UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

introduction à C++ et
résolution de problèmes de physique par ordinateur

Corrigé 10

Professeur : Alessandro Bravar
Alessandro.Bravar@unige.ch

Université de Genève
Section de Physique

Semestre de printemps 2015

Références :

M-Y. Bachmann, H. Catin, P. Epiney *et al.*
Méthodes numériques

A. Quateroni, F. Saleri et P. Gervasio
Calcul scientifique

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery
Numerical Recipes

`http://dpnc.unige.ch/~bravar/C++2015/10` :
pour les notes du cours, les exercices et les corrigés

10.1 Problèmes

1. **Pendule Euler**
   Voir le programme **Pendule_Euler.cpp** developpé pendant le cours.
   Soit L la longueur du pendule, ω sa vitesse angulaire et ϑ l'angle d'oscillation. L'équation exacte du mouvement s'écrit :

$$\frac{d^2\vartheta(t)}{dt^2} = -\frac{g}{L}\sin\vartheta(t) \tag{1}$$

   Si l'on considère les oscillations suffisement petite ($\vartheta < 5°$), l'on peut faire l'approximation :

$$\frac{d^2\vartheta(t)}{dt^2} \approx -\frac{g}{L}\vartheta(t) \tag{2}$$

   qui a comme solution :

$$\vartheta(t) = Acos(\omega_0 t + \phi_0) \tag{3}$$

   avec :

$$\omega_0 = \frac{2\pi}{T_0} \quad \text{et} \quad T_0 = 2\pi\sqrt{\frac{L}{g}} \tag{4}$$

   A et ϕ_0 sont déterminés à partir des conditions initiales.
   Dans ce programme on calcule les oscillations dans l'approximation des petites oscillations en utilisant la méthode d'Euler (avec dans les deux cas : $\vartheta_0 = 5°$ et $\omega_0 = 0$). Les valeurs de l'angle et de la vitesse angulaire (en considérant l'approximation des petites oscillations) sont données par :

$$\begin{aligned} \vartheta_{n+1} &= \vartheta_n + \omega_n\ \Delta t \\ \omega_{n+1} &= \omega_n + a_n \quad \text{avec ici} \quad a_n = -\frac{g}{L}\vartheta_n = \vartheta_n \end{aligned}$$

   La méthode de Runge étant d'ordre supérieur, elle donnera une meilleure approximation de la solution exacte que ne le fait la méthode d'Euler.

2. **Pendule Runge**
   Voir le programme **Pendule_Runge.cpp**.
   Ce programme fait également le calcul de la trajectoire d'un pendule avec l'approximation des petites oscillations mais en utilisant la méthode de Runge. La méthode de Runge consiste à calculer la vitesse au milieu de l'intervalle Δt et la position au début de l'intervalle. Pour démarrer l'itération il faut redéfinir la vitesse initiale ω_0 dans $t - \Delta t/2$ selon :

$$\omega_0 = \omega_0 - a_0\ \Delta t/2 \quad a_0 \text{ étant l'accélération} \quad a_0 = -\frac{g}{L}\ \vartheta_0$$

   qui nous permet d'obtenir

$$\omega_1 = \omega(\Delta t/2) = \omega_0 + a_0\ \Delta t$$

   à partir de la nouvelle vitesse initiale ω_0. Et on calcule ensuite l'accélération, la vitesse et la position à chaque itération selon l'algorithme :

$$\begin{aligned} a_n &= -\frac{g}{L}\ \vartheta_n \quad \text{avec ici} \quad \frac{g}{L} = 1 \\ \omega_{n+1} &= \omega_n + a_n\ \Delta t \\ \vartheta_{n+1} &= \vartheta_n + \omega_{n+1}\ \Delta t \end{aligned}$$

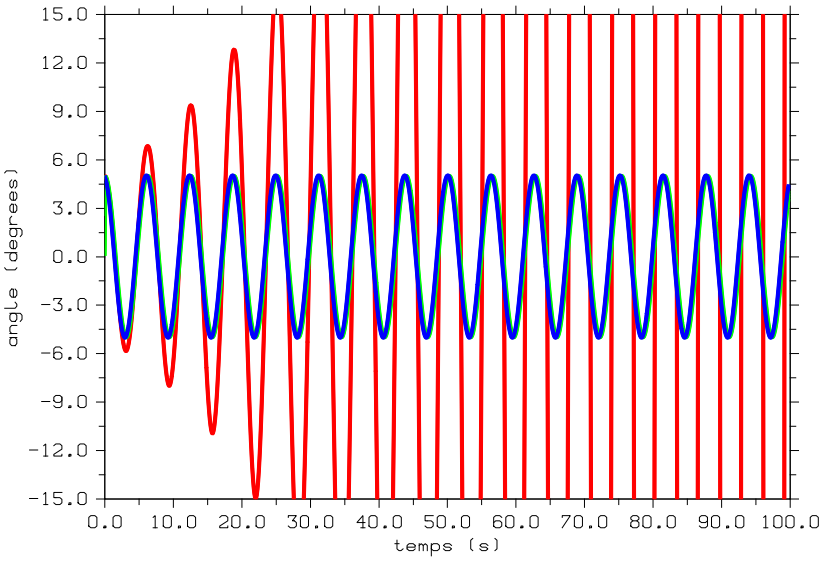

FIGURE 1 – Comparaison entre les méthodes d'Euler et de Runge pour un pendule.

La figure 1 montre en rouge la solution obtenue avec la méthode d'Euler, en bleu avec la méthode de Runge et en vert la solution exacte, qui se superpose parfaitement à la solution de Runge. La méthode de Runge est d'ordre supérieure à celle d'Euler et donne donc un bien meilleure approximation de la solution exacte.

3. **Oscillateur anharmonique**
L'oscillateur anharmonique est défini comme un oscillateur avec un force non linéaire, son équation de mouvement est la suivante :

$$\ddot{x} \;=\; -k\, f(x) = -k\, \frac{x}{|x|}|x|^{\alpha}$$

L'utilisateur doit en premier lieu entrer la valeurs des constantes k et α.
On considére l'élongation initiale $x_0 = 10$ et la vitesse initiale $v_0 = 0$.
Dans la méthode de Runge pour démarrer l'itération il faut d'abord, calculer $v_1 = v(\Delta t/2)$

$$v_1 = v_0 + a_0\ \Delta t/2 \quad \text{avec} \quad a_0 = -k\, \frac{x_0}{|x_0|}|x_0|^{\alpha}$$

ou redéfinir la vitesse initiale comme ci-dessous :

$$v_0 = v_0 - a_0\ \Delta t/2$$

Ensuite on calcule l'accélération, la vitesse et la position à chaque itération :

$$\begin{aligned} a_n &= -k\, \frac{x_n}{|x_|}|x_n|^{\alpha} \\ v_{n+1} &= x_n + a_n\ \Delta t \\ x_{n+1} &= x_n + v_{n+1}\ \Delta t \end{aligned}$$

La figure 2 montre les solutions pour valeurs de $\alpha = 0$ en bleu, $= 1$ en rouge, et $= 2$ en vert. Le coefficient $k/m = 1$ dans cet exemple. On constate qu'il y a une certaine periodicité et que l'amplitude maximale ne change que peu dans le temps d'intégration consideré.
Osc_anharmonique.cpp

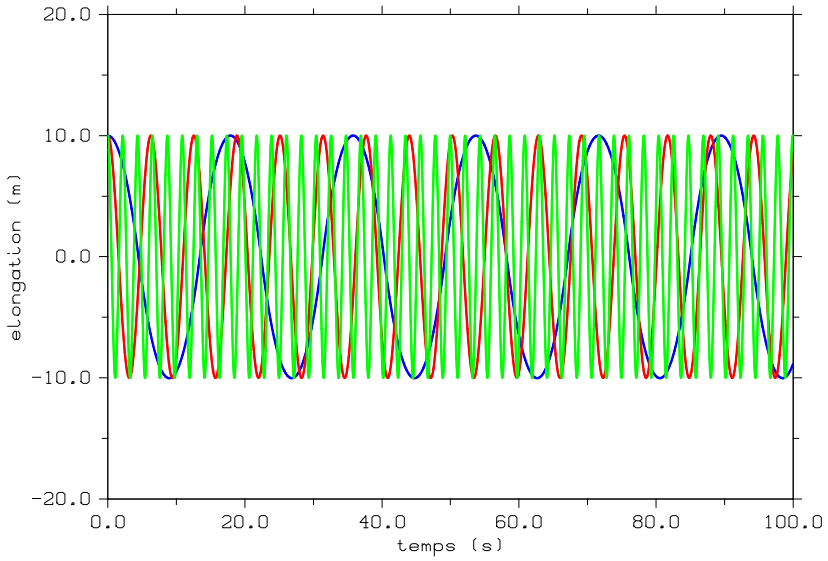

FIGURE 2 – Solutions avec différents valeurs du coefficient anharmonique.

```
1  //oscillateur anharmonique
2  #include <iostream>
3  #include <cmath>
4  #include "dislin.h"
5
6  using namespace std;
7
8  int main() {
9    cout << "Etude de l'oscillateur anharmonique" << endl;
10   cout << "Entrez la constante k/m de l'oscillateur : ";
11   double k = 1.;
12   cin >> k;
13   cout << "Entrez l'exposant alpha (alpha = 1 -> osc. harmonique) : ";
14   double alpha = 1.;
15   cin >> alpha;
16   cout << "Entrez l'elongation initiale : ";
17   double angle0 = 10.;
18   cin >> angle0;
19
20   //initialisation de DISLIN
21   metafl("XWIN");    //XWIN ou PDF
22   disini();
23   name("temps (s)", "X");
24   name("elongation (m)", "Y");
25   titlin("Oscillateur anharmonique", 1);
26   graf(0.,100.,0.,20.,-2*angle0,2*angle0,-2*angle0,angle0);
27   title();
28
29   const int STEPS = 10000;
30   double dt = 0.01;
31   double angle[STEPS], omega[STEPS], time[STEPS];
32   angle[0] = angle0;    //elongation initial
33   omega[0] = 0.;     //vitesse initiale
34   time[0] = 0.;
35
36   //methode de Runge
37   //conditions initiales
38   double acc;
39   acc = - angle[0]/abs(angle[0])*k * pow(abs(angle[0]),alpha);
```

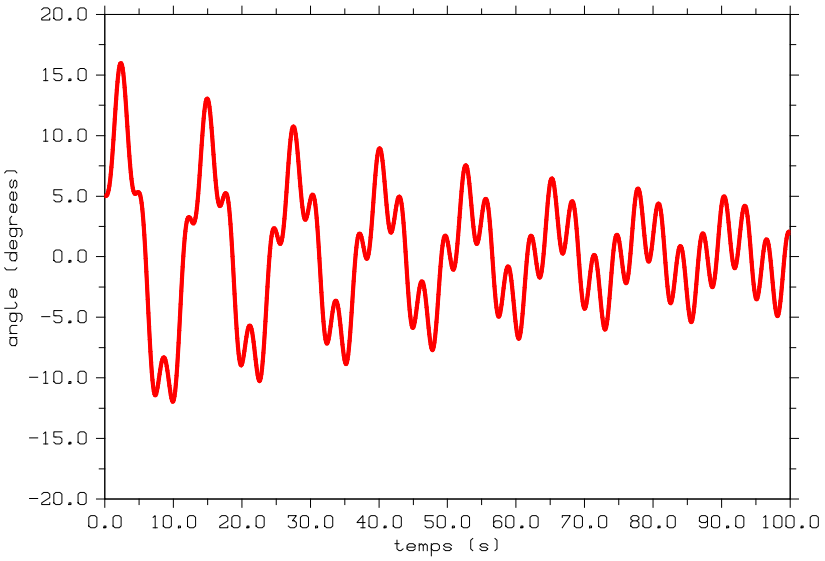

FIGURE 3 – Solutions du pendule physique avec longueur $L = 40$ m, $\gamma = 0.04$ s^{-1}, $F = 0.2$ s^{-2}, $\Omega = 1.0$ rad/s et angle initial 5°.

```
40    omega[0] = omega[0] - acc*dt/2.;      //methode de Runge !
41
42    for (int i=1; i<STEPS; i++) {
43      acc = - angle[i-1]/abs(angle[i-1])*k * pow(abs(angle[i-1]),alpha);
44      omega[i] = omega[i-1] + acc*dt;
45      angle[i] = angle[i-1] + omega[i]*dt;
46      time[i] = i*dt;
47    }
48
49    //dessin de la trajectoire
50    color("RED");
51    curve(time, angle, STEPS);
52    disfin();       //fin de DISLIN
53
54    return 0;
55  }
```

4. **Pendule physique**

Le pendule physique subit en plus de la gravité une force d'amortissement et une force d'entrainement. Dans le programme la solution présentée utilise la méthode de Runge. On calcule d'abord $v_1 = v(\Delta t/2)$. Dans l'algorithme de Runge on calcule l'accélération en fonction de l'ancienne vitesse et position, ensuite la nouvelle vitesse en fonction de cette accélération, et enfin la position en fonction de la nouvelle vitesse :

$$\begin{aligned} a &= -\frac{g}{L}\sin(\vartheta_{i-1}) - \gamma\ \omega_{i-1} + F\sin(\Omega\ t_{i-1}) \\ \omega_i &= \omega_{i-1} + a\ \Delta t \\ \vartheta_i &= \vartheta_{i-1} + \omega_{i-1}\ \Delta t \end{aligned}$$

Les equations ci-dessus sont formulées pour l'angle en radians (p. ex. l'accélération est donnée en rad/s^2). Les fonctions trigonométriques de la bibliothèque `C++` prennent comme arguments les angles en radians. La trajectoire du pendule physique est montrée dans la figure 3.

Pendule_physique.cpp

```
1  //pendule physique avec la methode de Runge
2  #include <iostream>
3  #include <cmath>
4  #include "dislin.h"
5
6  using namespace std;
7
8  int main() {
9    const int STEPS = 50000;
10   double dt = 0.002;
11   double angle[STEPS], omega[STEPS], time[STEPS];
12   angle[0] = 5.*M_PI/180.;   //angle initial en radians
13   omega[0] = 0.;   //vitesse angulaire initiale
14   time[0] = 0.;
15
16   //constantes du pendule physique
17   double g = 9.81;
18   double L = 40.;
19   double gamma = 0.04;
20   double force = 0.2;
21   double freq = 2.;
22
23   double accG, accAm, accEn, acc;
24   //vitesse initiale pour la methode de Runge
25   accG = -g/L * sin(angle[0]);
26   accAm = -gamma * omega[0];
27   accEn = force * sin(freq*time[0]);
28   acc = accG + accAm + accEn;
29   omega[0] = omega[0] - dt/2. * acc;
30   for (int i=1; i<STEPS; i++) {
31     accG = - g/L * sin(angle[i-1]);
32     accAm = - gamma * omega[i-1];
33     accEn = force * sin(freq*time[i-1]);
34     acc = accG + accAm + accEn;
35     omega[i] = omega[i-1] + acc*dt;
36     angle[i] = angle[i-1] + omega[i]*dt;
37     time[i] = i*dt;
38   }
39   for (int i=0; i<STEPS; i++)      //conversion de l'angle en degrees
40     angle[i] *= 180./M_PI;
41
42   //initialisation de dislin
43   metafl("XWIN");    // XWIN ou PDF
44   disini();
45   name("temps (s)", "X");
46   name("angle (degrees)", "Y");
47   graf(0.,100.,0.,10.,-20.,20.,-20.,5.);
48   //desin de la trajectoire
49   thkcrv(10);
50   color("RED");
51   curve(time, angle, STEPS);
52   //fin de DISLIN
53   disfin();
54
55   return 0;
56 }
```

5. **Pendule élastique**
   La décomposition des forces agissant sur la balle attachée au ressort en coordonnèes

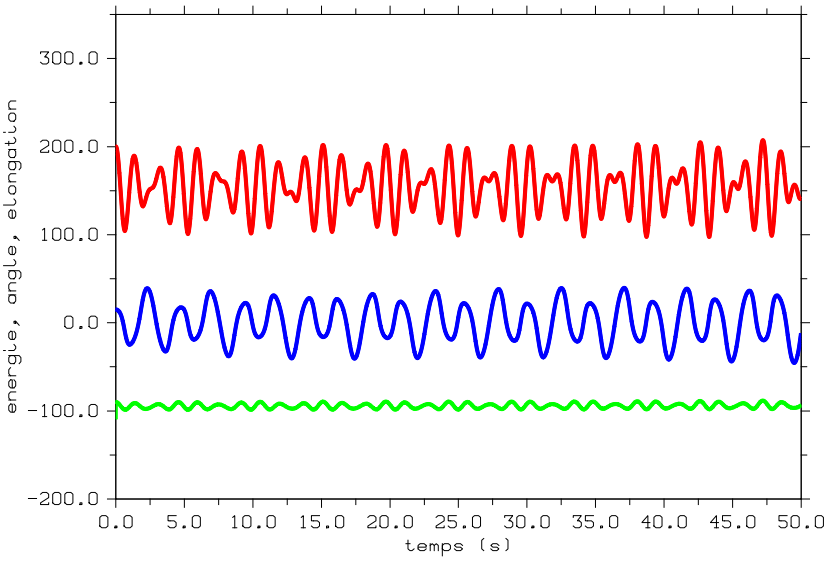

FIGURE 4 – Trajectoire du pendule élastique. La *courbe* verte décrit l'énergie en fonction du temps décalé de -100 unités, la *courbe* blue l'angle et la *courbe* rouge l'élongation décalé de $+100$ unités.

polaires est la suivante :

$$\begin{aligned} \vec{F} &= M\vec{a} = M\ddot{\vec{r}} \\ M\vec{g} - k(r-L)\hat{r} &= M(\ddot{r} - r\dot{\vartheta}^2)\hat{r} + M(r\ddot{\vartheta} + 2\dot{r}\dot{\vartheta})\hat{\vartheta} \\ M(g\cos\vartheta\ \hat{r} - g\sin\vartheta\ \hat{\vartheta}) - k(r-L)\hat{r} &= M(\ddot{r} - r\dot{\vartheta}^2)\hat{r} + M(r\ddot{\vartheta} + 2\dot{r}\dot{\vartheta})\hat{\vartheta} \end{aligned}$$

où r est l'élongation du ressort et ϑ l'angle entre le pendule et la verticale. L'on obtient deux équations du mouvement pour l'élongation du ressort et l'angle, qui sont liées :

$$\begin{aligned} \text{dans la composante } \hat{r} \quad &: \quad \ddot{r} = g\cos\vartheta - \frac{k}{M}(r-L) + r\dot{\vartheta}^2 \\ \text{dans la composante } \hat{\vartheta} \quad &: \quad \ddot{\vartheta} = -\frac{g}{r}\sin\vartheta - 2\frac{\dot{r}\dot{\vartheta}}{r} \end{aligned}$$

On doit résoudre les deux équations au simultanément. A chaque itération on calcule l'accélération angulaire et l'accélération radiale, la vitesse angulaire et la vitesse radiale, et la position angulaire et radiale avec la méthode de Runge.

La figure 4 montre l'élongation (en rouge), l'angle (en blue) et l'énergie (en vert) du pendule élastique pour une masse de 0.1 kg attachée au ressort de constante $k = 2$ N/m avec une élongation initiale de 2 m et d'angle initial de 15°.

Pendule_elastique.cpp

```
1  //pendule elastique
2  #include <iostream>
3  #include <cmath>
4  #include <ctime>
5  #include "dislin.h"        //bilbiotheque DISLIN
6
7  using namespace std;
8
9  int main() {
10   const int STEPS = 10000;
11   double dT = 0.005;
12   double angle[STEPS], longueur[STEPS], temps[STEPS];
13   double vitA[STEPS], vitL[STEPS], accA[STEPS], accL[STEPS];
```

```
14    double energie[STEPS];
15
16    //parametres du pendule
17    double L0 = 1.;    //longueur a repos du ressort en metres
18    double k = 2.;     //constante elastique du ressort en N/m
19    double g = 9.81;   //acceleration gravitatinelle
20    double m = 0.1;    //masse
21
22    cout <<"*****************************************************\n";
23    cout << "Nous avons un pendule avec un ressort au lieu d'une corde.\n"
24         << "La longueur a repos du ressort est " << L0 << " m et\n"
25         << "la constante elastique est " << k << " N/m.\n"
26         << "La masse est " << m << " kg.\n"
27         << "On assume que l'oscillation demarre depuis l'etat de repos\n"
28         << "a un certain angle et elongation initiales.\n\n";
29    cout << "Quelle est l'angle initial du pendule ? ";
30    cin >> angle[0];
31    cout << "Et l'elongation initiale du ressort en metres ? ";
32    cin >> longueur[0];
33
34    int opt;
35    do {
36      cout << "Que type de graphique souhaitez-vous ?" << endl;
37      cout << "1 pour dessiner un graphique" << endl;
38      cout << "2 pour une animation du mouvement du pendule" << endl;
39      cin >> opt;    cout << endl;
40    } while (opt!=1 && opt!=2);
41
42    temps[0] = 0.;
43    angle[0] *= M_PI/180.;
44
45    //ces vecteurs vont etre utilises avec DISLIN
46    double x[STEPS], y[STEPS];
47    x[0] = sin(angle[0])*longueur[0];
48    y[0] = -cos(angle[0])*longueur[0];
49
50    vitA[0]=0.;
51    vitL[0]=0.;
52    accA[0] = -1.*sin(angle[0])*g/longueur[0];
53    accL[0] = cos(angle[0])*g - k*(longueur[0]-L0)/m;
54
55    energie[0] = m*g*y[0] + k*(longueur[0]-L0)*(longueur[0]-L0)/2.;
56    cout << "L'energie initiale du systeme est : " << energie[0] << endl;
57
58    //initialisation de DISLIN
59    metafl("XWIN");    //XWIN ou PDF
60    disini();
61
62    //methode de Runge
63    for (int i=0; i<STEPS-1; i++) {
64      vitA[i+1] = vitA[i] + accA[i]*dT;
65      vitL[i+1] = vitL[i] + accL[i]*dT;
66      angle[i+1]= angle[i] + vitA[i+1]*dT;
67      longueur[i+1]=longueur[i] + vitL[i+1]*dT;
68      temps[i+1]=(i+1)*dT;
69
70      accA[i+1] = -1.*sin(angle[i+1])*g/longueur[i+1]
71                - 2.*(vitA[i+1]*vitL[i+1])/longueur[i+1];
72      accL[i+1] = cos(angle[i+1])*g - k*(longueur[i+1]-L0)/m
73                + longueur[i+1]*vitA[i+1]*vitA[i+1];
```

```
74
75      energie[i+1] = m*vitL[i+1]*vitL[i+1]/2.
76                   + m*longueur[i+1]*longueur[i+1]*vitA[i+1]*vitA[i+1]/2.
77                   + m*g*y[i+1] + k*(longueur[i+1]-L0)*(longueur[i+1]-L0)/2.;
78
79      if (abs(longueur[i+1]<=1.0E-6)) {
80        longueur[i+1]=1.0E-6;
81        cout << "Attention au conditions initiales et aux parametres du
              pendule !" << endl;
82      }
83
84      //tableaux de 1 element pour DISLIN
85      double xplot[1], yplot[1];
86      xplot[0] = sin(angle[i+1])*longueur[i+1];
87      yplot[0] = -cos(angle[i+1])*longueur[i+1];
88      x[i+1] = xplot[0];
89      y[i+1] = yplot[0];
90
91      double lignex[2]={0., x[i+1]};
92      double ligney[2]={0., y[i+1]};
93      //dessin du pendule
94      if (opt==2) {
95        erase();   //efface la fenetre du graphique
96        color("WHITE");
97        name("axe-X", "X");
98        name("axe-Y", "Y");
99        titlin("Animation du pendule elastique", 1);
100       graf(-4.*L0, 4.*L0, -4.*L0, 1.*L0, -5.*L0, 1.*L0, -5.*L0, 1.*L0);
101       title();
102
103       incmrk(-1);
104       color("RED");
105       marker(21);
106       curve(xplot,yplot,1);
107       incmrk(0);
108       thkcrv(1);
109       color("BLUE");
110       curve(x, y, i);
111       thkcrv(5);
112       color("YELLOW");
113       curve(lignex, ligney, 2);
114       sendbf();
115       endgrf();
116     }
117
118   }   //fin de la boucle
119
120   //dessin de la trajectoire
121   if (opt==1) {
122     name("temps (s)", "X");
123     name("energie, angle, elongation", "Y");
124     titlin("pendule elastique", 1);
125     graf(0., STEPS*dT, 0., 1000*dT, -200.*L0, 350.*L0, -200.*L0 , 100.*L0);
126     title();
127
128     for (int i=0; i < STEPS; i++) {
129       angle[i] *= 180/M_PI;
130       longueur[i] *= 100.;
131       energie[i] = energie[i]*10. -100.;
132     }
```

```
133
134       thkcrv(10);
135       color("RED");
136       curve(temps, longueur, STEPS);
137       color("BLUE");
138       curve(temps, angle, STEPS);
139       color("GREEN");
140       curve(temps, energie, STEPS);
141     }
142
143     disfin();       //fin de DISLIN
144
145     return 0;
146   }
```

6. **Orbite elliptique**
   L'intégration de l'orbite est effectuée avec la méthode de Runge. On redéfinit d'abord la vitesse initiale à $t - \Delta t/2$:

$$xvel[0] = xvel0 - xacc \times \frac{\Delta t}{2};$$
$$yvel[0] = yvel0 - yacc \times \frac{\Delta t}{2};$$

   ou Δt est la pas d'intégration pour la trajectoire. Pour calculer l'accélération à $t = 0$ on utilise la distance au centre gravitationnel :

$$dist = \sqrt{xpos[0] \times xpos[0] + ypos[0] \times ypos[0]};$$
$$xacc = -GM \times xpos[0] \ / \ dist^3;$$
$$yacc = -GM \times ypos[0] \ / \ dist^3;$$

   $GM = GM_{\odot}m$. Pour l'algorithme d'intégration, voir le programme **Orbite_Runge.cpp** développé pendant la leçon. On calcule l'énergie potentielle et cinétique à chaque pas avec les équations $E_{pot} = -GM_{\odot}m/r$ et $E_{cin} = \frac{1}{2}mv^2$. La figure 5 montre l'évolution de l'énergie (en unités AU, $M_{\odot}$, années). La somme de l'énergie potentielle et cinétique reste constante.

   Orbite.cpp

```
1  //etude de l'orbite avec la methode de Runge
2  #include <iostream>
3  #include <cstdlib>
4  #include <cmath>
5  #include "dislin.h"
6
7  using namespace std;
8
9  int main() {
10   const int steps = 20000;
11   const double GM = 1.;      //G_N * M_Soleil
12   const double m = 1.;       //masse planete
13   double xpos[steps], ypos[steps], xvel[steps], yvel[steps], time[steps];
14   double ePot[steps], eCin[steps], eTot[steps], l[steps];
15   //conditions initiales
16   xpos[0] = 1.0; ypos[0] = 0.0;
17   xvel[0] = 0.0; yvel[0] = 1.3;
18   time[0] = 0.;
19
```

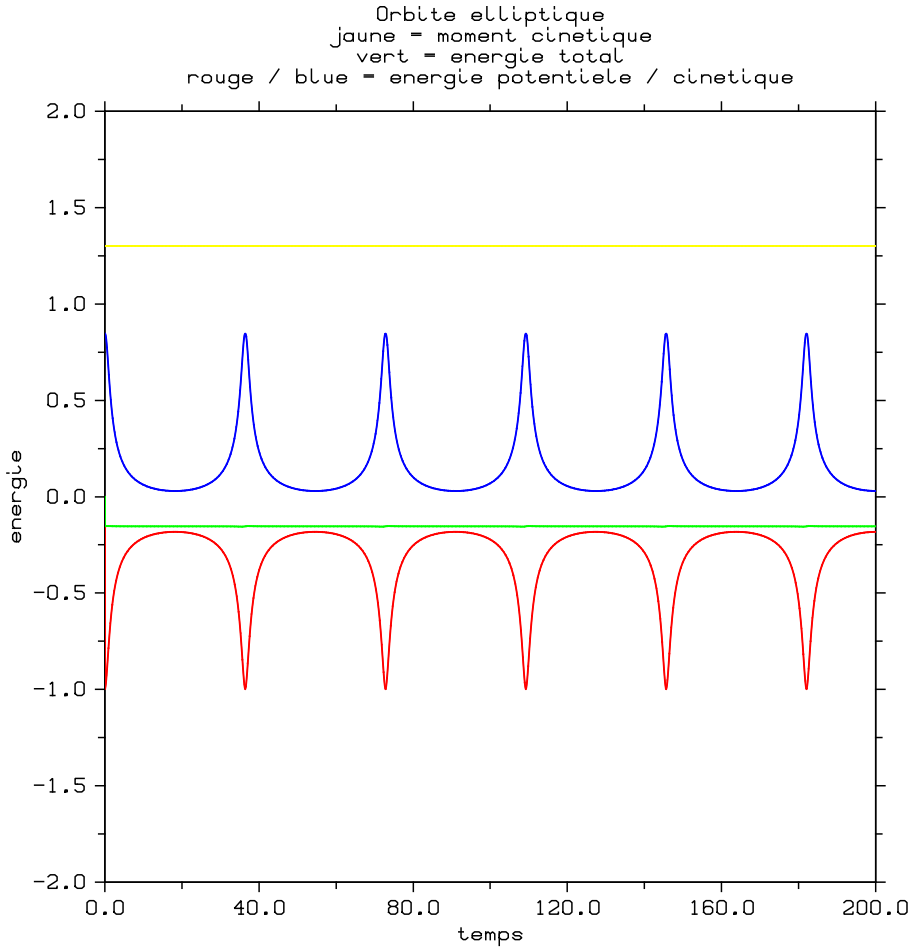

FIGURE 5 – Evolution de l'énergie cinétique (bleu) et potentielle (rouge) de l'orbite. Le moment cinétique (jaune) et l'énergie totale (vert) sont constants. A noter que l'énergie totale est négative, comme elle doit être pour une orbite fermé.

```
20  double dT = 0.01;    //pas d'integration
21
22  //methode de Runge
23  double dist, vel, xacc, yacc;      //varialbes auxiliares
24  //redefinition de la vitesse initiale
25  dist = sqrt(xpos[0]*xpos[0] + ypos[0]*ypos[0]);
26  vel = sqrt(xvel[0]*xvel[0] + yvel[0]*yvel[0]);
27  xacc = -GM * xpos[0] / (dist*dist*dist);  //force ou acceleration, GM = 1
28  yacc = -GM * ypos[0] / (dist*dist*dist);
29  xvel[0] = xvel[0] - xacc*dT/2.;      //vitesse
30  yvel[0] = yvel[0] - yacc*dT/2.;
31  for (int i=1; i<steps; i++) {
32    dist = sqrt(xpos[i-1]*xpos[i-1] + ypos[i-1]*ypos[i-1]);
33    vel = sqrt(xvel[i-1]*xvel[i-1] + yvel[i-1]*yvel[i-1]);
34    xacc = -GM * xpos[i-1] / (dist*dist*dist);
35    yacc = -GM * ypos[i-1] / (dist*dist*dist);
36    xvel[i] = xvel[i-1] + xacc*dT;      //vitesse
37    yvel[i] = yvel[i-1] + yacc*dT;
38    xpos[i] = xpos[i-1] + xvel[i]*dT;    //position
39    ypos[i] = ypos[i-1] + yvel[i]*dT;
40    time[i] = i*dT;
41    ePot[i] = -GM * m / dist;
42    eCin[i] = 0.5 * m * vel*vel;
43    eTot[i] = ePot[i] + eCin[i];
```

```
44      //L = m * (r x v) !!! r et v ne sont pas toujours orthogonaux
45      l[i] = m * (xpos[i] * yvel[i] - ypos[i] * xvel[i]);
46    }
47
48    //partie graphique avec DISLIN
49    metafl("XWIN");    //XWIN ou PDF
50    page(3000, 3000);    //pour creer une image proportionee
51    disini();      //initialisation de DISLIN
52
53    //dessin de l'orbite
54    name("axe-X [UA]", "X");
55    name("axe-Y [UA]", "Y");
56    titlin("Orbite elliptique",1);
57    graf(-7.,3.,-7.,1.,-5.,5.,-5.,1.);
58    title();
59    thkcrv(5);
60    color("RED");
61    curve(xpos, ypos, steps);
62    //dessin du soleil
63    incmrk(-1);
64    color("GREEN");
65    marker(21);
66    double xx, yy;
67    xx=0.;
68    yy=0.;
69    curve(&xx,&yy,1);    //la fonction s'attend un adresse memoire
70    endgrf();
71    erase();
72    system("PAUSE");      //avant de dessiner le deuxieme graph
73
74    //dessin de l'energie
75    name("temps", "X");
76    name("energie", "Y");
77    titlin("Orbite elliptique",1);
78    titlin("jaune = moment cinetique",2);
79    titlin("vert = energie total",3);
80    titlin("rouge / blue = energie potentiele / cinetique",4);
81    color("FORE");
82    graf(0., dT*steps, 0., dT*steps/5., -2., 2., -2., 0.5);
83    title();
84    incmrk(0);
85    thkcrv(2);
86    color("RED");
87    curve(time, ePot, steps);
88    color("BLUE");
89    curve(time, eCin, steps);
90    color("GREEN");
91    curve(time, eTot, steps);
92    color("YELLOW");
93    curve(time, l, steps);
94
95    disfin();      //fin de DISLIN
96
97    return 0;
98  }
```

7. **Orbite avec frottement**
   Un satellite est sur une orbite circulaire proche de la Terre, à 100 km de la surface. La force de frottement due à l'atmosphère représente 0.01% de la force gravitationnelle (situation

hypothétique). La force de frottement est toujours parallèle à la vitesse instantanée. Estimez le temps qu'il faut pour que le satellite descende à 50 km.
On redéfinit la vitesse a $t - \Delta T/2$ pour utiliser la méthode d'intégration de Runge :

$$xvel[0] = xvel0 - xacc \times \frac{\Delta t}{2};$$
$$yvel[0] = yvel0 - yacc \times \frac{\Delta t}{2};$$

Pour calculer l'accélération à $t = 0$ on calcule la distance au centre gravitationnel et on utilise les formules développées pendant le cours (voir **Orbite3.cpp**) pour calculer les forces avec le frottement :

$$F_x = -\frac{GMm}{r^3}\left(\frac{x}{r} - C_X\frac{y}{r}\right)$$
$$F_y = -\frac{GMm}{r^3}\left(\frac{y}{r} + C_X\frac{x}{r}\right)$$

avec les instructions :

$$dist = \sqrt{xpos[0] \times xpos[0] + ypos[0] \times ypos[0]};$$
$$xacc = -GM/dist^3 \times (xpos[i] - C_X \times ypos[i]);$$
$$yacc = -GM/dist^3 \times (ypos[i] + C_X \times xpos[i]);$$

Ensuite on étudie l'orbite avec la méthode de Runge : la position à $t = (i+1) \times \Delta T$ est calculée en utilisant la vitesse au pas $i + 1$, qui correspond à la vitesse au temps $t = (i + 1) \times \Delta T/2$. A chaque étape la distance du satellite à la Terre est calculée et comparée à la hauteur finale limite (`altfin`). Quand la distance à la Terre est plus petite que `altfin` le programme sort de la boucle et affiche le nombre de tours autours de la Terre.

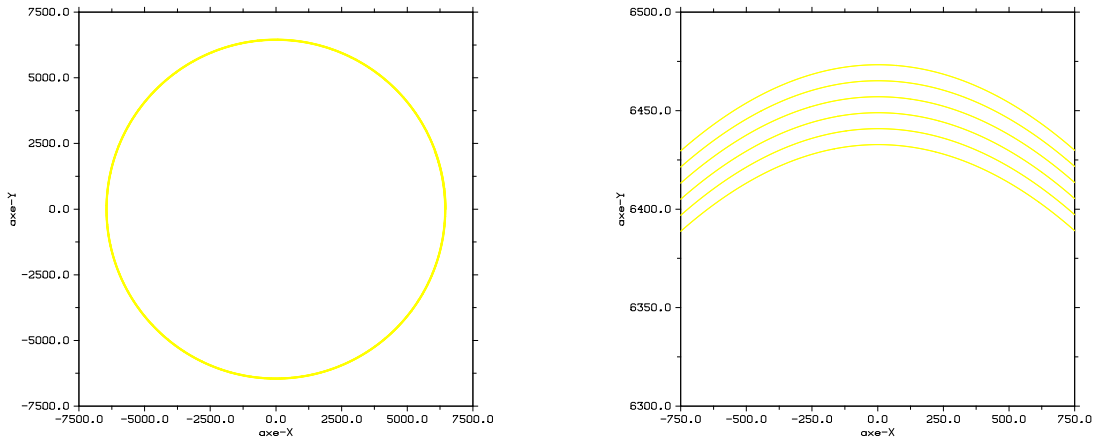

FIGURE 6 – L'orbite du satellite avec frottement, à gauche le mouvement complet et à droite un *zoom* qui montre la diminution du rayon de l'orbite à chaque rotation.

OrbFrottement.cpp

```
1 //etude d'une orbite avec frottement dans la haut atmosphere
2 #include <iostream>
3 #include <cmath>
```

```cpp
#include "dislin.h"

using namespace std;

int main() {
  const double RT = 6378.;    //en km
  const double GM = 398645.;    //G_N * masse_terre
  double altIni = 100.;
  double altFin = 50.;
  const double R0 = RT + altIni;
  const double V0 = sqrt(GM/R0);
  const double CX = 0.0001;      //coefficient de frottement

  const int steps = 50000;
  double xpos[steps], ypos[steps], xvel[steps], yvel[steps];
  //conditions initiales
  xpos[0] = R0; ypos[0] = 0.;
  xvel[0] = 0.; yvel[0] = V0;

  double dT = 2.;      //pas d'integration

  //methode de Runge
  double dist, xacc, yacc;      //variables auxiliares
  //redifinition de la vitesse initiale
  dist = sqrt(xpos[0]*xpos[0] + ypos[0]*ypos[0]);
  xacc = -GM / pow(dist,3) * (xpos[0] - CX*ypos[0]);    //acceleration
  yacc = -GM / pow(dist,3) * (ypos[0] + CX*xpos[0]);
  xvel[0] = xvel[0] -xacc*dT/2.;    //vitesse
  yvel[0] = yvel[0] -yacc*dT/2.;
  int nsteps = 0;
  int norbits = 0;
  //boucle
  for (int i=1; i<steps; i++) {
    dist = sqrt(xpos[i-1]*xpos[i-1] + ypos[i-1]*ypos[i-1]);
    xacc = -GM / pow(dist,3) * (xpos[i-1] - CX*ypos[i-1]);
    yacc = -GM / pow(dist,3) * (ypos[i-1] + CX*xpos[i-1]);

    xvel[i] = xvel[i-1] + xacc*dT;    //vitesse
    yvel[i] = yvel[i-1] + yacc*dT;

    xpos[i] = xpos[i-1] + xvel[i]*dT;    //position
    ypos[i] = ypos[i-1] + yvel[i]*dT;

    nsteps++;
    if (ypos[i-1]*ypos[i] < 0.) norbits = norbits + 1;
    if (dist < RT) {    //le satellite est tombe sur la Terre
      cout << "Le satellite est tombe sur la Terre !" << endl;
      break;
    }
  }

  cout << "apres " << int(nsteps*dT) << " seconds" << endl;
  cout << "et " << norbits/2 << " orbites";
  if (norbits%2 == 1) cout << " et demi";
  cout << endl;

  //partie graphique vec DISLIN
  metafl("XWIN");    //XWIN ou PDF
  page(3000, 3000);
  disini();      //initialisation de DISLIN
```

```
64    name("axe-X","X");
65    name("axe-Y","Y");
66    graf(-7500.,7500.,-7500.,2500.,-7500.,7500.,-7500.,2500.);
67    thkcrv(2);
68    color("yellow");
69    curve(xpos, ypos, nsteps);
70
71    disfin();       //fin de DISLIN
72
73    return 0;
74  }
```

8. **E cross B**
   Une particule de masse M et charge Q est soumise un champ électrique $\mathbf{E} = (0, E_y, 0)$ et magnétique $\mathbf{B} = (0, 0, B_z)$. La particule est au repos au point $P = (0, 0, 0)$ quand le champ électrique est allumé. Etudiez la trajectoire de la particule et dessinez la. Utilisez la méthode de Runge pour intégrer l'équation du mouvement. Quelle est la vélocité de dérive de la particule ? Et dans quelle direction ? Estimez la à partir de la trajectoire (paramètres : M = 0.001 kg, Q = 0.001 C, E = 100 V, B = 1 T).
   D'abord on analyse toutes les forces qui agissent sur la particule :

$$\mathbf{F} = m\mathbf{a} = Q(\mathbf{E} + \mathbf{v} \times \mathbf{B})$$
$$F_x = ma_x = Qv_yB$$
$$F_y = ma_y = -Qv_xB + QE$$
$$F_z = 0$$

   Vu que $v_{z,0} = 0$ et $F_z = 0$ la particule se déplace dans le plan $x - y$.
   On redéfinit les conditions initiales pour intgrer l'équation du mouvement avec la méthode de Runge :

$$a_{x,0} = \frac{Q}{m} v_{y,0}\ B$$
$$a_{y,0} = -\frac{Q}{m}\ v_{x,0}\ B + \frac{Q}{m}\ E$$

   et

$$v_{x,0} = v_{x0} - a_{x,0}\ \frac{\Delta t}{2}$$
$$v_{y,0} = v_{y0} - a_{y,0}\ \frac{\Delta t}{2}$$

   Puis on utilise le même algorithme que l'on a développé pour étudier les orbites, mais on remplace la force gravitationnelle par la force définie précedemment. La trajectoire de la particule est montrée dans la figure 7.
   Pour estimer la vitesse de dérive on mesure le temps entre deux changements successives de signe de la composante y de la vitesse.

   EcrossB.cpp

```
1  //etude de la trajectorie d'une particule dans
2  //un champ electromagnetique
3  #include <iostream>
4  #include <cmath>
5  #include "dislin.h"
6
```

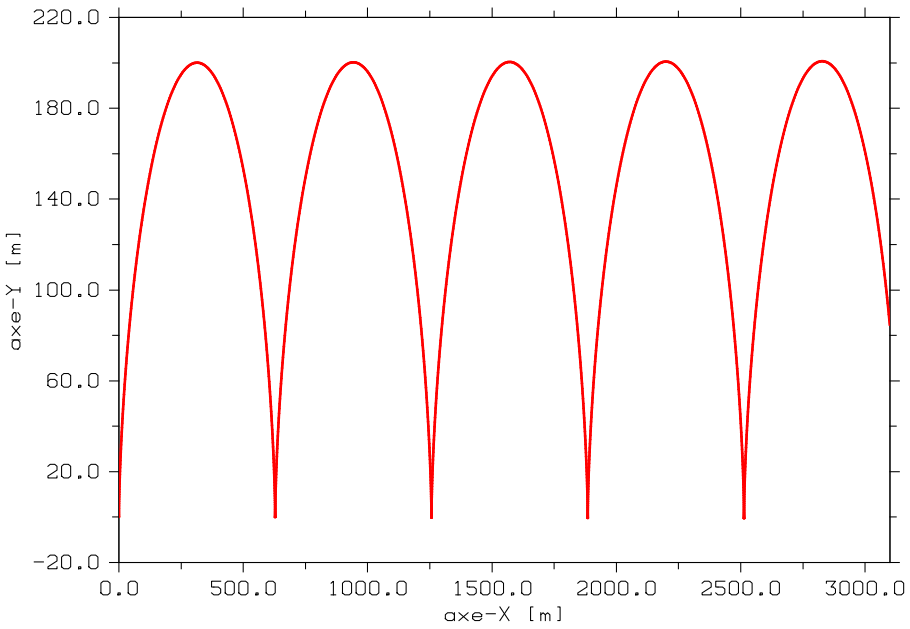

FIGURE 7 – Trajectoire d'une particule charge dans un champ électromagnétique.

```
7  using namespace std;
8
9  int main() {
10   //parametres
11   const int steps = 60000;
12   const double M = 0.001;        //masse
13   const double Q = 0.001;     //charge
14   const double E = 100.;      //E
15   const double B = 1.;      //B
16   double xpos[steps], ypos[steps], xvel[steps], yvel[steps];
17   //conditions initiales
18   xpos[0] = 0.0;    ypos[0] = 0.0;
19   xvel[0] = 0.0;    yvel[0] = 0.0;
20
21   double dT = 0.0005;    //pas d'integration
22
23   //methode de Runge
24   double xacc, yacc;       //varialbes auxiliares
25   double vdrift;
26   //redefinition de conditions initiales
27   xacc = Q/M*yvel[0]*B;     //acceleration
28   yacc = -Q/M*xvel[0]*B + Q/M*E;
29   xvel[0] = xvel[0] - xacc*dT/2.;       //vitesse
30   yvel[0] = yvel[0] - yacc*dT/2.;
31
32   for (int i=1; i<steps; i++) {
33     xacc = Q/M*yvel[i-1]*B;     //acceleration
34     yacc = -Q/M*xvel[i-1]*B + Q/M*E;
35     xvel[i] = xvel[i-1] + xacc*dT;       //vitesse
36     yvel[i] = yvel[i-1] + yacc*dT;
37     xpos[i] = xpos[i-1] + xvel[i]*dT;     //position
38     ypos[i] = ypos[i-1] + yvel[i]*dT;
39
40     //vitesse de derive
```

```
41      if (yvel[i-1]>0. && yvel[i]<0.) {
42        vdrift = xpos[i] / (i*dT);
43        cout << "La vitesse de derive est : " << vdrift << " m/s." << endl;
44      }
45    }
46
47    //partie graphique avec DISLIN
48    metafl("XWIN");    //XWIN ou PDF
49    disini();      //initialisation de DISLIN
50
51    //dessin de la trajectoire
52    name("axe-X [m]", "X");
53    name("axe-Y [m]", "Y");
54    titlin("Champ ElectroMagnetique",1);
55    graf(0.,3100.,0.,500.,-20.,220.,-20.,40.);
56    title();
57    thkcrv(5);
58    color("RED");
59    curve(xpos,ypos,steps);
60    disfin();     //fin de DISLIN
61
62    return 0;
63  }
```

9. **Système binaire**
   Etudiez un système binaire pour deux objets de même masse. Il vous faut écrire 4 équations différentielles de premier ordre pour chaque objet.
   Les étoiles ont une masse m_A et m_B respectivement, et la force qui agit sur chacune est donnée par les équations :

$$\begin{aligned} m_A\vec{a}_A &= -\frac{G_N m_B m_A}{|\vec{r}_B - \vec{r}_A|^3}(\vec{r}_B - \vec{r}_A) \\ m_B\vec{a}_B &= -\frac{G_N m_A m_B}{|\vec{r}_B - \vec{r}_A|^3}(\vec{r}_A - \vec{r}_B) \end{aligned} \tag{5}$$

   Les 4 équations différentielles pour l'étoile **A** sont :

$$\begin{aligned} \dot{v}_{x,A} &= -\frac{G_N m_B}{|\vec{r}_B - \vec{r}_A|^3}(x_B - x_A) \\ \dot{v}_{y,A} &= -\frac{G_N m_B}{|\vec{r}_B - \vec{r}_A|^3}(y_B - y_A) \\ \dot{x}_A &= v_{x,A} \\ \dot{y}_A &= v_{y,A} \end{aligned} \tag{6}$$

   Par la conservation du moment cinétique, la composante z peut être negligée. La distance entre les deux étoiles est $|\vec{r}_B - \vec{r}_A| = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$. Pour l'étoile **B** les équations sont similaires, mais en remplaçant A par B et vice versa.
   On a aussi ajouté une animation du système binaire. Pour ralentir l'exécution du programme on utilise la fonction `sleep`, qui malheureusement est différente sur Windows et Unix. Pour pouvoir utiliser le même programme sur Windows et Unix, on utilise les directives de précompilation pour choisir la bonne fonction.

   EtoileBinaire.cpp

```
1  //etude des orbites de deux etoiles binaires
2  //methode de Runge
```

```
3  #include <iostream>
4  #include <cmath>
5  #include "dislin.h"
6
7  /*
8  fonction SLEEP(ms)
9  malheureusement la fonction SLEEP est differente sur Windows et Unix :-(
10 pour pouvoir utiliser le meme programme sur Windows et Unix,
11 on utilise le directives de precompilation pour choisir la bonne fonction
12 */
13 #ifdef __UNIX__
14 #include <unistd.h>
15 #define SLEEP(ms) usleep(ms*1000);
16 #else
17 #include <windows.h>
18 #define SLEEP(ms) Sleep(ms);
19 #endif
20
21 using namespace std;
22
23 int main() {
24   const int steps = 30000;
25   const double GN = 1.;
26   //attention aux conditions initiales
27   //etoile A
28   const double MA = 1.;
29   double xposA[steps], yposA[steps], xvelA[steps], yvelA[steps];
30   xposA[0] = 0.3; yposA[0] = 0.;
31   xvelA[0] = 0.; yvelA[0] = 1.0;
32   //etoile B
33   const double MB = MA;
34   double xposB[steps], yposB[steps], xvelB[steps], yvelB[steps];
35   xposB[0] = -xposA[0]; yposB[0] = 0.;
36   xvelB[0] = 0.; yvelB[0] = -yvelA[0];
37
38   double dT = 0.005;
39
40   //methode de Runge
41   double dist, xaccA, yaccA, xaccB, yaccB;   //variables auxiliares
42   //redefinition des vitesses initiales
43   dist = sqrt(pow((xposA[0]-xposB[0]),2) + pow((yposA[0]-yposB[0]),2));
44   //etoile A
45   xaccA = -GN*MB / pow(dist,3) * (xposA[0]-xposB[0]);
46   yaccA = -GN*MB / pow(dist,3) * (yposA[0]-yposB[0]);
47   xvelA[0] = xvelA[0] -xaccA*dT/2.;   //vitesse
48   yvelA[0] = yvelA[0] -yaccA*dT/2.;
49   //etoile B
50   xaccB = -GN*MA / pow(dist,3) * (xposB[0]-xposA[0]);
51   yaccB = -GN*MA / pow(dist,3) * (yposB[0]-yposA[0]);
52   xvelB[0] = xvelB[0] -xaccB*dT/2.;   //vitesse
53   yvelB[0] = yvelB[0] -yaccB*dT/2.;
54   for (int i=1; i<steps; i++) {
55     //distance entre les etoiles
56     dist =
57       sqrt(pow((xposA[i-1]-xposB[i-1]),2)+pow((yposA[i-1]-yposB[i-1]),2));
58
59     //etoile A
60     xaccA = -GN*MB / pow(dist,3) * (xposA[i-1]-xposB[i-1]);
61     yaccA = -GN*MB / pow(dist,3) * (yposA[i-1]-yposB[i-1]);
62
```

```
63      xvelA[i] = xvelA[i-1] + xaccA*dT;    //vitesse
64      yvelA[i] = yvelA[i-1] + yaccA*dT;
65
66      xposA[i] = xposA[i-1] + xvelA[i]*dT;    //position
67      yposA[i] = yposA[i-1] + yvelA[i]*dT;
68
69      //etoile B
70      //A et B sont inverse dans le calcul de l'acceleration !!!
71      xaccB = -GN*MA / pow(dist,3) * (xposB[i-1]-xposA[i-1]);
72      yaccB = -GN*MA / pow(dist,3) * (yposB[i-1]-yposA[i-1]);
73
74      xvelB[i] = xvelB[i-1] + xaccB*dT;    //vitesse
75      yvelB[i] = yvelB[i-1] + yaccB*dT;
76
77      xposB[i] = xposB[i-1] + xvelB[i]*dT;    //position
78      yposB[i] = yposB[i-1] + yvelB[i]*dT;
79    }
80
81    //graphique DISLIN
82    metafl("XWIN");
83    page(3000, 3000);
84    disini();
85    name("axe-X","X");
86    name("axe-Y","Y");
87    titlin("Animation d'un systeme binaire",1);
88
89  /*
90    //dessin des orbites
91    graf(-1.,1.,-1.,1.,-1.,1.,-1.,1.);
92    thkcrv(2);
93    color("red");
94    curve(xposA, yposB, steps);
95    color("yellow");
96    curve(xposB, yposB, steps);
97  */
98
99    //animation DISLIN
100   double xx[1], yy[1];    //variables auxilieres
101   for (int j=1 ; j<steps; j++) {
102     //pour accelerer l'animation on ne dessine que toutes les 100
          iterations
103     SLEEP(5);     //pour arreter l'execution pendant 10 ms
104     if (j%10==0) {
105       erase();    //on efface le tableau
106       color("FORE");    //on remet un axe decrit en dessous en blanc
107       graf(-0.5,0.5,-0.5,0.2,-0.5,0.5,-0.5,0.2);
108       title();
109
110       //dessiner les etoiles
111       incmrk(-1);
112       color("GREEN");
113       marker(21);
114       xx[0]=xposA[j];
115       yy[0]=yposA[j];
116       curve(xx,yy,1);
117
118       incmrk(-1);
119       color("RED");
120       marker(21);
121       xx[0]=xposB[j];
```

```
122         yy[0]=yposB[j];
123         curve(xx,yy,1);
124
125         sendbf();
126         endgrf();
127       }
128     }
129
130     disfin();    //fin de DISLIN
131
132     return 0;
133 }
```

10. **Classe Planete**
    Voir le programme **Planete.cpp** développé pendant le cours. Le programme est aussi décomposé en plusieurs fichiers (voir le *projet* Planete) : `Planete.h` qui contient la définition de la classe `Planete`, `Planete.cpp` qui contient la définition des fonctions membres de la classe `Planete`, et `main.cpp` le programme principal qui utilise la classe `Planete`. Chaque planète est définie par son nom, son rayon orbital, sa période orbitale, l'excentricité de l'orbite et enfin l'inclinaison de l'orbite. La vitesse initiale et la position initiale de chaque planète sont calculées à partir de ces données.