \mathbb{R}[X]$ un polynôme non constant. On construit alors une suite P_n de polynômes de la manière suivante.

$$\begin{cases} P_1 = P \\ P_2 = -P' \\ \forall k > 2, P_k = -R_k, \text{ tel que } P_{k-2} = Q_k P_{k-1} + R_k \end{cases}$$

P_k est ainsi **l'opposé** du reste de la division euclidienne de P_{k-2} par P_{k-1}. La suite P_n est une suite finie. Soit m le plus petit entier tel que $P_{m+1} = 0$.

On définit alors la suite de Sturm du polynome P comme la suite S_n telle que $\forall k \in [0, m], S_k(x) = \dfrac{P_k(x)}{P_m(x)}$. On définit ensuite, pour tout $x \in \mathbb{R}$, le $m+1$ uplet $S(x) = (S_0(x), S_1(x), ..., S_m(x))$.

On note alors $CS(x)$ le nombre de changements de signe de ce $m+1$ uplet dont on aurait ignoré les zéros. Par exemple, si $S(x) = (-1, 0, 0, 2, 4, -1, 0, 2)$, $CS(x) = 3$.

Le théorème de Sturm s'énonce alors ainsi.

Théorème 1 *Soient $P \in \mathbb{R}[X]$, $(a, b) \in \mathbb{R}^2$ et $r_{[a,b]}$ le nombre de racines de P comprises dans $[a, b]$. Alors*

$$r_{[a,b]} = CS(b) - CS(a)$$

Algorithme et implémentation. Créer un fichier `modsturm.f90` contenant un module `modsturm` dans lequel vous coderez les sous-programmes de ce paragraphe.

1. Ecrire à la main l'algorithme d'Euclide qui permet de construire la suite de Sturm.
2. *Une difficulté d'implémentation de l'algorithme de construction de la suite de Sturm réside dans le fait qu'on ne connaît pas à l'avance le nombre de polynômes dans la suite de Sturm : ceci pose problème au moment d'allouer la mémoire pour le tableau contenant la suite. Une manière*

de résoudre ce problème peut-être de faire deux fois l'algorithme d'Euclide : une première fois "à vide", sans sauvegarder les différents polynômes P_k, $k = 0, ..., m-1$, pour connaître le nombre m et le polynôme final P_m, une deuxième fois après avoir alloué le tableau à la bonne taille, en sauvegardant tous les polynômes d'étape.

   (a) Ecrire une subroutine `sturm_vide` qui effectue l'algorithme de construction de la suite de Sturm sans sauvegarder les polynômes de la suite. Cette subroutine aura pour arguments :
   - un polynôme,
   - un entier qui aura pour la valeur la taille de la suite
   - un polynôme `PM` qui aura pour valeur le polynome P_m de la suite.

   (b) Implémenter l'algorithme de construction de la suite de Sturm dans une subroutine `sturm` qui prendra en arguments :
   - le polynôme considéré,
   - un polynôme `PM` (qu'on supposera calculé avant ce sous-programme),
   - un entier représentant la taille de la suite,
   - un tableau S de polynômes qui contiendra la suite de Sturm à la fin de la subroutine.

3. Ecrire une subroutine `changement_signe` qui prend un tableau de réels et un entier qui contiendra le nombre de changements de signe dans cette liste.
4. Implémenter une fonction `nb_racines` qui prend un polynôme et deux réels a et b en paramètre, et qui retourne le nombre de racines du polynôme comprises dans $[a, b]$.
5. Tester tous ces sous-programmes.

3 Recherche de toutes les racines réelles d'un polynôme par dichotomie

Créer un fichier `modracine.f90` contenant un module `modracine` dans lequel vous coderez les sous-programmes et fonctions de ce paragraphe.

1. En supposant que l'on sache que le polynôme P n'a qu'une racine dans $[a, b]$, écrire une fonction `dichotomie` prenant en paramètre le polynôme P, les bornes de l'intervalle $[a, b]$ et l'erreur d'approximation requise. Cette fonction retourne la racine de ce polynôme dans $[a, b]$ calculée en utilisant la dichotomie.
2. En utilisant le théorème suivant :

   Théorème 2 (Borne de Cauchy) *Soit $P(X) = a_0 + a_1 X + ... + a_n X^n \in \mathbb{R}[X]$, alors, toutes les racines de P sont comprises dans l'intervalle $[-M, M]$, avec $M = 1 + \max_{0 \leq i \leq n} |a_i|$*

   Ecrire une subroutine `separe_racine` qui prend en paramètre d'entrée un polynôme et le nombre r de racines réelles de ce polynôme et comme paramètre de sortie un tableau de dimension $r \times 2$ contenant sur la ligne i les bornes a_i, b_i d'un intervalle $[a_i, b_i]$ contenant comme unique racine la racine i.

4 Construction des polynômes de Tchebychev et recherche des racines

Définition 3 (Polynômes de Tchebychev par récurrence) *La suite* $(T_n)_{n\in\mathbb{N}}$ *des polynômes de Tchebychev se définit par récurrence :*

$$\begin{cases} T_0(X) = 1 \\ T_1(X) = X \\ T_{n+2}(X) = 2XT_{n+1}(X) - T_n(X) \end{cases}$$

Créer un fichier `modtchebychev.f90` contenant un module `modtchebychev` dans lequel vous coderez les sous-programmes et fonctions de ce paragraphe.

1. Ecrire une fonction `tchebychev` qui prend en paramètre un entier n et qui retourne le polynôme T_n.
2. Ecrire une subroutine `racine_tchebychev` qui prend en paramètre
   - un polynome correspondant au polynôme de Tchebychev dont on veut calculer les racines,
   - un entier nr égal au nombre de racines du polynôme (fourni en entrée de la subroutine),
   - un tableau de nr réels contenant les racines du polynôme qui auront été calculées.
3. (facultatif) Ecrire une subroutine `save_racines` qui prend comme paramètre un entier n. Cette subroutine calculera les racines du polynômes de Tchebychev de degré n et les écrira dans un fichier (non binaire) nommé `racine`.

5 Comparaison des différentes interpolations

1. Dans un fichier interp.f90, écrivez un programme qui calcule le polynôme d'interpolation de Lagrange de la fonction $f(x) = \frac{1}{1+25x^2}$ en prenant comme points d'interpolation les racines du polynôme T_5 puis T_{15} et enfin celle de T_{18}. Comme dans l'exercice 3 du TP 6, ce polynôme d'interpolation sera évalué en 101 points équirépartis, ces évaluations seront affichées sauvegardée dans des fichiers nommés `data5`, `data15`, `data18`.

   Afin d'éviter les erreurs de copier-coller, on pourra faire une fonction interne prenant comme argument le degré du polynôme et le nom du fichier et l'appeler 3 fois.
2. Afficher sur un graphique la fonction f, les polynômes d'interpolation calculés au TP 6 avec des points équirépartis, et les polynômes d'interpolation évalués à la question précédente.