Théophile Avenel,
Mathis Jamin

SRT3 - Travaux pratiques Méthodes Numérique

TP1 Méthodes numériques : Pivot de Gauss

RÉSUMÉ

TP1 Méthodes numériques : Pivot de Gauss

SOMMAIRE

Théophile Avenel,
Mathis Jamin

Introduction

Dans ce TP, nous allons aborder la résolution numérique des systèmes d'équations linéaires de la forme $A \cdot X = B$ A·X=B en utilisant la méthode de Gauss, avec et sans pivotage. Nous explorerons la précision et la complexité de ces algorithmes, en les mesurant en termes de nombre d'opérations et de temps d'exécution. Ensuite, nous utiliserons Gnuplot pour visualiser les résultats, modéliser la dépendance du temps d'exécution en fonction de la taille du problème, et comparer ces résultats avec les prévisions théoriques. Enfin, nous appliquerons ces méthodes pour résoudre des problèmes complexes en électronique, notamment en traçant le diagramme de Bode du circuit considéré, afin de valider nos résultats numériques avec des simulations effectuées sous LTSpice.

1. Utilisation basique

1) Pour commencer le TP, nous avons compilé le fichier gauss_test_simple.c puis nous l'avons exécuté.

```
sudo bash
gcc -o gauss_test_simple gauss_test_simple.c alglin.c -lm
./gauss_test_simple
```

Figure 1 : Commande pour compiler et exécuter le fichier gauss_test_simple.c

Remarque : Nous avons besoin d'inclure dans gcc le fichier alglin.c, car si nous ne le faisons pas, nous obtenons l'erreur.

```
référence indéfinie vers « alloc_matrix »
(.text+0x11d) : référence indéfinie vers « affiche_prob »
(.text+0x134) : référence indéfinie vers « gauss_piv_part »
```

Figure 2 : Erreur, références, indéfinies

Ces références sont définies dans le fichier via la macro-commande `#include "alglin.h"`.
L'exécution du fichier gauss_test_simple nous donne le résultat suivant :

```
└─# ./gauss_test_simple
Matrice carrée principale, colonne par colonne
Colonne 1
1
2
3
Colonne 2
1
4
9
Colonne 3
1
8
27

Vecteur second membre
1
1
1
La solution est :
1.83333333333
             -1
0.166666666667
Temps d'execution: 0 s et 1 us
```

Figure 3 : Exécution du fichier gauss_test_simple

Comme nous pouvons le constater via la figure 3 ci-dessus, le programme nous affiche la matrice carrée principale en modélisant un exemple de système à trois inconnues [le vecteur second membre], puis nous donne la solution de ce système.

Nous effectuons ensuite une vérification manuelle pour vérifier que la solution donnée est bien correcte.

Pour vérifier manuellement la solution donnée par le programme, nous devons vérifier si les valeurs fournies pour les inconnues x_1, x_2, et x_3 satisferont le système d'équations représenté par la matrice.

La matrice et le vecteur second membre donnés par le programme sont :

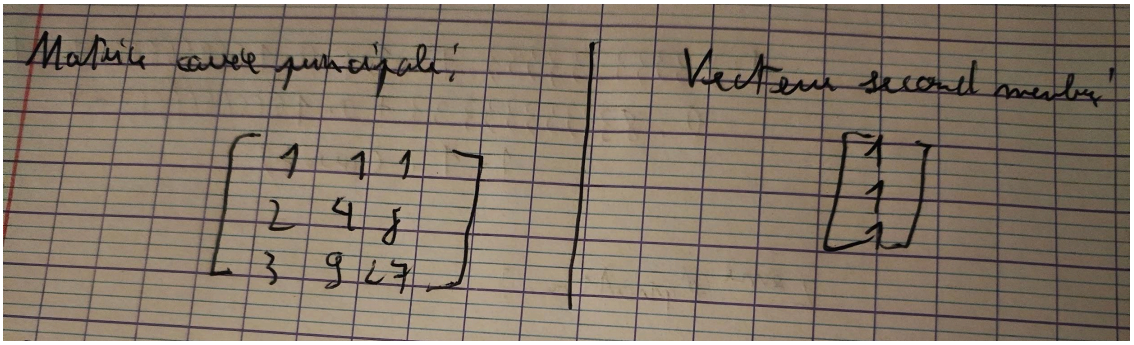


Figure 4 : La matrice carrée principale et le vecteur second membre

Cela représente le système d'équation suivant :

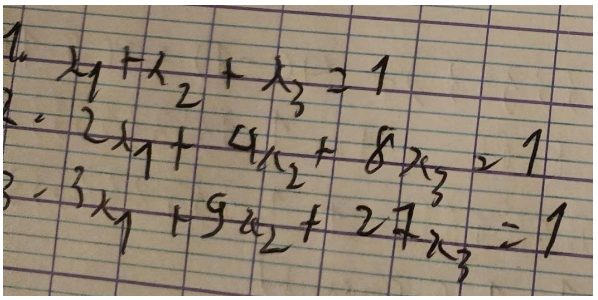


Figure 5 : Système d'équations formé par la figure 4

1. $x_1 + x_2 + x_3 = 1$
2. $2x_1 + 4x_2 + 8x_3 = 1$
3. $3x_1 + 9x_2 + 27x_3 = 1$

La solution donnée par le programme est :

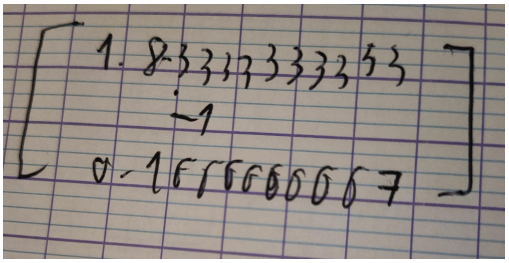


Figure 6 : Solution donnée par le programme au système d'équations figure 5

Pour effectuer la vérification manuelle, nous devons substituer x_1 = 1.83333333333; x_2=-1 et x_3 = 0.16666666667 dans chacune des équations et vérifier qu'elles sont satisfaisantes.

Première équation :

$$x_1+x_2+x_3 = 1$$

$$1.83333333333 + (-1) + 0.16666666667 =1$$

$$1=1 \text{ (Correct)}$$

Deuxième équation :

$$2x_1+ 4x_2+ 8x_3 = 1$$

$$2(1.83333333333) + 4(-1) + 8(0.16666666667) =1$$

$$3.6666666666 -4 + 1.3333333336$$

$$1=1 \text{ (Correct)}$$

Troisième équation :

$$3x_1+ 9x_2+ 27x_3 = 1$$

$$3(1.83333333333)+9(-1)+27(0.166666666667)=1$$

$$5.49999999999-9+4.499999999989=1$$

$$-3.50000000001+4.499999999989=1$$

$$1=1\text{(Correct)}$$

La solution fournie par le programme satisfait toutes les équations du système. La solution est correcte. La solution x=[1.83333333333,−1,0.166666666667] satisfait bien le système d'équations donné par la matrice principale et le vecteur second membre.

2) Nous allons désormais modifier le programme pour résoudre le système A.X = B avec les matrices suivantes :

$$A = \begin{pmatrix} 2 & 1 & 4 \\ 2 & 1 & 0 \\ 0 & 3 & 0 \end{pmatrix} \text{ et } B = \begin{pmatrix} -1 \\ -3 \\ -1 \end{pmatrix}$$

Figure 7 : Implémentation des nouvelles matrices

Nous modifions le programme pour implémenter les nouvelles matrices.

```
// Initialisation de la matrice A
my_mm[0][0] = 2; my_mm[0][1] = 1; my_mm[0][2] = 4;
my_mm[1][0] = 2; my_mm[1][1] = 1; my_mm[1][2] = 0;
my_mm[2][0] = 0; my_mm[2][1] = 3; my_mm[2][2] = 0;

// Initialisation du vecteur B
my_mm[0][N] = -1;
my_mm[1][N] = -3;
my_mm[2][N] = -1;
```

Figure 8 : Initialisation des nouvelles matrices dans le code

```
Matrice carrée principale, colonne par colonne
Colonne 1
2
2
0
Colonne 2
1
1
3
Colonne 3
4
0
0

Vecteur second membre
-1
-3
-1
La solution est :
-1.33333333333
-0.333333333333
         0.5
Temps d'execution: 0 s et 2 us
```

Figure 9 : Exécution du code avec les nouvelles matrices

Comme nous pouvons le constater via la figure ci-dessus, le programme nous affiche la matrice carrée principale en modélisant un exemple de système à trois inconnues [le vecteur second membre], puis nous donne la solution de ce système.

Nous effectuons ensuite une vérification manuelle pour vérifier que la solution donnée est bien correcte.

Pour vérifier manuellement la solution donnée par le programme, nous devons vérifier si les valeurs fournies pour les inconnues x_1, x_2, et x_3 satisfont le système d'équations représenté par la matrice.

La matrice et le vecteur second membre donnés par le programme sont :

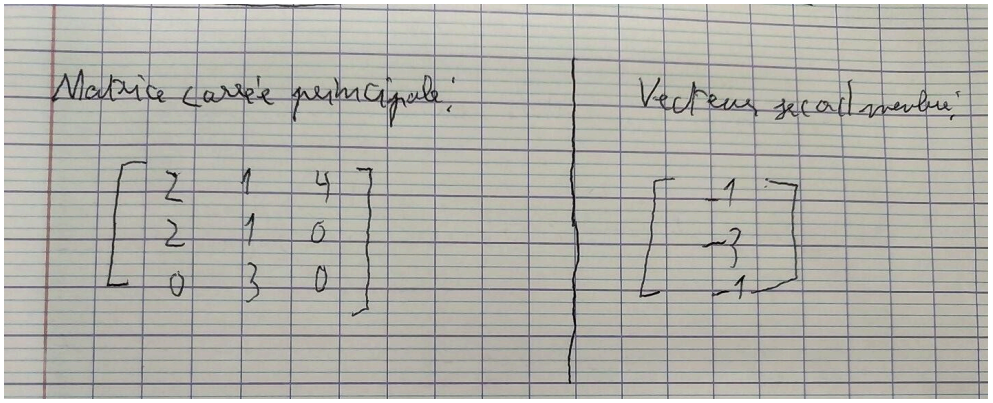


Figure 10 : La matrice carrée principale et le vecteur second membre

Cela représente le système d'équation suivant :

1. $2x_1+x_2+4x_3=-1$
2. $2x_1+x_2+0x_3=-3$ donc $2x_1+x_2=-3$
3. $0x_1+3x_2+0x_3=-1$ donc $3x_2=-1$

La solution donnée par le programme est :

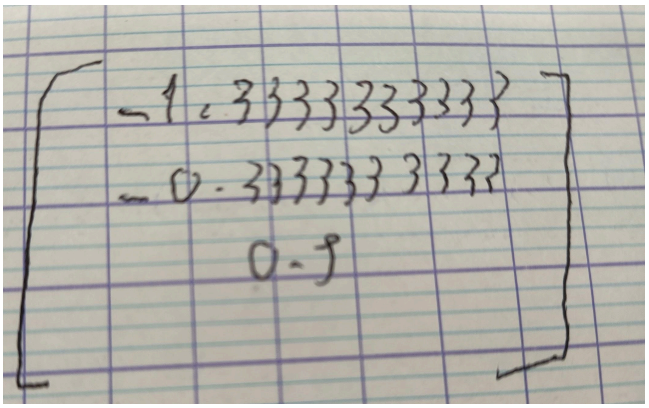


Figure 11 : Solution fournie par le programme

Pour effectuer la vérification manuelle, nous devons substituer x_1 = -1.33333333333 ; x_2=-0.33333333333 et x_3 = 0.5 dans chacune des équations et vérifier qu'elles sont satisfaisantes.

Première équation :

$$2x_1+x_2+4x_3 = 1$$

$$2(-1.33333333333) + (-0.33333333333) + 4(0.5) =1$$

$$-2.6666666666 -0.3333333333 + 2 = 1$$

$$-3 +2 = -1$$

$$-1=-1 \text{ (Correct)}$$

Deuxième équation :

$$2x_1+ x_2 = -3$$

$$2(-1.33333333333) + -0.33333333333 = -3$$

$$-2.6666666666 -0.3333333333 = -3$$

$$-3=-3 \text{ (Correct)}$$

Troisième équation :

$$3x_2 = -1$$

$$3(-0.33333333333)=-1$$

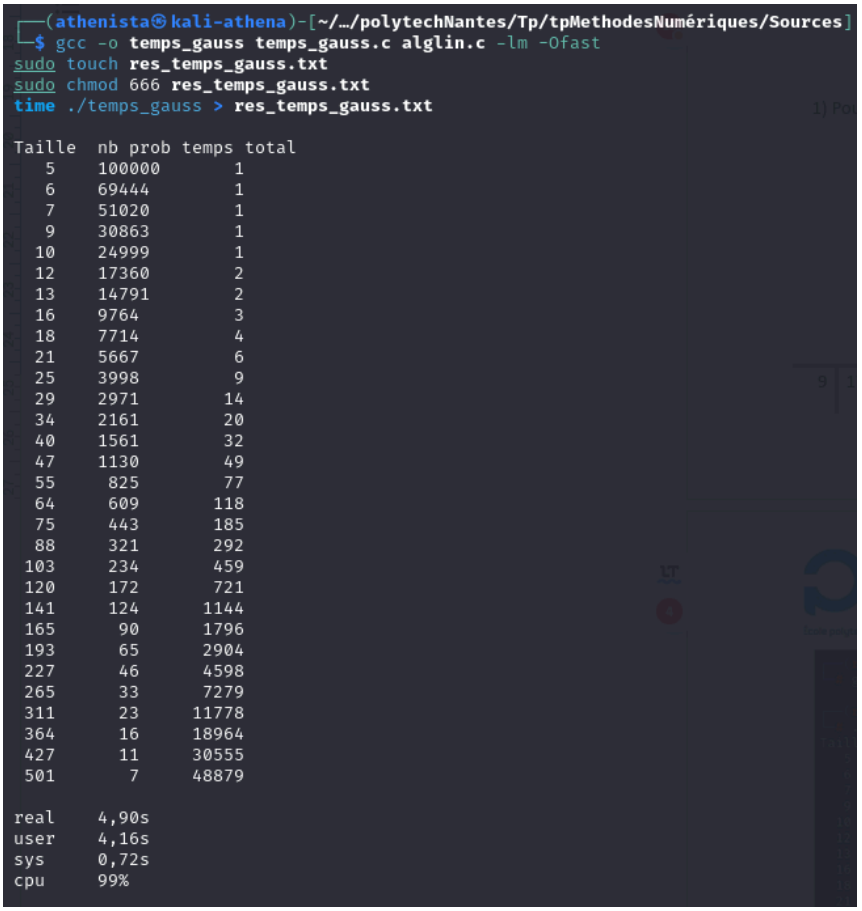
$$-1=-1\text{(Correct)}$$

La solution fournie par le programme satisfait toutes les équations du système. La solution est correcte. La solution x=[-1.33333333333,-0.33333333333,0.5] satisfait bien le système d'équations donné par la matrice principale et le vecteur second membre.

3. Méthode de Gauss

1) Pour compiler le fichier source et l'exécuter, nous allons utiliser les commandes suivantes.

```
gcc -o temps_gauss temps_gauss.c alglin.c -lm -Ofast

sudo touch res_temps_gauss.txt

sudo chmod 666 res_temps_gauss.txt

time ./temps_gauss > res_temps_gauss.txt
```



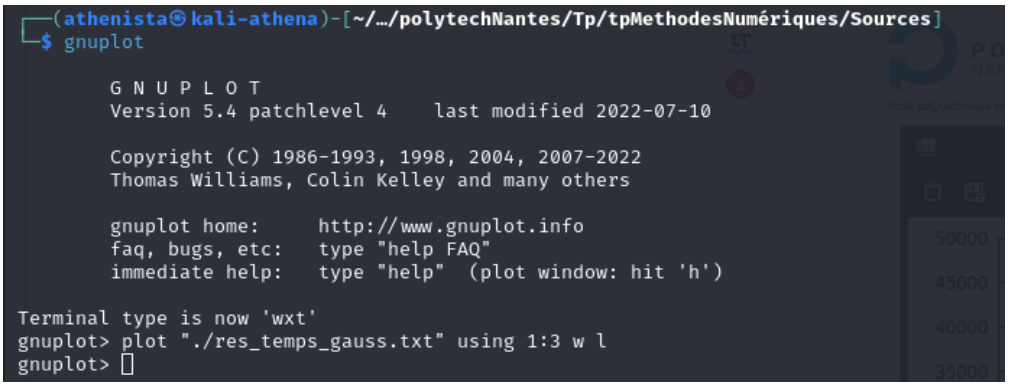
```
┌──(athenista㉿kali-athena)-[~/…/polytechNantes/Tp/tpMethodesNumériques/Sources]
└─$ gcc -o temps_gauss temps_gauss.c alglin.c -lm -Ofast
sudo touch res_temps_gauss.txt
sudo chmod 666 res_temps_gauss.txt
time ./temps_gauss > res_temps_gauss.txt

Taille  nb prob temps total
   5    100000       1
   6    69444        1
   7    51020        1
   9    30863        1
  10    24999        1
  12    17360        2
  13    14791        2
  16    9764         3
  18    7714         4
  21    5667         6
  25    3998         9
  29    2971        14
  34    2161        20
  40    1561        32
  47    1130        49
  55     825        77
  64     609       118
  75     443       185
  88     321       292
 103     234       459
 120     172       721
 141     124      1144
 165      90      1796
 193      65      2904
 227      46      4598
 265      33      7279
 311      23     11778
 364      16     18964
 427      11     30555
 501       7     48879

real    4,90s
user    4,16s
sys     0,72s
cpu     99%
```

Figure 12 : Compilation et exécution des fichiers temps_gauss.c et alglin.c

2) Nous allons ensuite lancer gnuplot pour tracer la dépendance du temps moyen en fonction de la taille.

gnuplot

plot "./res_temps_gauss.txt" using 1:3 w l



```
┌──(athenista㉿kali-athena)-[~/…/polytechNantes/Tp/tpMethodesNumériques/Sources]
└─$ gnuplot

        G N U P L O T
        Version 5.4 patchlevel 4    last modified 2022-07-10

        Copyright (C) 1986-1993, 1998, 2004, 2007-2022
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'wxt'
gnuplot> plot "./res_temps_gauss.txt" using 1:3 w l
gnuplot>
```

Figure 13 : Compilation et exécution des fichiers temps_gauss.c et alglin.c

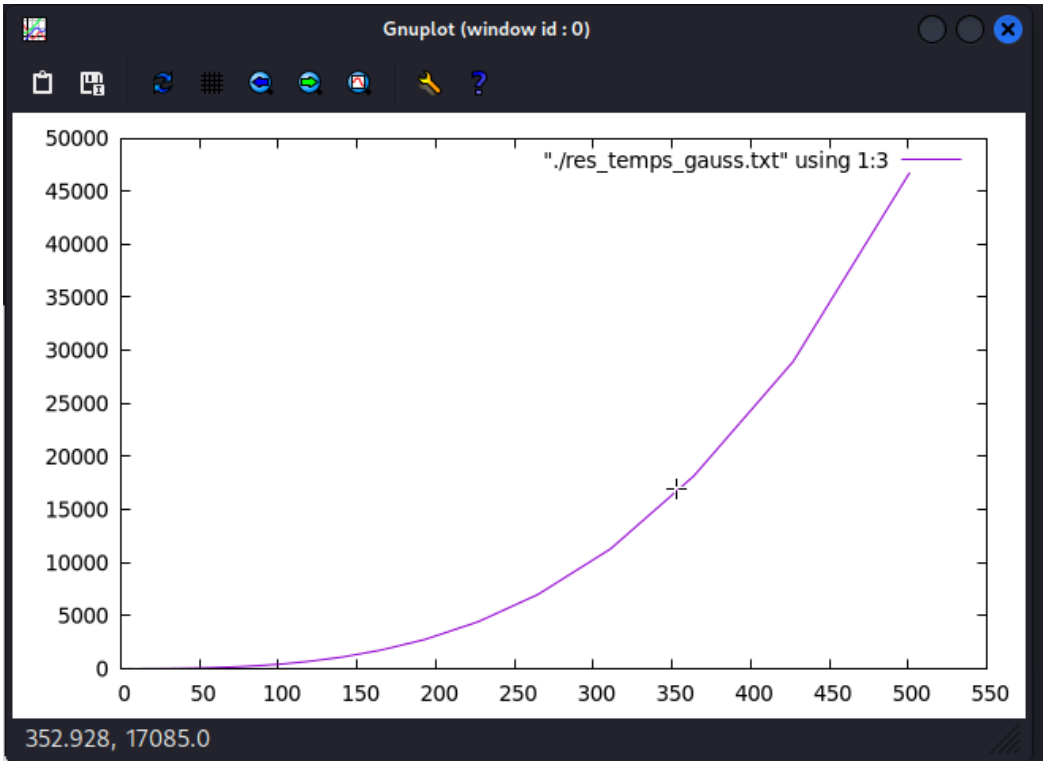


Figure 14 : Schéma gnuplot traçant la dépendance du temps moyen en fonction de la taille

3) Pour déterminer la dimension (ou l'unité) de α, nous allons utiliser le raisonnement suivant :

L'algorithme de Gauss pour résoudre un système linéaire $A \cdot X=B$ a une complexité cubique en termes de nombre d'opérations, soit $O(n^3)$. Cette complexité signifie que le nombre total d'opérations nécessaires pour résoudre un système de taille n est proportionnel à n^3. Pour l'équation de base : $T_e=\alpha n^3$,Te désigne le temps d'exécution en secondes, n représente la taille du système (sans unité), et α est le coefficient à déterminer.

Pour que l'équation $T_e=\alpha n^3$ reste cohérente au niveau des unités, αn3 doit avoir les mêmes unités que T_e, donc les secondes (s).

Vu que n^3 est sans unité, les unités de α doivent être les mêmes que celles de T_e donc des secondes (s).

Ainsi, en écrivant les unités explicitement : [Te]=s ; [n]= sans unité, donc $[n^3]$= sans unité.

Ainsi, pour que αn^3 ait l'unité des secondes, α doit avoir l'unité des secondes divisée par n^3.

$[\alpha]=s/n^3$

Donc, α quantifie le temps nécessaire pour une opération unitaire de l'algorithme par rapport à la taille du problème.

4. Afin de tester le modèle précédent, nous allons définir une fonction sous Gnuplot :

f(x) = a*x*x*x

fit f(x) "res_temps_gauss.txt" using 1:3 via a

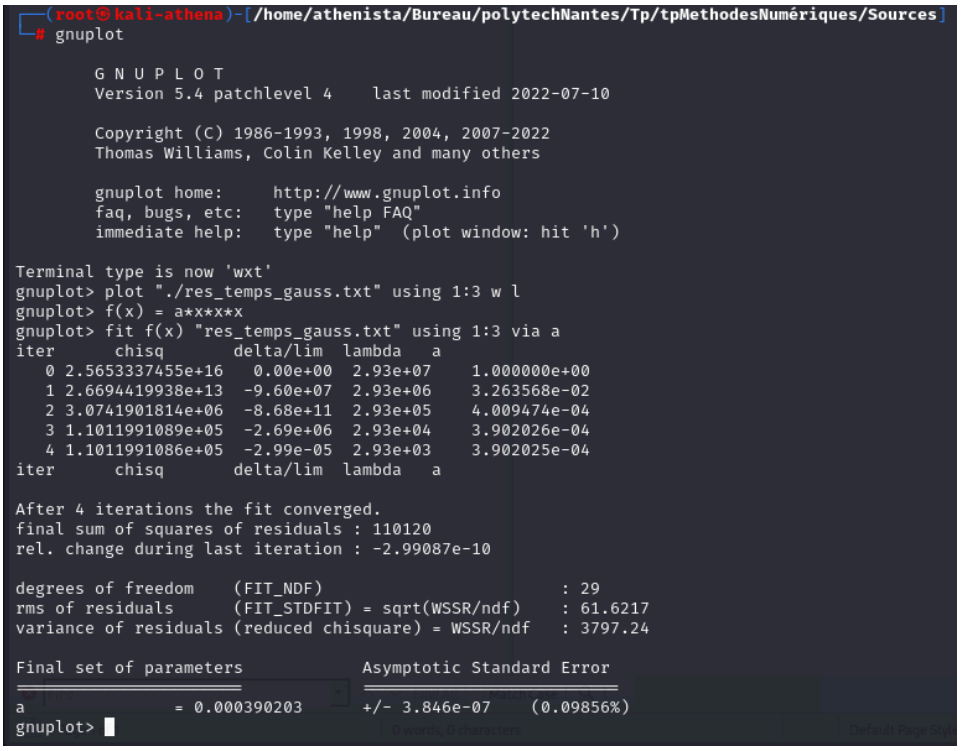


Figure 15 : Test du modèle précédent avec une fonction cubique pour obtenir valeur de α (désigné par a sous Gnuplot)

Le résultat obtenu ci-dessus, montre l'ajustement des données à la loi $f(x)=a \cdot x^3$ avec Gnuplot ce qui a permis de déterminer le paramètre a à 0.000390203. Cette valeur signifie que le temps d'exécution moyen T_e pour résoudre un système linéaire de taille n est proportionnel à n^3. Le coefficient a s'exprime en secondes/n^3. Le calcul de l'ajustement a convergé après quatre itérations, avec une somme des carrés des résidus de 110120, indiquant une bonne approximation du modèle aux données. Le RMS (Root Mean Square) des résidus est de 61.6217, représentant l'écart-type des résidus ce qui permet de quantifier la dispersion des résidus, et la variance des résidus est 3797.24, montrant la variabilité des écarts.

5) En utilisant le modèle obtenu, donnez le temps d'exécution moyen pour un problème de taille 10000, exprimé en heures, minutes et secondes (pensez que Gnuplot permet aussi d'effectuer des calculs...).

Pour un problème de taille n=10000, le temps d'exécution estimé est $Te=0.000390203 \cdot 10000^3=0.000390203 \cdot 10^{12}=3.90203\times10^8$ secondes. Converti en heures, cela donne $3.90203\times10^8/3600\approx108389$ heures, soit environ 4516 jours.

4. Calculs complexes en électronique

1) Nous allons commencer par compléter le code en ajoutant les lignes nécessaires là où c'est indiqué dans la boucle principale, pour initialiser la matrice complexe.

```
59      // Boucle principale, pour les diverses fréquences
60      for(k=0; k<nb_pt; k++) {
61          // On commence par l'initialisation du problème, la matrice étant creuse on fait d'abord W=0
62          for(i=0; i<N; i++) {
63              for(j=0; j<N+1; j++) {
64                  (W[i])[j] = zero;
65              }
66          }
67          // On continue par les 24 affectations nécessaires
68          // placez ici le code d'initialisation du problème
69
70          // Initialisation du problème
71          REAL omega = DEUXPI * f[k];
72          W[0][0] = un; W[0][3] = un; W[0][4] = moins_un;
73          W[1][1] = un; W[1][2] = un; W[1][4] = un;
74          W[2][5] = un; W[2][6] = moins_un;
75          W[3][8] = un; W[3][9] = moins_un;
76          W[4][6] = un; W[4][7] = moins_un; W[4][8] = moins_un;
77          W[5][1] = un; W[5][5] = complex(-RE, 0.0);
78          W[6][2] = un; W[6][6] = complex(0.0, -L1 * omega);
79          W[7][3] = un; W[7][8] = complex(0.0, -L2 * omega);
80          W[8][4] = un; W[8][7] = complex(0.0, 1/ (C1 * omega));
81          W[9][0] = un; W[9][9] = complex(-RS, 0.0);
82
83          // Initialisation du vecteur B
84          W[1][N] = _E_;
85
86          //cplx_affiche_prob(10,W);
87
88          // Le système est prêt pour la resolution
89          if (cplx_gauss_piv_tot(N, W, &my_time) == -1) {
90              // matrice singulière, on passe à la fréquence suivante
91              continue;
92          }
93          // Affichage des résultats: fréquence puis pour chaque inconnue, sous la forme module et arguments
94          printf("%8.8e\t", f[k]);
95          for(i=0; i<N-1; i++){
96              //printf("Fréquence", "Module", "Phase");
97              printf("%8.8e\t%8.8e\t", 20*log10(cmod((W[i])[N])), carg2((W[i])[N])*360/DEUXPI);
98          }
99          printf("%8.8e\t%8.8e\n", 20*log10(cmod((W[N-1])[N])), carg2((W[N-1])[N])*360/DEUXPI);
100     }
101     free(W);
102     exit(0);
103 }
```

Figure 16 : Initialisation de la matrice complexe dans le code

Pour compiler le programme, nous allons utiliser la commande suivante :

gcc -o elec_calc elec_calc.c cplx_alglin.c -lm -Ofast

Exécuter le programme, nous allons utiliser la commande suivante :

time ./elec_calc 1e2 1e8 5000 > elec_calc_results.txt

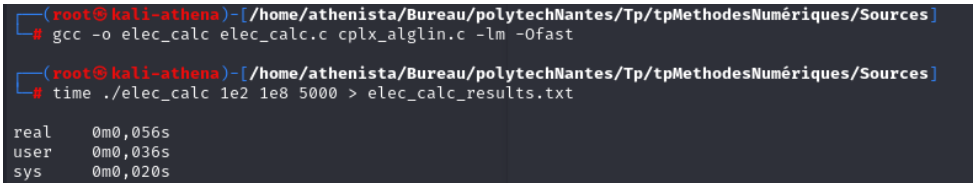


Figure 17 : Compilation et exécution de code

Le programme ./elec_calc 1e2 1e8 5000 a pris 0.056 seconde en temps total (real) pour s'exécuter, utilisant 0.036 seconde de temps CPU utilisateur (user) et 0.020 seconde de temps CPU système (sys).

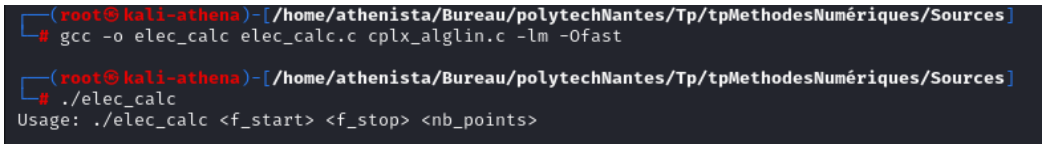


Figure 18 : Paramètres nécessaires au code elec_clac.c

Le fichier de code elec_calc.c requiert trois paramètres : La fréquence de début (<f_start>), la fréquence de fin (<f_stop>) et le nombre de points.

2) **Nous allons commencer par tracer le diagramme de Bode avec Gnuplot** :

set logscale x 10

plot "elec_calc_results.txt" using 1:2 w l title "Module"

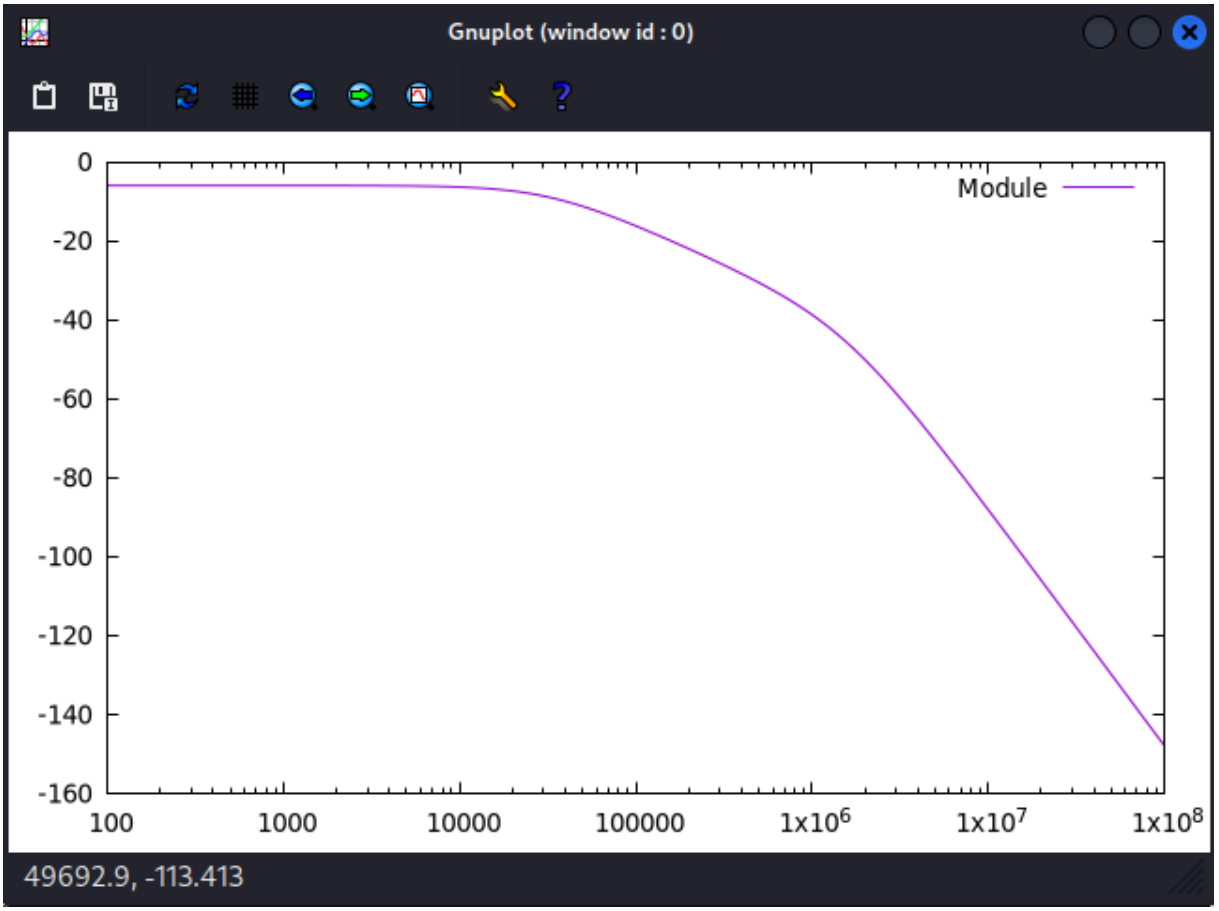


Figure 18 : Représentation du module

plot "elec_calc_results.txt" using 1:3 w l title "Phase"

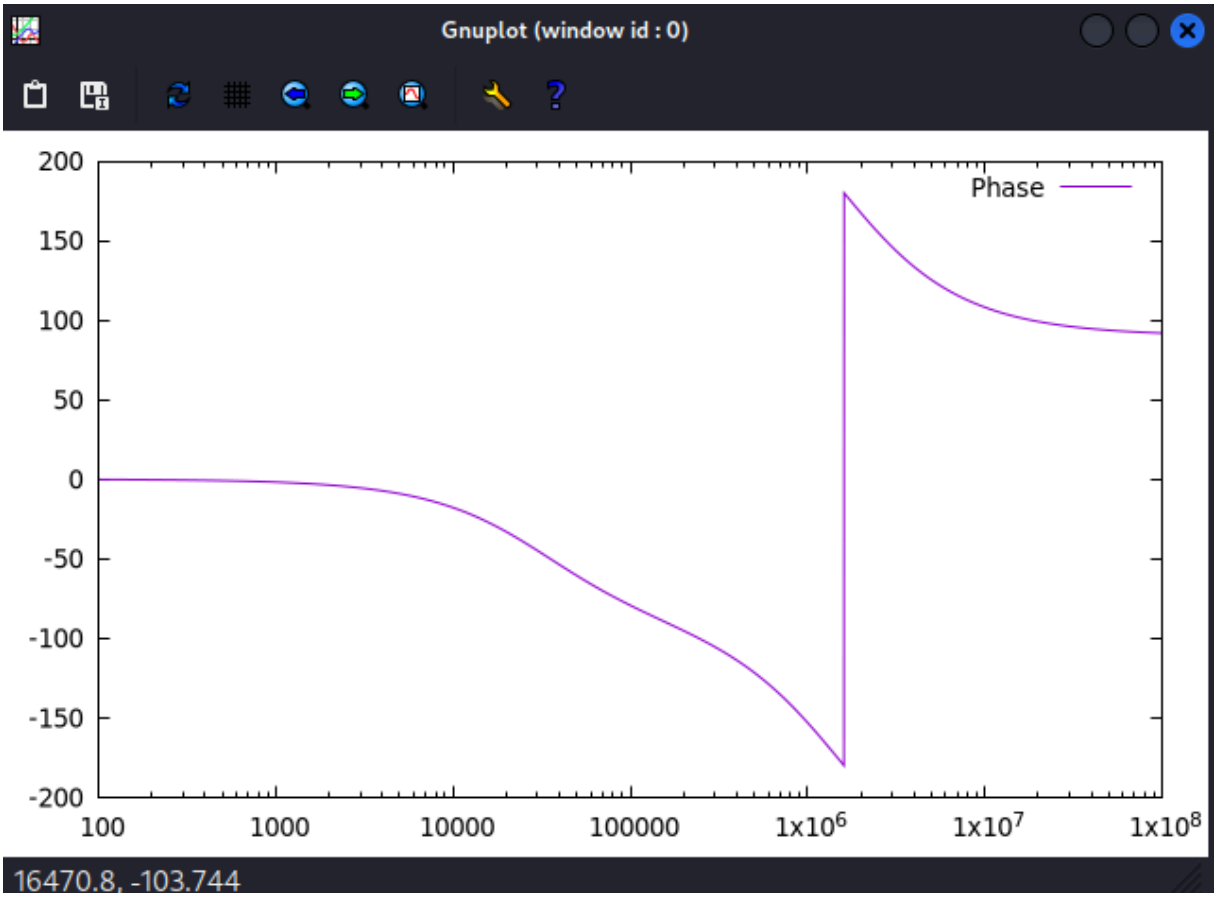


Figure 19 : Représentation de la phase

3 a) Analyse qualitative de la fonction de transfert en basses et hautes fréquences

Basses fréquences :

- À basses fréquences, les condensateurs C1 se comportent comme des circuits ouverts (impédance élevée).
- Les inductances L1 et L2 se comportent comme des courts-circuits (impédance faible).

Dans ce cas, le courant ne passe pas à travers les condensateurs. L'impédance des inductances est faible, ce qui permet au signal de passer plus facilement.

Hautes fréquences :

- À hautes fréquences, les condensateurs C1 se comportent comme des courts-circuits (impédance faible).
- Les inductances L1 et L2 se comportent comme des circuits ouverts (impédance élevée).

Dans ce cas, le courant passe facilement à travers les condensateurs, tandis que les inductances bloquent le passage du courant en raison de leur impédance élevée.

3b) Calcul de la formule littérale de la fonction de transfert

Pour calculer la fonction de transfert) H(ω) du circuit, nous devons exprimer la tension de sortie Vs en fonction de la tension d'entrée E.

1. **Déterminer les impédances :**
   - Impédance du condensateur C1 : ZC1=1/jωC1
   - Impédance de l'inductance L1 : ZL1=jωL1
   - Impédance de l'inductance L2 : ZL2=jωL2Z
   - Impédance de la résistance RS : ZRS=RS
2. **Expression de la fonction de transfert :**
   H(ω)=E(ω)/Vs(ω)

3) Nous traçons le diagramme de Bode du circuit sous LTSpice. Nous pouvons constater sur la figure 21 que cela représente un filtre passe bas. Il laisse passer les basses fréquences et atténue les hautes fréquences comme sur le diagramme de bode de GNUPLOT.

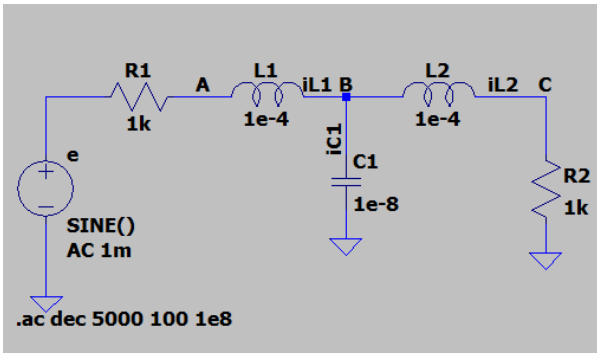


Figure 20 : schéma électronique

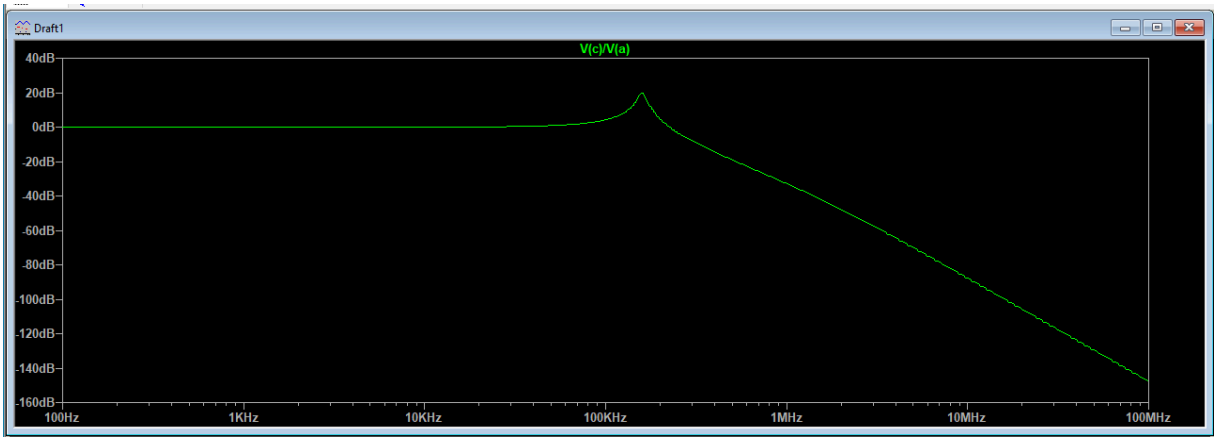


Figure 21 : diagramme de bode

Conclusion

Ce TP nous a permis d'explorer et de comprendre les méthodes numériques de résolution des systèmes d'équations linéaires, en mettant en pratique l'algorithme de Gauss avec différentes stratégies de pivotage. Grâce à l'utilisation de Gnuplot, nous avons pu visualiser et analyser la complexité et la précision de ces méthodes. En appliquant ces techniques à un problème complexe en électronique, nous avons validé nos résultats théoriques avec des simulations pratiques, illustrant ainsi l'importance de ces méthodes dans les calculs complexes nécessaires à la simulation de circuits analogiques.