L3 - Méthodes numériques
TP 2 - Calcul de fonctions et résolution d'équations

Support de TP

Les TPs visent à appliquer de manière concrète les notions vues en cours. Il est donc conseillé de venir avec vos supports de cours ou de les télécharger sur le web.

Un programme C type ainsi que le fichier Makefile permettant de le compiler est disponible sur le site du cours. Pour chacune des parties du TP, vous devez créer un programme "tp2_i.c" où i est le numéro de la section.

Les TPs notés doivent être soumis sur la page Jalon du cours dans les deux semaines suivant le TP. Vous devez soumettre un fichier zip contenant le code **compilable** avec un simple appel à `make` ainsi qu'un court rapport de 2 pages maximum en **PDF** (sans page de titre). Le rapport ne doit contenir aucune sortie terminal ou ligne de code. Vous devez par contre pour chaque section du TP rédiger une réflexion sur les notions utilisées et ce que vous avez appris.

1 Affichage de courbes sous Gnuplot et gestion de vecteurs

Dans ce TP, vous serez amené à tracer des courbes. Vous avez accès pour cela à une interface C vers le logiciel de tracé Gnuplot (`"gnuplot_i.h"`). Un certain nombre de fonctions permettant une utilisation simplifiée de vecteurs est également fourni dans le fichier `"vecutils.h"`.

1. Compiler et exécuter le fichier `tp2_0.c`. Enregistrer ce fichier sous le nom `tp2_1.c` et comprendre la liste d'opérations nécessaire à l'affichage des courbes.
2. Ouvrir le fichier `"vecutils.h"` et comprendre l'utilité de chacune des fonctions. Ajouter des commentaires expliquant leur utilisation potentielle.
3. Coder dans le fichier `"vecutils.h"` les fonctions :

   ```
   double *vsum(double *a, double *b, int n)
   ```

   ```
   double *vsub(double *a, double *b, int n)
   ```

   qui calculent respectivement `c=a+b` et `c=a-b` et retournent un pointeur vers `c`.

   Coder également les fonctions :

   ```
   void vsum2(double *a, double *b, int n, double *c)
   ```

   ```
   void vsub2(double *a, double *b, int n, double *c)
   ```

   qui calculent `c=a+b` et `c=a-b` sans allocation mémoire (on suppose que `c` a été alloué).
4. Modifier le code pour afficher les courbes $y = |sin(x) - x|$ ainsi que $y = 2x - 1$ (avec la fonction `gnuplot_plot_xy`). Penser à utiliser les fonctions de gestion de vecteur de `"vecutils.h"`. Afficher les courbes en échelle linéaire et logarithmique (voir documentation de gnuplot).

2 Complexité algorithmique et temps de calcul

— Coder les fonctions `puissance` et `func` définies dans l'Exercice 5 du cours 1.
— Coder la version avec récurrence de la fonction `func` discutée dans le cours 2.
— Afficher le temps de calcul nécessaire à chacune des fonctions à l'aide des fonctions `tic` et `toc` définies dans le fichier `timing.h"`. Tester des valeurs de `n` allant de 100 à 10000.
— Tracer avec Gnuplot le temps de calcul (retourné par `toq`) pour différents n allant de 100 à 10000 par pas de 100.

3 Résolution d'équations

Dans cette section vous allez calculer la valeur de $\sqrt{2}$. Ceci se fera par la résolution de l'équation suivante

$$f(x) = x^2 - 2 = 0$$

1. Déclarer dans votre code la valeur :

   ```
   #define SQRT2 1.41421356237309504880
   ```

   Imprimer cette valeur avec 16 décimales.
2. Calculer l'erreur entre `sqrt(2)` et la valeur définie ci-dessus. Conclusions concernant la précision de la fonction `sqrt`.
3. Coder la méthode de la dichotomie en prenant $a = 1$ et $b = 2$ pour l'estimation de $\sqrt{2}$. Remplir le long des itérations un vecteur contenant l'erreur $|x_k - \sqrt{2}|$. On prendra un nombre d'itérations max `n=60`.
4. Visualiser l'évolution de l'erreur en fonction des itérations avec `gnuplot`.
5. Coder la méthode de la fausse position en prenant $a = 1$ et $b = 2$ pour l'estimation de $\sqrt{2}$. Remplir le long des itérations un vecteur contenant l'erreur $|x_k - \sqrt{2}|$.
6. Coder la méthode de Newton vue en cours pour l'estimation de $\sqrt{2}$. Remplir le long des itérations un vecteur contenant l'erreur $|x_k - sqrt2|$. On initialisera avec $x_0 = 1$.
7. Visualiser l'évolution de l'erreur en fonction des itérations pour la dichotomie, la fausse position et la méthode de Newton. Comparer leur vitesse de convergence. Combien d'itérations sont nécessaires pour que chaque méthode converge ?
8. Étudier l'effet de l'initialisation x_0 sur la convergence de la méthode de Newton.

4 Calcul de fonction

Dans cette section vous allez coder l'évaluation de la fonction $\exp(x)$. On supposera ici que la fonction $exp(x)$ définie dans `math.h"` retourne le double le plus proche de la valeur exacte.

1. Coder une fonction `exptaylor(x,n)` qui calcule la valeur de $exp(x)$ en utilisant la série de Taylor suivante jusqu'à l'ordre n :

   $$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

2. Tracer l'erreur relative $|exp(x) - exptaylor(x)|/|exp(x)|$ de cette approximation pour $n = 10$ et $n = 100$ sur l'intervalle $[-10, 10]$ en échelle linéaire et log. Que remarquez vous pour les grandes valeurs de x ?
3. Implémenter la réduction de l'intervalle par rapport à $ln(2)$ vue en cours. Utiliser pour cela l'approximation de Taylor à l'ordre 100 autour de 0. Vous pourrez utiliser la fonction puissance vue en cours mais attention à la modifier pour gérer les puissances négatives. La valeur de $ln(2)$ peut être définie par

   ```
   #define LN2 0.69314718055994530942
   ```

4. Tracer l'erreur relative lorsque l'on utilise la réduction de l'intervalle.Votre implémentation est elle précise par rapport à la précision machine ?