CHAPITRE

TP 3

Algorithme de Hoerner

Numéro 1 : puissance

Développez un programme qui permettra à l'utilisateur d'élever à la puissance n un nombre réel. Le programme comprendra dans sa partie principale un appel à fonction qui retournera le résultat $y = x^n$.

Numéro 2 : Construction d'une fonction et sa dérivée

Comment, en modifiant l'algorithme de l'exercice 1, peut-on construire un polynôme d'ordre n à partir d'une suite de coefficients $\{a_i\}$? Nous ferons appel à une table de déclaration

int a[20] ;

permettant de traiter des polynômes d'ordre 19 au maximum.

- Modifiez votre algorithme de l'exercice 1 pour construire le polynôme $P_n(x)$ (voir expression ci-après).
- Expliquez pourquoi le polynôme ne peut être d'ordre supérieur à 19 selon la donne du problème.
- Ajoutez une condition de branchement qui permette d'inclure des exposants négatifs dans l'expression du polynôme.

Lorsqu'une fonction peut être approximée par un plynôme,

$$f(x) \approx P_n(x) = \sum_{i=0}^{n} a_i x^i \ ,$$

il est aussi possible d'évaluer la dérivée de cette fonction par

$$\begin{aligned} f'(x) = \frac{\mathrm{d}P_n(x)}{\mathrm{d}x} &= \sum_{i=0}^{n} i a_i x^{i-1} \\ &\equiv \sum_{i=0}^{n-1} a_i' x^i \end{aligned}$$

c'est-à-dire un deuxième polynôme, d'ordre $n-1$.

- Déterminer les coefficients a_i' en termes des a_i.
- Écrivez un programme pour retourner la valeur d'un polynôme P_n à une valeur de x que donnera l'utilisateur, et sa dérivée, comme indiqué ci-haut. Vous fixerez l'ordre du polynome $n = 10$.
- Mise en pratique : tester votre algorithme sur le polynome de second ordre

  $$P_2(x) = x^2 + 3x + 1$$

  en veillant à initialiser les coefficients $\{a_i\}$ correctement.

  Optionnel : Toujours avec $n = 10$, intégrez votre fonction retournant la factorielle d'un entier (voir TP2) à votre algorithme ci-haut pour évaluer

  $$P_n(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 ... \approx \exp(x)$$

  soit la fonction $f(x) = e^x$. Pourriez-vous généraliser cette méthode pour traiter des fonctions trigonométriques $\sin\theta, \cos\theta$?

Numéro 3 : Générateur de nombres aléatoires

La fonction *rand()* (qui ne prend aucun argument) retourne un entier compris entre 0 et une valeur maximale, $rand_{max}$. Cette valeur $rand_{max}$ est définie dans le fichier stdlib.h. Pouvez-vous localiser ce fichier ? Typiquement $rand_{max} = 2^{31} - 1 = 2147483647$. La suite de nombres retournés par appel successif de rand() est uniformément distribuée dans l'intervalle $[0, rand_{max}]$.

- Développez un algortihme pour générer N nombres aléatoires, compris entre [0,1]. Il faut donc **convertir** l'entier retourné par rand() en flottant avant de normaliser le résultat.
- pour lancer rand(), il convient de l'initialiser grâce à un entier $I_0 > 0$ par la fonction *srand(* I_0 *)*. Il est impératif de choisir I_0 et de taper l'instruction

$$\text{srand}(I_0)\,;$$

avant de lancer rand().

– Faites l'implantation de cet algorithme : le programme source s'appellera *random.c*.
– Une façon de vérifier que votre suite de nombres est bien distribuée entre [0,1] est que la moyenne de tous les nombres tirées doit converger vers 1/2 pour N très grand :

$$\lim_{N\to\infty} \sum_i I_i/N = 1/2$$

Ici nous appellerons I_i le ieme flottant correspondant au ieme tir de rand().

Optionnel Lorsque vous êtes satisfait/e que votre boucle génère bien des nombres aléatoires compris entre [0,1], développez un algorithme pour obtenir le nombre π. Rappelez pour cela les grandes lignes de l'algorithme de Buffon. Combien de tirs N faut-il pour reconnaitre les deux premières décimales de $\pi = 3.14..$?