

Projet de programmation C++

Résolution de circuit

JÉRÉMIE FOURMANN (Promo 2013 - Électronique - Enseeiht)

MAXIME MORIN (Promo 2013 - Électronique - Enseeiht)

30 décembre 2011

Plan

1	Objectif	2
2	Organisation du code	3
2.1	Le concept général	3
2.2	L'objet circuit	3
2.3	L'objet source	4
2.4	Le programme principal (main)	4
3	Résultats	5
3.1	Exemple 1	5
3.2	Exemple 2	5
3.3	Réponse du CircuitA	6
3.4	Réponse du CircuitB	7
3.5	Réponse du CircuitC	8
3.6	Réponse du CircuitD	9
A	Listing du programme	10
A.1	main.cpp	10
A.2	circuits.h	11
A.3	circuits.cpp	13
A.4	sources.h	17
A.5	sources.cpp	18
B	Script de tracé des courbes et Gnuplot	20
B.1	collecte.sh	20
B.2	aTrace	20

1 Objectif

Nous devons réaliser un programme en C++ permettant de résoudre des équations différentielles du 1^{er} et 2^e ordre à coefficients constants. Nous allons utiliser une méthode de résolution numérique de type différences finies. Nous utiliserons plus particulièrement la méthode d'Euler.

L'utilisateur du programme pourra via un terminal :

- choisir le type de circuit (1 ou 2 ordre)
- choisir les valeurs de ces composants
- choisir le type de source (c'est à dire le second membre de l'équation différentielle)

remarque : Le pas et la durée de la simulation sont réglés par défaut. L'utilisateur n'y a pas accès. Nous n'avons pas voulu implémenter cette fonctionnalité pour ne pas surcharger le programme, d'autant plus qu'elle n'apporte pas plus de concept orienté objet.

Le programme écrira dans le terminal ou dans un fichier formaté la solution numérique trouvée. Il sera alors possible de la tracer (voir annexe Gnuplot).

Nous allons à présent détailler et expliquer la structure de notre programme et les choix que nous avons faits. Puis nous commenterons les résultats obtenus sur les divers circuits. Une fois notre code opérationnel nous regarderons les limites de la méthode d'Euler.

En annexe nous détaillons notre méthode pour obtenir et tracer les résultats via un script élémentaire unix et le programme Gnuplot.

2 Organistion du code

2.1 Le concept général

Nous devons nous familiariser avec les notions de classe, héritage, polymorphisme .
En effet en fonction du type de circuit la méthode de résolution n'est plus la meme, nous faisons donc appel a la notion de polymorphysme pour résoudre se problème. Dans notre cas les fonctions circuitSolve() et diffsolve() auront plusieurs versions possible en fonction du circuit.

2.2 L'objet circuit

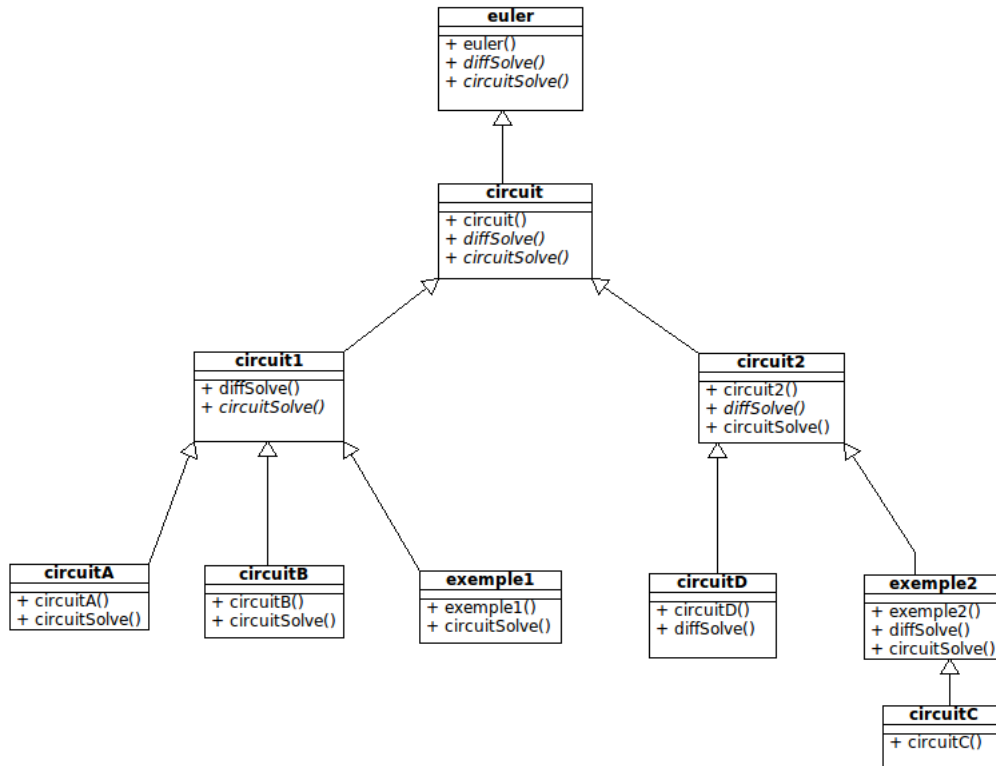


FIGURE 1 – Hièrarchie de la classe circuit

Les principales caractéristique de cet objet :

La classe Euler :

- permet de définir les paramètres de simulation
- 2 méthode circuitSolve() et diffSolve() en virtuelles qui permettrons la résolutions du problème en fonction de la classe instanciée

La classe Circuit :

- Hérite d'Euler
- Son constructeur permet le choix de la Source

La classe Circuit1 :

- Hérite de circuit
- définition de la fonction circuiSolve(),qui permet de calculer résoudre au'+bu=E

La classe Circuit1 :

- Hérite de circuit
- définition de la fonction circuiSolve(),qui permet de calculer résoudre au''+bu'+cu=E

Les autres classes, héritant de ces 2 dernières, permettent de résoudre les circuits A,B,C,D, nous avons essayé de relier les exemples dans le cas général en considérant leurs seconds membres comme des sources particulières.

2.3 L'objet source

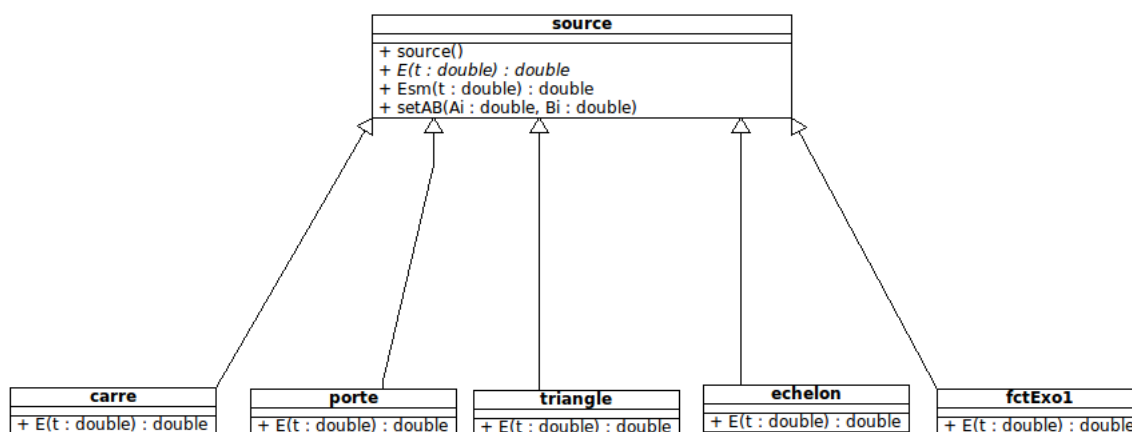


FIGURE 2 – Hiérarchie de la classe source

Les classes carré, échelon... héritent de la classe mère source. Elles définissent chacune une version de la fonction $E(t)$ et règlent certains attributs qui leur sont propres comme l'amplitude, fréquences, offset La fonction $E(t)$ renvoie la valeur à l'instant t de la source. L'objet Circuit va avoir comme attributs une source (ou plutôt un pointeur sur une source), car elle intervient dans la définition de l'équation différentielle à résoudre (second membre).

2.4 Le programme principal (main)

Notre *main* crée un pointeur vers un type de montage et l'initialise selon le choix de l'utilisateur. Il exécute ensuite la méthode de résolution et d'affichage de la solution du circuit qui est `diffsolve()`.

Remarque : La seule sortie du programme est *stdout*. Pour alléger au maximum le programme, nous n'avons pas voulu utiliser une sortie supplémentaire (vers un fichier via *fprintf* par exemple). Lorsque nous voulons collecter les données de sortie du programme dans un fichier nous utilisons la syntaxe Unix : `./programme > fichier.txt`

3 Résultats

3.1 Exemple 1

Résolution de l'équation différentielle du 1^{er} ordre :

$$\begin{cases} u'(t) = -3 \cdot u(t) - 3 \cdot t \\ u(0) = 0 \end{cases}$$

La solution exacte étant $u(t) = -1/3 \cdot \exp(-3t) - 1/3$

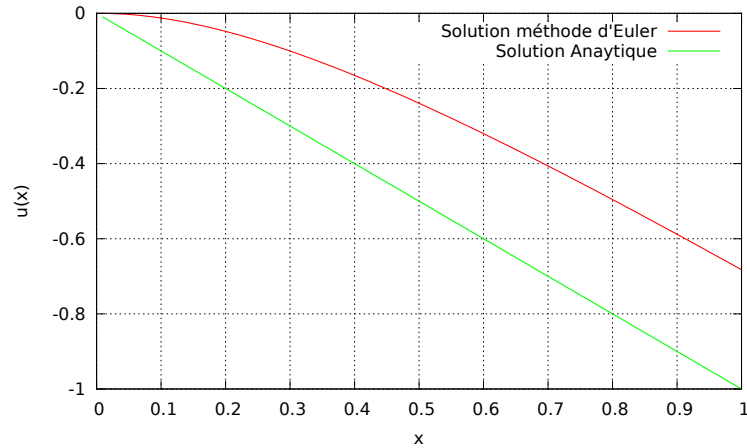


FIGURE 3 – Solution de l'exemple 1

3.2 Exemple 2

Résolution de l'équation différentielle du 1^{er} ordre :

$$\begin{cases} u''(t) = -\lambda \cdot u(t) \\ u(0) = 0 \\ u'(0) = 1 \end{cases}$$

La solution exacte étant $u(t) = \sin(t)$

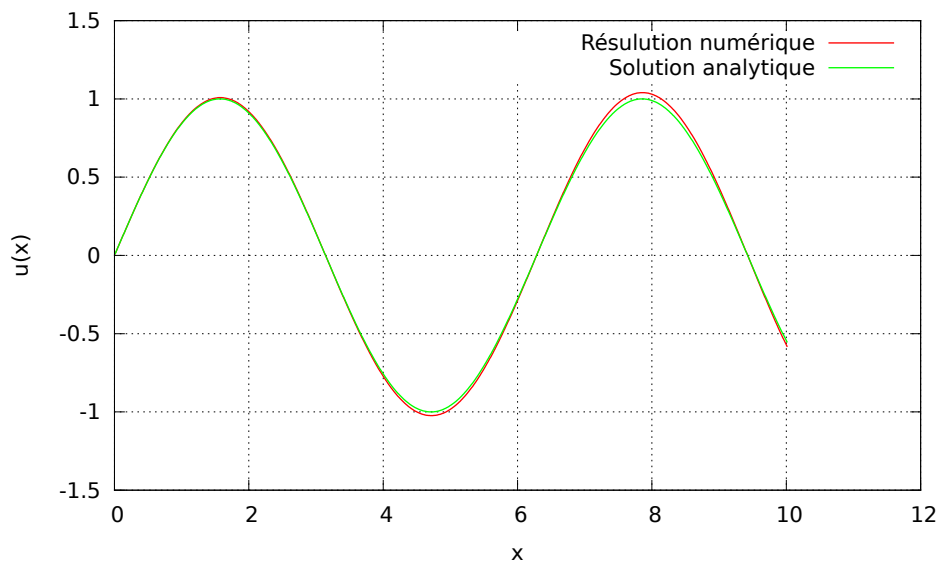


FIGURE 4 – Solution de l'exemple 2

3.3 Réponse du CircuitA

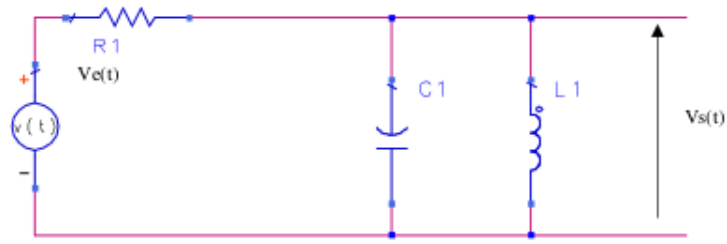


FIGURE 5 – Circuit A

Le circuit A est un circuit RC du 1^{er} ordre, régit par l'équation différentielle :

$$Vs + RC \cdot Vs' = Ve$$

Nous allons étudier la réponse du circuit A à plusieurs type d'excitation, avec comme paramètres de simulation :

- R=1 et C=1
- durée= ,durée=

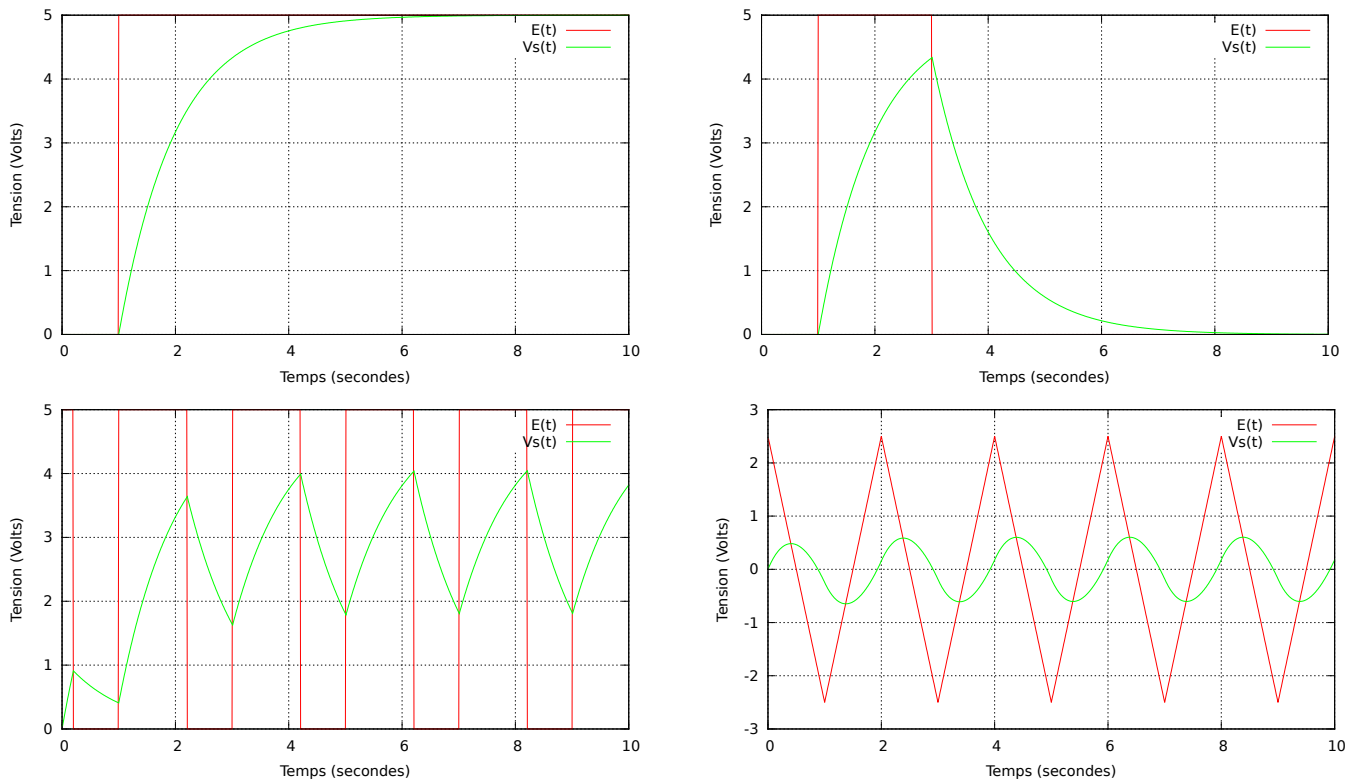


FIGURE 6 – Réponse du circuit A

3.4 Réponse du CircuitB

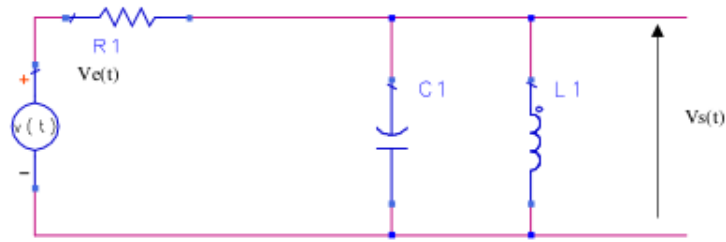


FIGURE 7 – Circuit B

Le circuit B est un circuit RC du 1^{er} ordre, il n'est pas linéaire à cause de la diode, nous devons donc l'étudier dans 2 cas .

1er cas : diode passante ($V_d > 0.6$), on a une équation de charge de la capacité .

2ième cas : diode bloqué ($V_d < 0.6$), on a une équation de décharge de la capacité dans la résistance.

Notre programme teste la condition de V_d pour savoir quelle équation il doit résoudre, par ailleurs ici on doit prendre en compte les conditions initiales.

Car lors d'un changement d'équation, nous avons des conditions "initiales" différentes .

Nous allons étudier la réponse du circuit B à plusieurs types d'excitation, avec comme paramètres de simulation :

- $R=1$ et $C=1$
- durée= , durée=

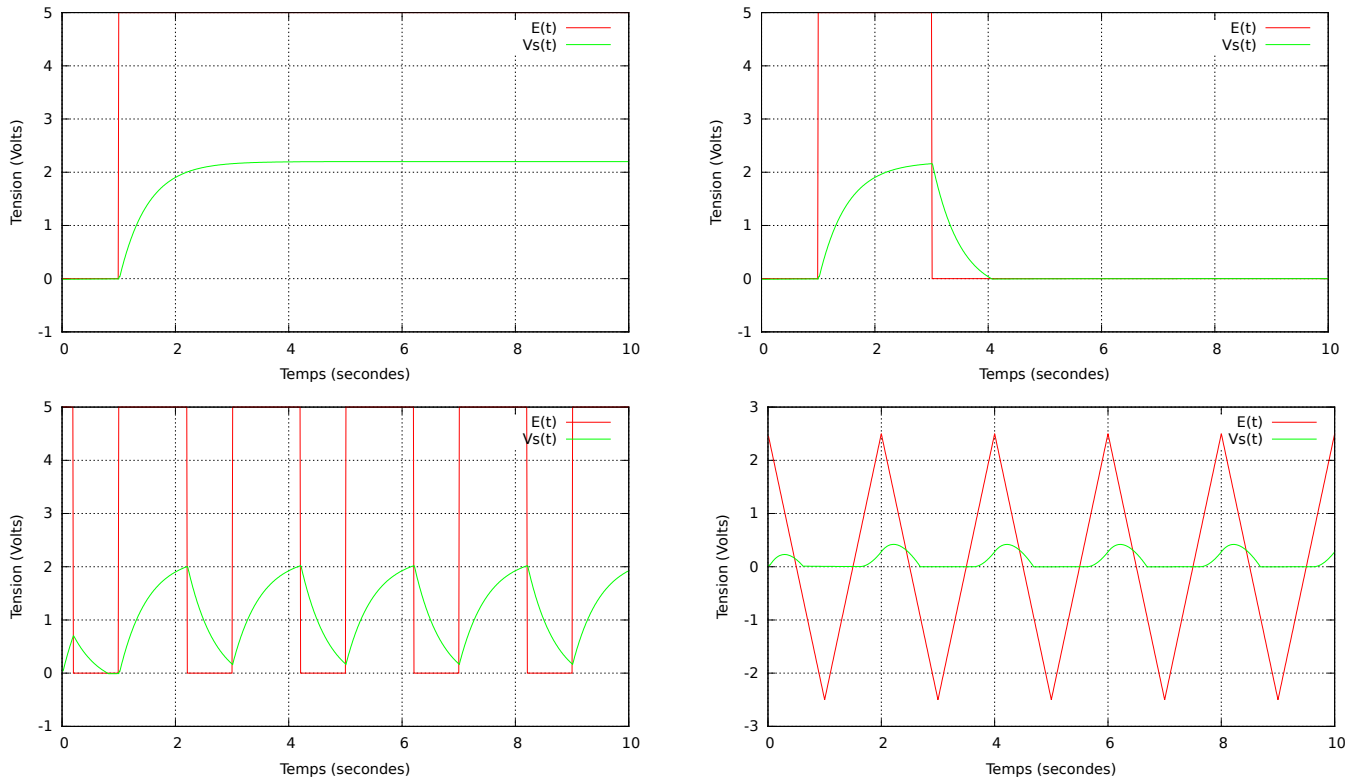


FIGURE 8 – Réponse du circuit B

3.5 Réponse du Circuit C

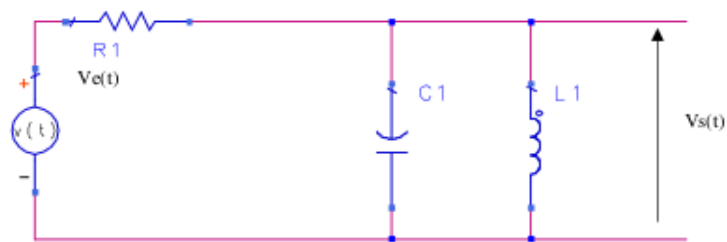


FIGURE 9 – Circuit A

Le circuit C est un circuit RLC du 2^e ordre, régit par l'équation différentielle :

Todo

Nous allons étudier la réponse du circuit C à plusieurs types d'excitations, avec comme paramètres de simulation :

- $R=1$ et $C=1$
- pas= , durée=

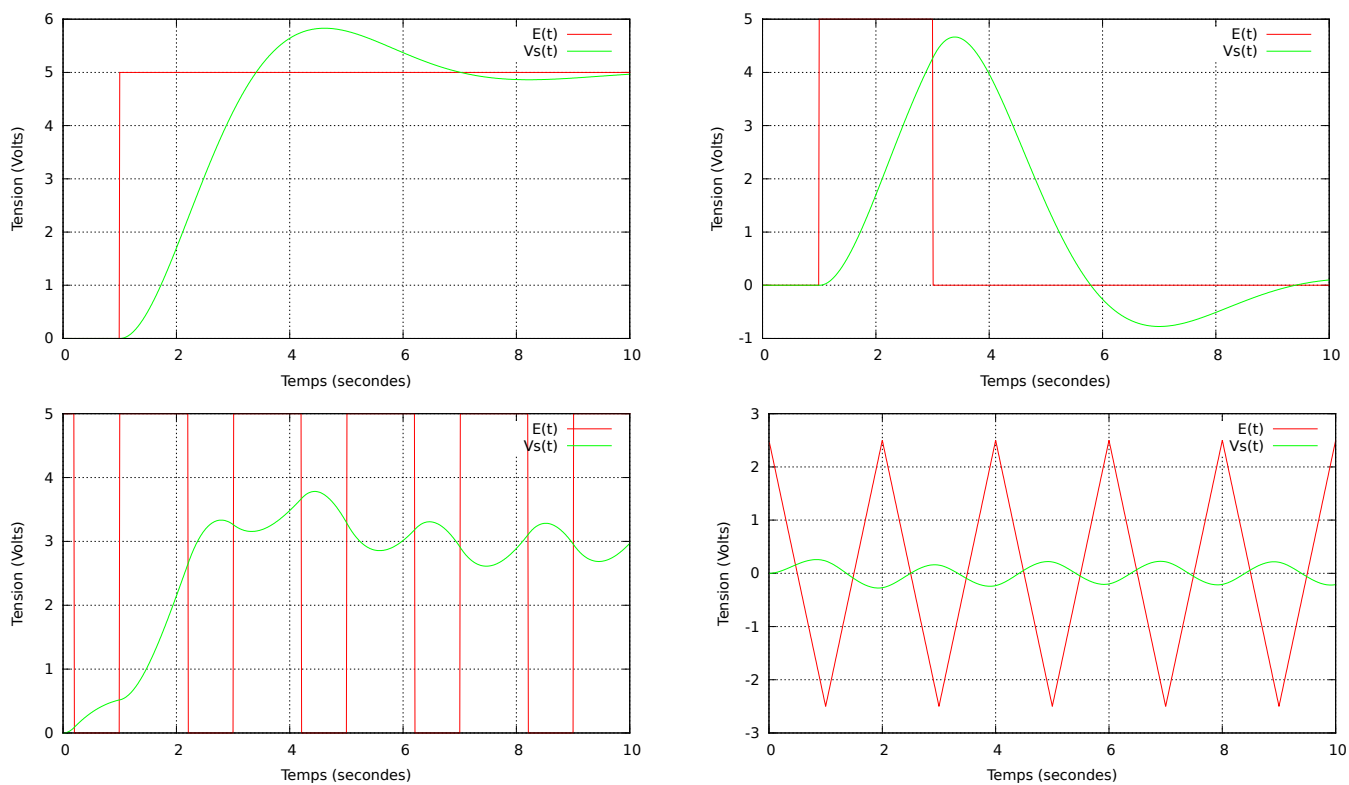


FIGURE 10 – Réponse du circuit C

3.6 Réponse du CircuitD

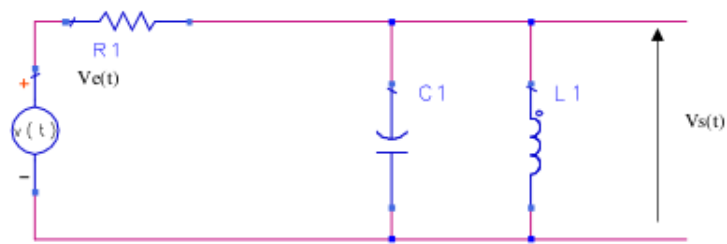


FIGURE 11 – Circuit D

Le circuit D est un circuit RLC du 2^e ordre, régit par l'équation différentielle :

Todo

Nous allons étudier la réponse du circuit D à plusieurs types d'excitations, avec comme paramètres de simulation :

- $R=1$ et $C=1$
- pas= , durée=

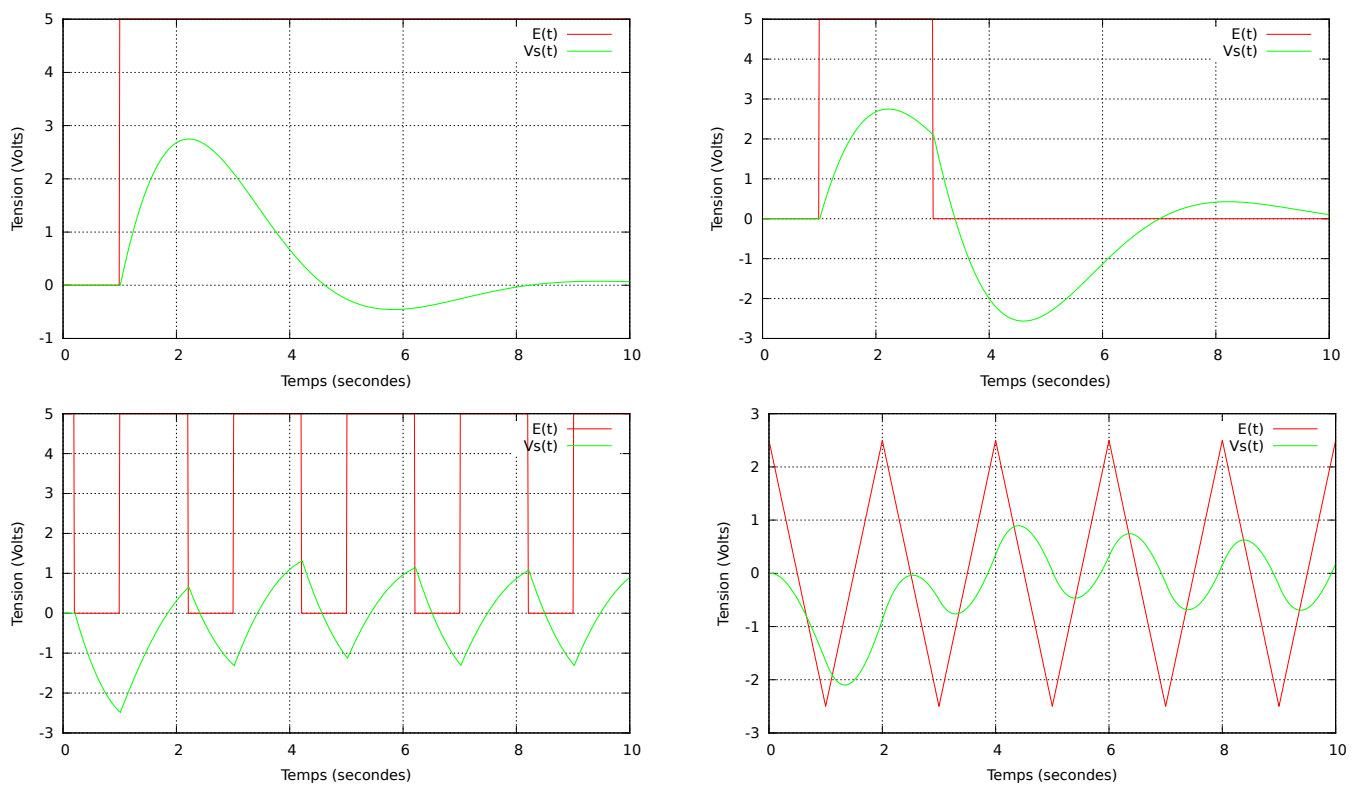


FIGURE 12 – Réponse du circuit D

A Listing du programme

A.1 main.cpp

```
1  /* Programmation orientee objet : BE2 */
2  /* Jeremie Fourmann et Maxime Morin   */
3  /* main.cpp                           */
4  /* Programme principal                 */
5
6
7  #include <iostream>
8  #include "circuits.h"
9  #include "sources.h"
10
11 using namespace std;
12
13 int main(int argc, char **argv)
14 {
15     cout.width(6);
16     cout.precision(4);
17
18     circuit * montage;
19     int choix=0;
20
21     cout << "#Premier Ordre :" << endl;
22     cout << "#1 - Exemple 1" << endl;
23     cout << "#2 - Circuit A" << endl;
24     cout << "#3 - Circuit B" << endl;
25     cout << "#Deuxime Ordre :" << endl;
26     cout << "#4 - Exemple 2" << endl;
27     cout << "#5 - Circuit C" << endl;
28     cout << "#6 - Circuit D" << endl;
29     cin >> choix;
30
31     switch(choix){
32     case 1:
33         montage = new exemple1;
34         break;
35     case 2:
36         montage = new circuitA;
37         break;
38     case 3:
39         montage = new circuitB;
40         break;
41     case 4:
42         montage = new exemple2;
43         break;
44     case 5:
45         montage = new circuitC;
46         break;
47     case 6:
48         montage = new circuitD;
49         break;
50     default:
51         cout << "#Mauvaix choix" << endl;
52         return 0;
53     }
54
55
56     montage->circuitSolve();
57
58     return 0;
59 }
```

A.2 circuits.h

```
1  /* Programmation orientee objet : BE2 */
   /* Jeremie Fourmann et Maxime Morin */
3  /* circuits.h */
   /* Declaration des classes circuits */
5
   #ifndef DEF_circuits
7   #define DEF_circuits
   #include "sources.h"
9
   /* Classe "euler" pour la resolution de au'+bu=f. */
11  class euler{
       protected:
13         double pas,duree,t ;
         source *generateur;
15     public:
         euler();
17         virtual void diffSolve()=0;
         virtual void circuitSolve()=0;
19 };

21
   /* Classe "circuit" (permet le choix de la source) */
23  class circuit : public euler{
       protected:
25         double a,b,ci,u,up;
     public:
27         circuit();
         virtual void diffSolve()=0;
29         virtual void circuitSolve()=0;
   };

31
   /* Classe "circuit1" (1er ordre) */
33  class circuit1 : public circuit{
     public:
35         void diffSolve();
         virtual void circuitSolve() =0; //defini en fct du circuit
37 };

39
   /* Classe "exemple1". */
41  class exemple1 : public circuit1{
     public:
43         exemple1();
         void circuitSolve();
45 };

47
   /* Classe "circuitA". */
49  class circuitA : public circuit1{
       protected:
         double R,C;
51     public:
         circuitA();
53         void circuitSolve();
   };

55
   /* Classe "circuitB". */
57  class circuitB : public circuit1{
       protected:
59         double Rd,C,R;
     public:
61         circuitB();
         void circuitSolve();
63 };
```

```

65  /* Classe "circuit2" (2eme Ordre)*/
    class circuit2 : public circuit{
67      protected:
          double ci2,u2,u2p;
69      public:
          circuit2();
71          virtual void diffSolve()=0;
          virtual void circuitSolve();
73  };
    /* Classe "exemple2" (2eme Ordre)*/
75  class exemple2 : public circuit2{
        public:
77          exemple2();
          void diffSolve();
79          void circuitSolve(); //Redefinition pour les besoins de l'exemple
    };
81
    /* Classe "circuitC". */
83  class circuitC : public exemple2{
        protected:
85          double R,C,L;
        public:
87          circuitC();
    };
89
    /* Classe "circuitD". */
91  class circuitD : public circuit2{
        protected:
93          double R,C,L;
        public:
95          circuitD();
          void diffSolve();
97  };
99  #endif

```

A.3 circuits.cpp

```
1  /* Programmation orientee objet : BE2 */
   /* Jeremie Fourmann et Maxime Morin */
3  /* circuits.cpp */
   /* Definition des classes circuits */
5
6  #include <iostream>
7  #include <math.h>
   #include "circuits.h"
9
10 using namespace std;
11
12 euler::euler(){
13     pas=0.01;
14     duree=10;
15     t=0.0;
16 }
17
18 /* Choix de la source lors de la creation d'un circuit. */
19 circuit::circuit(){
20     int choix=0;
21     a=0.0;
22     b=0.0;
23     ci=0.0;
24     u=0.0;
25     up=0.0;
26
27     cout << "#Choisir la source ?" << endl;
28     cout << "#1 - Echelon" << endl;
29     cout << "#2 - Porte" << endl;
30     cout << "#3 - Carre" << endl;
31     cout << "#4 - Triangle" << endl;
32     cout << "#5 - Rampe f(t)=-3*t (Exemple1)" << endl;
33     cout << "#6 - Nulle (Exemple 2)" << endl;
34     cin >> choix;
35
36     switch(choix){
37     case 1:
38         generateur=new echelon;
39         break;
40     case 2:
41         generateur=new porte;
42         break;
43     case 3:
44         generateur=new carre;
45         break;
46     case 4:
47         generateur=new triangle;
48         break;
49     case 5:
50         generateur=new fctExo1;
51         break;
52     case 6:
53         generateur=new echelon; /* Generateur quelconque. */
54         generateur->setAB(0,0); /* Coupe le generateur. */
55         break;
56     default:
57         break;
58     }
59 }
60
61 void circuit1::diffSolve(){
62     up=u;
63     u=(pas/a)*(generateur->Esm(t)+up*(-b+a/pas));
```

```

65         t=t+pas;
66     }
67
68     exemple1::exemple1(){ //Cas "mathematique" de l'exercice 1
69         a=1;
70         b=3;
71         ci = 0;
72     }
73
74     void exemple1::circuitSolve(){
75         cout << "#Temps" << " " << "SolEuler" << " " << "SolExacte" << " " << endl;
76         while(t<= duree){
77             diffSolve();
78             cout << t << " " << u << " " << -(1/3)*exp(-3*t) -t + (1/3) << endl;
79         }
80     }
81
82     /* Circuit A avec comme parametres R et C */
83     circuitA::circuitA(){
84         cout << "#Choix des valeurs pour le circuit suivant :" << endl ;
85         cout << "#____/\\/\\/\\____ " << endl ;
86         cout << "#|      R      |_" << endl ;
87         cout << "#E          C ---" << endl ;
88         cout << "#|_____|" << endl ;
89
90         cout << "#Valeur de R (Ohm) : " << endl;
91         cin >> R ;
92         cout << "#Valeur de C (Farad) : " << endl ;
93         cin >> C ;
94
95         a=R*C;
96         b=1;
97         generateur->setAB(1,0); // Esm(t) = E(t)
98     }
99
100     /* Resolution de l'equation differentielle du circuitA pour la source choisie. */
101     void circuitA::circuitSolve(){
102
103         cout << "#Temps" << " " << "Ve" << " " << "Vs" << " " << endl;
104         while(t<= duree){
105             diffSolve();
106             cout << t << " " << generateur->E(t) << " " << u << endl;
107         }
108     }
109
110     /* Circuit B avec comme parametres Rd, R et C. */
111     circuitB::circuitB(){
112         cout << "#Choix des valeurs pour le circuit suivant :" << endl ;
113         cout << "#____/\\/\\/\\____|\\_____ " << endl ;
114         cout << "#|      Rd      |/" << endl ;
115         cout << "#|          D      /      |_" << endl ;
116         cout << "#E          R \\      --- C " << endl ;
117         cout << "#|          /      |" << endl ;
118         cout << "#|_____|_____|" << endl ;
119
120         cout << "#Valeur de Rd (Ohm) : " << endl;
121         cin >> Rd ;
122         cout << "#Valeur de R (Ohm) : " << endl;
123         cin >> R ;
124         cout << "#Valeur de C (Farad) : " << endl ;
125         cin >> C ;
126     }
127
128     /*Resolution des equations differentielles circuitB pour la source

```

```

131 choisie, pour les deux differents etats de la diode */
void circuitB::circuitSolve(){
133     bool bloquee=1; //Flag d'etat de la diode
    double vd=.7; // A t=0, C dechargee donc D passante (vd>0.6)
135     ci=0; // C dechargee

137     cout << "#Temps" << " " << "Ve" << " " << "Vs" << " " << "Vd" << endl;
    while(t<=duree){
139         if(vd>=.6 && bloquee ){
            a=Rd*C;
141             b=1+Rd/R;
            generateur->setAB(1,-0.6); // Offset pour le second membre
143             ci=u;
            cout << "#Diode passante"<<endl;
145             bloquee=0;
        }
147         if(vd<.6 && !bloquee )
        {
149             a=R*C;
            b=1;
151             generateur->setAB(0,0); // Second membre nul, decharge de C dans R
            ci=u;
153             cout << "#Diode bloquee"<<endl;
            bloquee=1;
155         }
        diffSolve();
157         vd=generateur->E(t)-u-Rd*C*(u-up)/pas+u/R;
        cout << t << " " << generateur->E(t) << " " << u << " " << vd << endl;
159     }
}

161 /*Circuit 2 ordre*/
163 circuit2::circuit2(){
165     u2=0.0;
    u2p=0.0;
167     ci2=0;

169 }

171 void circuit2::circuitSolve(){
    cout << "#Temps" << " " << "ESM" << " " << "Vs" << " " << endl;
173     u=ci;
    u2=ci2;
175     while(t<=duree){
        diffSolve();
177         cout << t << " " << generateur->Esm(t) << " " << u << endl;
    }
179 }

181 /*Resolution de l'exemple numero 2 */
exemple2::exemple2(){ //Cas mathematique de l'exercice 2
183     a=0.0;
    b=-1.0;
185     ci2=1;
}

187 void exemple2::diffSolve(){
189     up=u;
    u2p=u2;
191     u=up+pas*u2p;
    u2=u2p+pas*(b*up+a*u2p+generateur->Esm(t));
193     t=t+pas;
}

195 void exemple2::circuitSolve(){

```

```

197     cout << "#Temps" << " " << "Entree" << " " << "Sortie-Solution" << " " << "Sinus(sol exact)" << "
    " << endl;
    u=ci;
199     u2=ci2;
    while(t<=duree){
201         diffSolve();
        cout << t << " " << generateur->Esm(t) << " " << u << " " << sin(t) << endl;
203     }
}
205
/*Constructeur du circuitC*/
207 circuitC::circuitC(){ //Cas special de l'exercice 2
    cout << "#Valeur de R (Ohm) : " << endl;
209     cin >> R ;
    cout << "#Valeur de L (Henry) : " << endl;
211     cin >> L ;
    cout << "#Valeur de C (Farad) : " << endl ;
213     cin >> C ;

215     a=-R/L;
    b=-1/(L*C);
217     ci=0.0;
    ci2=0.0;
219
    generateur->setAB(1,0);
221 }

223 circuitD::circuitD(){
    cout << "#Valeur de R (Ohm) : " << endl;
225     cin >> R ;
    cout << "#Valeur de L (Henry) : " << endl;
227     cin >> L ;
    cout << "#Valeur de C (Farad) : " << endl ;
229     cin >> C ;

231     a=-1/(R*C);
    b=-1/(L*C);
233     ci=0.0;
    ci2=0.0;
235
    generateur->setAB(-a,0);
237 }

239 void circuitD::diffSolve(){
    up=u;
241     u2p=u2;
    u=up+pas*u2p;
243     u2=u2p+pas*(b*up+a*u2p)+(generateur->Esm(t)-generateur->Esm(t-pas)); //on code la deriv de la fct
    second membre
    t=t+pas;
245 }

```


A.4 sources.h

```
1  /* Programmation orientee objet : BE2 */
   /* Jeremie Fourmann et Maxime Morin */
3  /* sources.h */
   /* Declaration des classes sources */
5
   #ifndef DEF_sources
7  #define DEF_sources

9  /* Classe mere : source. */
   class source{
11
       protected:
13         double T,phi,offset,ampli,alpha,sauvAmpli;
           double A,B;
15     public:
           source();
17         virtual double E(double t)=0;//fct virtuelle de la source
           double Esm(double t); // Transformation affine de E pour changer amplitude
19                                     // ou ajouter un offset dans le second membre
           void setAB(double Ai, double Bi); //accesseur pour les valeurs A et B
21 };

23 /* Classe fille permettant de traiter l'exemple 1. */
   class fctExo1 : public source{
25     public:
           double E(double t);
27 };

29 /* Classes filles pour les differents signaux d'entree. */
   class echelon : public source{
31     public:
           double E(double t);
33 };

35 class porte : public source{
       public:
37         double E(double t);
       };

39 class triangle : public source{
       public:
41         double E(double t);
       };
43 class carre : public source{
       public:
45         double E(double t);
       };
47
49 #endif
```

A.5 sources.cpp

```
1  /* Programmation orientee objet : BE2 */
2  /* Jeremie Fourmann et Maxime Morin */
3  /* sources.cpp */
4  /* Definition des classes sources */
5
6  #include <iostream>
7  #include "sources.h"
8  #include <math.h>
9
10 using namespace std;
11
12
13 /* Methodes de la classe mere "source". */
14 source::source(){
15     T=2;
16     phi=1;
17     offset=0;
18     ampli=5;
19     alpha=.6;
20     A=1, B=0;
21 }
22
23 double source::Esm(double t) // Transformation affine du signal de la source
24 {
25     return A*E(t)+B;
26 }
27
28
29 /* Definitions des sources filles pour differents types de signaux ou fonctions. */
30
31 double fctExo1::E(double t){
32
33     return -3*t;
34 }
35
36 void source::setAB(double Ai, double Bi)
37 {
38     A = Ai;
39     B = Bi;
40 }
41
42
43 double echelon::E(double t){
44     double fx;
45     if(phi <=t ) fx= offset+ampli;
46     else fx= offset;
47     return fx;
48 }
49
50
51 double porte::E(double t){
52     double fx;
53     if(phi < t && t <phi+T) fx=offset+ampli;
54     else fx=offset;
55     return fx;
56 }
57
58 double carre::E(double t){
59     double fx;
60     if((t-phi)-floor((t-phi)/T)*T<T*alpha) fx=offset+ampli;
61     else fx=offset;
62     return fx;
63 }
```

```
65 double triangle::E(double t){  
    double fx;  
67     if((t-phi)-floor((t-phi)/T)*T<=T/2) fx=((t-phi)-floor((t-phi)/T)*T-.5)*ampli+offset;  
        else fx=(-((t-phi)-floor((t-phi)/T)*T)+2-.5)*ampli+offset;  
69     return fx;  
}
```

B Script de collecte des données et Gnuplot

En fainéant et pauvres étudiant que nous sommes, nous cherchions une solution rapide, gratuite et disponible chez soi pour générer les tracés demandés. Notre programme prend en entrée une combinaison correspondant au choix de l'utilisateur et renvoi les données dans la sortie standart. Par exemple pour demander au programme de nous afficher les résultats de l'exemple 1 on utilise le *pipe* qui dirige la sortie d'un programme vers l'entrée d'un autre :

```
echo 1 5 | ./main
```

En se plaçant évidemment au préalable dans le répertoire contenant notre fichier objet appelé *main*. Pour envoyer ces résultats dans un fichier on utilise le chevron :

```
1 echo 1 5 | ./main > donnees.dat
```

Automatiser la procedure revient à exécuter cette inscruptions pour toutes les données demandées, c'est quasiment du copier-coller :

B.1 collecte.sh

```
1
3 echo 2 1 1 1 | ../main > ../plot/CAechelon.dat;
  echo 2 2 1 1 | ../main > ../plot/CAporte.dat;
5 echo 2 3 1 1 | ../main > ../plot/CAcarre.dat;
  echo 2 4 1 1 | ../main > ../plot/CAtriangle.dat;
7
  echo 3 1 1 1 1 | ../main > ../plot/CBechelon.dat;
9 echo 3 2 1 1 1 | ../main > ../plot/CBporte.dat;
  echo 3 3 1 1 1 | ../main > ../plot/CBcarre.dat;
11 echo 3 4 1 1 1 | ../main > ../plot/CBtriangle.dat;
13
  echo 4 6 | ../main > ../plot/exemple2.dat;
15
  echo 5 1 1 1 1 | ../main > ../plot/CEechelon.dat;
  echo 5 2 1 1 1 | ../main > ../plot/CEporte.dat;
17 echo 5 3 1 1 1 | ../main > ../plot/CEcarre.dat;
  echo 5 4 1 1 1 | ../main > ../plot/CEtriangle.dat;
19
  echo 6 1 1 1 1 | ../main > ../plot/CDechelon.dat;
21 echo 6 2 1 1 1 | ../main > ../plot/CDporte.dat;
  echo 6 3 1 1 1 | ../main > ../plot/CDcarre.dat;
23 echo 6 4 1 1 1 | ../main > ../plot/CDtriangle.dat;
25 gnuplot aTrace;
  rm images/concat.pdf
27 pdftk images/*.pdf cat output images/concat.pdf;
```

Les dernières lignes appellent GNUplot. Le fichier *aTrace* dit au logiciel où se trouvent les données, où il doit les tracées, leur format de sortie, les légendes, etc... Lui aussi est un peu répétitif.

B.2 aTrace

```
2 set terminal pdf
  #set key off
4 a=2 #regle epaisseur des traces
  set key top right #positionne la legende
6 set grid

8 set xlabel "x"
  set ylabel "u(x)"
10
  set output "images/exemple1.pdf"
12 plot "../plot/exemple1.dat" using 1:2 title "Solution mthode d'Euler" with lines linewidth a , "../plot/
  exemple1.dat" using 1:3 title "Solution Anaytique" with lines linewidth a
```

```

unset output
14
set output "images/exemple2.pdf"
16 plot "../plot/exemple2.dat" using 1:3 title "Rsolution numrique" with lines linewidth a,"../plot/exemple2.dat"
using 1:4 title "Solution analytique" with lines linewidth a
unset output
18
set xrange[0:10]
20
set xlabel "Temps (secondes)"
22 set ylabel "Tension (Volts)"

24 set output "images/CAechelon.pdf"
plot "../plot/CAechelon.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CAechelon.dat" using 1:3
title "Vs(t)" with lines linewidth a
26 unset output

28 set output "images/CAporte.pdf"
plot "../plot/CAporte.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CAporte.dat" using 1:3 title
"Vs(t)" with lines linewidth a
30 unset output

32 set output "images/CACarre.pdf"
plot "../plot/CACarre.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CACarre.dat" using 1:3 title
"Vs(t)" with lines linewidth a
34 unset output

36 set output "images/CATriangle.pdf"
plot "../plot/CATriangle.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CATriangle.dat" using 1:3
title "Vs(t)" with lines linewidth a
38 unset output

40
set output "images/CBechelon.pdf"
42 plot "../plot/CBechelon.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CBechelon.dat" using 1:3
title "Vs(t)" with lines linewidth a
unset output
44
set output "images/CBporte.pdf"
46 plot "../plot/CBporte.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CBporte.dat" using 1:3 title
"Vs(t)" with lines linewidth a
unset output
48
set output "images/CBcarre.pdf"
50 plot "../plot/CBcarre.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CBcarre.dat" using 1:3 title
"Vs(t)" with lines linewidth a
unset output
52
set output "images/CBtriangle.pdf"
54 plot "../plot/CBtriangle.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CBtriangle.dat" using 1:3
title "Vs(t)" with lines linewidth a
unset output
56
set output "images/CCechelon.pdf"
58 plot "../plot/CCechelon.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CCechelon.dat" using 1:3
title "Vs(t)" with lines linewidth a
unset output
60
set output "images/CCporte.pdf"
62 plot "../plot/CCporte.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CCporte.dat" using 1:3 title
"Vs(t)" with lines linewidth a
unset output
64
set output "images/CCcarre.pdf"
66 plot "../plot/CCcarre.dat" using 1:2 title "E(t)" with lines linewidth a,"../plot/CCcarre.dat" using 1:3 title
"Vs(t)" with lines linewidth a

```

```

unset output

68
set output "images/CCtriangle.pdf"
70 plot "../plot/CCtriangle.dat" using 1:2 title "E(t)" with lines linewidth a, "../plot/CCtriangle.dat" using 1:3
   title "Vs(t)" with lines linewidth a
unset output

72

74 set output "images/CDechelon.pdf"
   plot "../plot/CDechelon.dat" using 1:2 title "E(t)" with lines linewidth a, "../plot/CDechelon.dat" using 1:3
   title "Vs(t)" with lines linewidth a
76 unset output

78 set output "images/CDporte.pdf"
   plot "../plot/CDporte.dat" using 1:2 title "E(t)" with lines linewidth a, "../plot/CDporte.dat" using 1:3 title
   "Vs(t)" with lines linewidth a
80 unset output

82 set output "images/CDcarre.pdf"
   plot "../plot/CDcarre.dat" using 1:2 title "E(t)" with lines linewidth a, "../plot/CDcarre.dat" using 1:3 title
   "Vs(t)" with lines linewidth a
84 unset output

86 set output "images/CDtriangle.pdf"
   plot "../plot/CDtriangle.dat" using 1:2 title "E(t)" with lines linewidth a, "../plot/CDtriangle.dat" using 1:3
   title "Vs(t)" with lines linewidth a
88 unset output

```