Emma TUPINIER
ENER 2 App

15/01/2018

COMPTE-RENDU MÉTHODE NUMÉRIQUE

Le but du TP est de programmer des résolutions numériques de fonctions sous le langage C. Dans un premier temps nous allons faire un programme permettant la création d'une fonction simple et d'écrire ses valeurs dans un fichier texte, puis nous allons programmer la résolution d'équation f(x) par la méthode de dichotomie. Enfin nous allons présenter la programmation de résolution d'équation par la méthode de Newton et de la sécante.

- **Création d'un fonction simple et écriture des valeurs dans un fichier texte :**

Voici le programme que nous avons créé, le programme est écrit en rose et les commentaires en bleu:

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <math.h>  /*Dans cette partie nous appelons les différentes bibliothèques nécessaires à notre programme */

double y,x,a,b,h,i; /*Les différentes variables sont définies*/
double g(double); /*La fonction est définie*/
void write_values(double a, double b, double h); /*La fonction void permet d'appeler la fonction write_values sans en afficher sa valeur */

int main ()

/*Dans cette partie nous demandons à l'utilisateur d'entrer les paramètres nécessaires afin de calculer le couple x et f(x) entre les bornes a et b, avec un pas de h.

a permettra de déterminer le x de départ de la fonction
b le x auquel le programme doit se terminer
write_values(a,b,h) permet de calculer la fonction à l'aide des paramètres précédents */


{
        printf("Entrer la borne inferieur de l'intervalle\n");
        scanf("%lg",&a);
        printf("Entrer la borne superieur de l'intervalle\n");
        scanf("%lg",&b);
        printf("Entrer le pas\n");
        scanf("%lg",&h);




        write_values(a,b,h);
```

```
return EXIT_SUCCESS;
}

/*Dans cette partie nous allons définir la fonction permettant de calculer les valeurs du tableau*/

double g(double x)
{
	return x*x*x-(3*x)+1;
}

/*Ici nous appelons la fonction afin de mettre ses résultats dans un tableau tout en prenant en compte les valeurs que l'utilisateur aura entré.

Ces résultats seront rangés dans un fichier texte et qui sera ouvert directement après l'exécution du programme à l'aide de la commande "FILE *f = fopen("values.txt","w")".

La boucle for permet de lancer le programme tant que les conditions ne sont pas atteintes.  */

void write_values(double a, double b, double h)
{
	double x,y;
	x=a;
	FILE *f = fopen("values.txt","w");
	for(i=a; i<=b; i=i+h)
	{
		y=g(x);
		fprintf (f, "x=%lg,f(x)=%lg\n",x,y);
		x=x+h;
	}
```

- **<u>Programme de résolution par la méthode de dichotomie :</u>**

La méthode de dichotomie permet de résoudre l'équation f(x)=0. Cette méthode est une méthode d'encadrement. En effet, on suppose que f change de signe et donc la fonction f(x) possède forcement un 0. On coupe l'intervalle sur [a,b] et on garde l'intervalle où f(x) change de signe. On obtient un encadrement 2 fois plus petit et on réitère l'opération jusqu'à obtenir x.

Voici le programme que nous avons obtenu :

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <math.h> /*Dans cette partie nous appelons les différentes bibliothèques nécessaires à notre programme

Les intervalles auxquels nous allons couper notre fonctions sont : [-1.9;-1.8], [0.3;0.4], [1.5;1.6] */

double a,b,c,e,iteration; /*Les différentes variables sont définies*/
```

```
double f(double x)        /*La fonction est définie*/
{
        return x*x*x-(3*x)+1;   /* la méthode de dichotomie sera affectée à cette fonction */
}
```

int main (int argc,char ***argv[])** /*Ici nous allons lancer le programme et mettre ses résultats sous forme de tableau. « Int argc » correspond au nombre de paramètres que le programme va utiliser et « char *argv) correspond au tableau de chaîne de caractères contenant les paramètres du programme.

La partie ci-dessous permet de ranger les variables sur le terminal.*/

```
        {
        sscanf(argv[1], "%lg", &a);
        sscanf(argv[2], "%lg", &b);
        sscanf(argv[3], "%lg", &e);

        c = (a+b)/2;
```

/*Nous coupons l'intervalle en 2 ; la boucle while ci-dessous permet de choisir quel intervalle le programme doit prendre */

```
        iteration = 0;

        FILE g = fopen("values_dicho.txt","w");
```

/*Ces résultats seront rangés dans un fichier texte et qui sera ouvert directement après l'exécution du programme.

Dans la boucle while, nous pouvons voir la fonction fabs est la fonction d'arrêt. Tant que b-a > e, nous continuons les itérations.

Si f(a)*f(c) < 0, b prend la valeur de c, ce qui permet de définir notre nouvel intervalle.
De manière analogue, si f(b)*f(c)<0 alors c'est a qui prendra la valeur de c.

```
        while (fabs(b-a) > e) {
                if (f(a)*f(c) < 0) {
                        b = c;
                }
                if (f(c)*f(b) < 0) {
                        a = c;
                }
                c = (a+b)/2;
                iteration = iteration + 1;
        }
```

La dernière partie permet d'afficher les résultats finaux.

```
        fprintf(g, "L'endradrement est [%.10f,  %.10f]\n", a, b);
        fprintf(g, "La solution est %.10f, le test vaut %lg\n", c, f(c));
        fprintf(g, "nombre d'itération : %lg\n", iteration);

        return EXIT_SUCCESS;
}
```

Emma TUPINIER
ENER 2 App

15/01/2018

- **Programme de résolution par la méthode de Newton :**

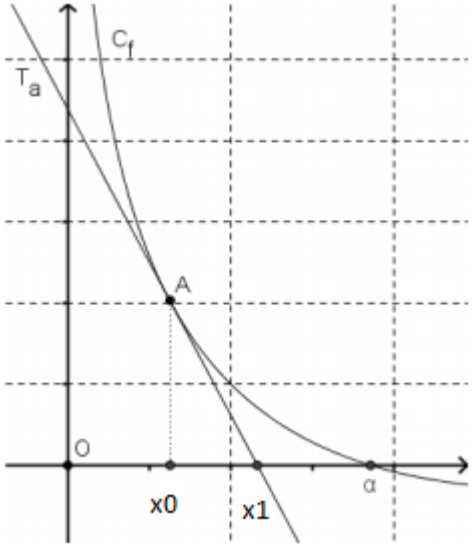
La méthode de Newton est une méthode de résolution de l'équation f (x) = 0.

Soit f une fonction dérivable sur un intervalle I. L'équation f(x)=0 admet une racine unique α sur l'intervalle I.
Soit $x_0 \in I$ une valeur approchée de α. On utilise l'approximation affine g de f au point x_0.

La tangente Ta se calcul à l'aide de la formule : $g(x)=f(x_0)+(x-x_0)f'(x_0)$.

Ainsi Ta coupe l'axe des abscisses au $x_1=x_0-\frac{f(x0)}{f'(x0)}$
x_1 peut être une meilleure approximation de α. On réitère le processus en repartant de x_1 jusqu'à trouver la meilleure approximation de α.

Voici un graphique représentant la méthode :



Graphique représentant la résolution d'équation par la méthode de Newton

Voici le programme permettant cette résolution d'équation :

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <math.h> /*Dans cette partie nous appelons les différentes bibliothèques nécessaires à notre programme

Les intervalles auxquels nous allons couper notre fonctions sont [-1.9;-1.8], [0.3;0.4], [1.5;1.6] */


double f(double x) {     /*Ici nous définissons notre fonction qui aura pour forme *x*x - 3*x + 1 */
        return x*x*x - 3*x + 1;
}
```

```
double df(double x) {  /*Ici nous définissons la dérivé de la fonction précédente, ayant pour forme 3*x*x - 3 */

        return 3*x*x - 3;
}
/*Comme précédemment, la partie ci-dessous permet de lancer le programme et mettre ses résultats sous forme de tableau,*/
int main (int argc,char *argv[]) {
        double x0, x1, epsilon, erreur;
        int iteration;

        sscanf(argv[1], "%lg", &x0);
        sscanf(argv[2], "%lg", &epsilon);

        iteration = 0;
```

/*Ci-dessous nous allons créer un boucle do. Cette boucle permettra de calculer $x_1=x_0-\frac{f(x0)}{f'(x0)}$.
De plus on pose erreur=x0-x1 et tant que erreur > epsilon, on continue les itérations.

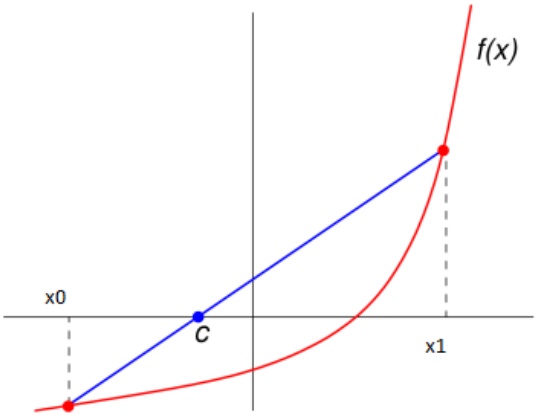
```
        do {
                printf("x0 = %lg\n", x0);
                x1 = x0 - f(x0) / df(x0);
                erreur = fabs(x0-x1);
                x0 = x1;
                iteration = iteration + 1;
        }
        while (erreur > epsilon);

        printf("iteration = %d\n", iteration);

        return EXIT_SUCCESS;
```

- **Programme de résolution par la méthode de la sécante :**

La méthode des sécantes permet de résoudre l'équation f(x)=0 par encadrement de la solution. La méthode est la suivante :



Graphique représentant la méthode de la sécante

On prend 2 point a et b de la fonction, on construit le segment reliant (x0,f(x0)) et (x1,f(x1)).
La droite à pour équation y-f(x1)=$\frac{f(x1)-f(x0)}{x1-x0}$(x-x0)
On choisit x de telle sorte que y=0

Voici le programme qu'on obtient :

```
include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <math.h> /*Dans cette partie nous appelons les différentes bibliothèques nécessaires à notre programme

Les intervalles auxquels nous allons couper notre fonctions sont [-1.9;-1.8], [0.3;0.4], [1.5;1.6]



double f(double x) {   /*Ici nous définissons notre fonction qui aura pour forme *x*x - 3*x + 1 */
        return x*x*x-(3*x)+1;
}


void secante(double x1, double x0, double epsilon) {  /*On appelle la fonction sécante sans en afficher la valeur */

        double x2, erreur;
        int iteration = 0;

        FILE *g = fopen("values_secante.txt","w");  /*Ces résultats seront rangés dans un fichier texte et qui sera ouvert directement après l'exécution du programme.


Ci-dessous nous créons la boucle do. Cette boucle permet de calculer l'équation
0=$\frac{f(x1)-f(x0)}{x1-x0}$(x-x0) + f(x1)
```

```
De même que dans le programme précédent, tant que x1-x0 > epsilon, nous continuons les itérations.
*/

        do {
                x2 = ( ( x0 * f(x1) ) - ( x1 * f(x0) ) ) / ( f(x1) - f(x0) );
                erreur = fabs(x1-x0);
                x0 = x1;
                x1 = x2;
                iteration = iteration + 1;
        }
        while (erreur > epsilon);

        fprintf(g, "L'endradrement est [%.10f,  %.10f]\n", x1, x0);
        fprintf(g, "nombre d'itération : %d\n", iteration);
        printf("nombre d'itération : %d\n", iteration);
}


int main (int argc,char *argv[]) {    /* Ici nous rangeons les résultats sous forme de tableau */

        double x1,x0,epsilon;

        sscanf(argv[1], "%lg", &x0);
        sscanf(argv[2], "%lg", &x1);
        sscanf(argv[3], "%lg", &epsilon);

        secante(x1,x0,epsilon);

        return EXIT_SUCCESS;
}
```

Conclusion :

A l'aide de ce TP nous avons pu programmer plusieurs méthodes de résolution de l'équation f(x)=0. En effet, ces résolutions peuvent être faisables à la main mais à l'aide d'une boucle la résolution se fait plus rapidement.