Résolution d'un système d'équations par la méthode du pivot maximum

Algorithme

Tout d'abord on demande à l'utilisateur d'entrer le nombre d'équations et d'inconnues du système et ensuite les coefficients des inconnues.

***Données** : entiers n et p, respectivement le nombre d'équations à résoudre et le nombre d'inconnues*
***Privées** : entier i compte les lignes, entier j compte les colonnes*

```
afficher(« Entrer le nombre d'équations a résoudre »)
lire(n)
afficher(« Entrer le nombre d'inconnues du système »)
lire(p)
```

***Données** : tableau de réels matrice[n][p] contenant les coefficients ordonnés des équations du système et le second membre des équations (constante d'indice p)*
***Privées** : entier i compte les lignes, entier j compte les colonnes*

```
pour  i de 1 à n
    pour j de p à 0
            afficher(Entrer le coefficient  de x_j de l'équation i)
            lire(matrice[i][p-j])
    fin pour
fin pour
```

Ensuite on recherche dans une ligne son coefficient maximum en valeur absolue ; on repère son indice et on divise cette ligne par ce coefficient. Pour les autres lignes, on annule le coefficient de même colonne (indice) que celui du coefficient maximum en soustrayant au coefficient concerné le coefficient maximum multiplié par une constante appropriée.

Pendant cette étape on peut déjà détecter le cas où le système est impossible à résoudre, si dans une équation on obtient ax=b avec a=0 et b≠0.

***Données** : tableau de réels matrice[n][p]*
***Résultats** : réel max coefficient maximum de la ligne i, entier indicemax son indice*
***Privées** : entier i compte les lignes, entier j compte les colonnes*

```
max<=matrice[i][0]
pour j de 1 à n
    si (valeur absolue(matrice[i][j])<valeur absolue(max))
    alors   max<=matrice[i][j]
            indicemax<=j
    fin si
fin pour
```

***Données** : réel max coefficient maximum de la ligne i, entier indicemax son indice*
***Résultats** : tableau de réels matrice[k][j] contenant les coefficients des équations du système après les opérations sur chaque ligne en fonction de la ligne i*
***Privées** : entier i désigne un ligne ou équation donnée, entier j compte les indices des colonnes (coefficients), entier k compte les lignes*

```
si (max=0 et matrice[i][p]<>0)
    alors afficher(« Le système est impossible à résoudre »)
    sinon
    pour j de 1 à p+1
            matrice[i][j]=matrice[i][j]/max
    fin pour
    pour k de 1 à n
            si k<>i
            alors   pour j de 1 à p+1
                    matrice[k][j]=matrice[k][j]-matrice[i][j]*matrice[k][indicemax]
                    fin pour
            fin si
    fin pour
fin si
```

Ce traitement terminé on obtient une matrice de type (x représente un réel quelconque variable) :

1	x	x	x	x	x	
0	x	x	x	x	x	
0	x	x	x	x	x	
0	x	x	x	x	x	
0	x	x	x	x	x	

dans le cas où la ligne 1 est traitée en premier ; en répétant le traitement sur les autres lignes, on obtient la matrice suivante, de type diagonale :

1	0	0	0	0	x	
0	1	0	0	0	x	
0	0	1	0	0	x	
0	0	0	1	0	x	
0	0	0	0	1	x	

On obtient simplement les inconnues en prenant la valeur x correspondante, son coefficient étant à 1.

Programme en C

Après codage de l'algorithme précédent, on obtient le code suivant :

```
#include <tcconio.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <Windows.h>

float max(float *T,int *indice,int taille)
{
	float max=0 ;
	int  i=0 ;
	for (i=0;i<taille;i++)
	{
		if(fabs((double)T[i])>fabs((double)max))
		 {
			max = T[i] ;
			*indice = i ;
		 }
	}
	return max ;
}
```

//----------fonction max qui cherche le coefficient maximum dans une ligne----------

```
void afficher()
{
	textcolor(YELLOW);
	clrscr();
	printf("\311\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315
\315\315\315\315\315\315\315");
	printf("\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315");
	printf("\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315
\315\315\315\315\315\315\315\315\315\315\315\273\n ");
	printf("\272");

	textcolor(RED);
	printf("     _____  _____  _______________  ____      ___    __  ");
	textcolor(YELLOW);
	printf("\272 \n ");
	printf("\272");
	textcolor(RED) ;
	printf("    /    /  /    / /          / /   /      /   |  /  | ");
	textcolor(YELLOW);
	printf("\272 \n ");
	printf("\272");
	textcolor(RED);
	printf("   /    / /    / /____     ____/ /   /____   /    | /   | ");
	textcolor(YELLOW);
	printf("\272\n ");
	printf("\272");
	textcolor(GREEN);
	printf ("   /    /__/    /     /    /     / ___   / /     |/    | ") ;
```

```
        textcolor(YELLOW) ;
        printf ("\272 \n ") ;
        printf ("\272");
        textcolor(GREEN);
        printf ("  /           /      /    /     /  /__/  /  /   /|   /|  | ") ;
        textcolor(YELLOW) ;
        printf ("\272 \n ") ;
        printf ("\272");
        textcolor(GREEN);
        printf (" /______________/     /_____/     /__________/  /___/ |__/ |__| ") ;
        textcolor(YELLOW) ;
        printf ("\272 \n ") ;
        printf ("\272");
        printf ("                                   ");
        printf ("\272 \n ") ;
        printf ("\272");
        printf ("                                   ");
        printf ("\272 \n ") ;
        printf ("\272");
        textcolor(GREEN);
        printf ("         made by LOURD Rodolphe & ZERIATE Sofiane     ") ;
        textcolor(YELLOW) ;
        printf ("\272 \n ") ;
        printf ("\272");
        textcolor(GREEN);
        printf ("                        Projet LO44        ") ;
        textcolor(YELLOW) ;
        printf ("\272\n ");
        printf ("\272");
        printf ("                                   ");
        printf ("\272 \n ");
        printf
("\310\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\31
5\315\315\315") ;
        printf ("\315\315\315\315\315\315\315\315\315\315\315\315\315\315");
        printf
("\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\315\31
5\315\315\315\315\315\315\274\n ") ;
        printf ("\n ") ;
        }
```

//---------fonction afficher qui affiche l'interface pour l'utilisateur à l'écran----------

```
void division (float *T,float max,int taille )
{
        int i = 0   ;
        while ( i != taille )
        {
                T[i] = T[i]/max ;
                i++ ;
        }
}
```

//---------fonction division qui divise tous les coefficients d'une ligne par le coefficient maximum donné par la fonction max----------

```
void facsoustraction (float *T,float *T1 ,int indice,int taille)
{
        int i ;
        float k=T1[indice];
        for (i=0;i<taille;i++)
        {
                T1[i]=T1[i]-T[i]*k ;
        }
}
```

//----------fonction facsoustraction qui annule les coefficients des lignes dans la même colonne (même indice) que le coefficient maximum d'une ligne traitée----------

```
int main()
{
        float Tab[15][15];
        float maxi ;
        int i ,indice,j,choix=1,n=0,p;
        int g = 0;
        clrscr ()  ;
        afficher () ;
        printf (" Entrez le nombre d'equations a resoudre :\n" ) ;
        scanf ("%d",&n) ;
        clrscr ()  ;
        afficher () ;
        printf ("Entrer le nombre d'inconnues :" ) ;
        scanf ("%d",&p) ;
        clrscr ()  ;
        afficher () ;
        for (i = 0 ; i < n ; i++ )
        {
                printf ("Veuillez saisir les coefficients de la %deme equation\n",i+1) ;
                for ( j = 0 ; j < p ; j++ )
                  {
                        printf ("\t\t coefficient de x%d :",j+1);
                        scanf ("%f",&Tab[i][j]) ;
                }
                 printf ("Entrer le resultat de l equation %d : ",i+1);
                 scanf ("%f",&Tab[i][p]) ;
                 clrscr ()  ;
                 afficher () ;
        }
        for (i = 0 ; i < n; i++ )
        {
                maxi = max (Tab[i],&indice,p);
                if ( maxi == 0 && Tab[i][p]!= 0 )
                        {
                                printf ("Le systeme est impossible" ) ;
                                getchar () ;
                                return -1 ;
                        }
                if (maxi != 0)
                {
                        division (Tab[i],maxi,p+1);
                        for (j=0 ; j < n ; j++ )
                        {
                                if  (i != j)
```

```
                        facsoustraction (Tab[i],Tab[j],indice,p+1);
                }
        }
}
for (i = 0 ; i < n ; i++ )
{
        g=0 ;
        for ( j = 0 ; j < p ; j++ )
        {
                if (Tab[i][j]==1)
                {
                        printf("x%d = ",j+1) ;
                        indice = j ;
                        g++ ;
                }
        }
        if ( g != 0 )
                printf ("%f",Tab[i][p]);
        else
        {
                printf ("%f = ",Tab[i][p]);
        }
        for ( j = 0 ; j < p ; j++ )
        {
                if (Tab[i][j]!=0 && j != indice)
                {
                        if (Tab[i][j]<0 )
                        {
                                printf ( "- %f*x%d ",fabs(Tab[i][j]),j+1) ;
                        }
                        if (Tab[i][j]>0 )
                        {
                                printf ( "+ %f*x%d ",fabs(Tab[i][j]),j+1) ;
                        }
                }
        }
        printf ("\n") ;
}
getchar () ;
return 0;
}
```

//----------programme principal main----------

Application

En utilisant la seconde loi de Kirchhoff, on obtient les trois équations suivantes, soit une pour chaque maille du circuit étudié :

$20-2*(I_1+I_3)-3*I_1-8*(I_1-I_2)=0$
$8*I_1-2*I_2-5=0$
$3*I_1+I_2-7*I_3=0$

Après simplification et adaptation au programme, on obtient :

$13*I_1-8*I_2-2*I_3=20$
$8*I_1-2*I_2=5$
$3*I_1+I_2-7*I_3=0$

Cela nous donne un système de 3 équations à 3 inconnues ; après traitement, les variables I_1, I_2 et I_3 valent :

$I_1 = 0.034$ A
$I_2 = -2.363$ A
$I_3 = -0.323$ A