UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

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

Corrigé du Contrôle Continu #3

`http://dpnc.unige.ch/~bravar/C++2015/CC3`

Professeur : Alessandro Bravar
Alessandro.Bravar@unige.ch

Université de Genève
Section de Physique

Semestre de printemps 2015

1 Questions

1. **Quelle est la différence entre les membres d'une classe (C++ !) de type `public` et `private` et pourquoi on utiliserait l'un ou l'autre ?**
   On peut accéder aux membres des type `public` depuis l'extérieur de la classe, tandis que on peut accéder aux membres `private` seulement depuis l'intérieur de la classe. Pour éviter de modifier accidentellement les données d'une classe depuis l'extérieur on déclare ces variables comme membres `private` de la classe.
2. **Qu'est-ce que c'est une fonction d'accès ?**
   On appelle *fonctions d'accès* les méthodes d'une classe, qui permettent de modifier les membres de type `private` de cette classe.
3. **Expliquez la surcharge d'une fonction et celle d'un opérateur.**
   On peut utiliser le même nom pour des fonctions différentes, typiquement pour des fonctions qui effectuent des taches similaires. On appelle cette fonctionnalité de C++ la surcharge des fonctions.
   On peut aussi utiliser le même opérateur ($+$, $-$, ...) pour effectuer des opérations similaires avec des types de données dérivés (les objets). Cette fonctionnalité est connue comme la surcharge des opérateurs.
4. **A quoi sert le constructeur d'une classe ?**
   Pour initialiser les membres d'une classe au moment de la déclaration de la classe, on utilise les constructeurs.
5. **Expliquez en quelque mot les idées à la base de la *programmation orientée objet* et pourquoi elle peut être utile.**
   Il s'agit de représenter les objets et leurs relations. Un objet représente un concept, une idée abstrait (par example une *voiture*) et il est implémenté par une classe. Chaque objet a une structure interne et un comportement, et il interagit avec ses pairs : on ne sépare plus les données et les fonctions, qui agissent sur ces données. Les classes facilitent la réutilisation et l'extension des éléments du code (programme), ce qui est très utile dans le développement de logiciels complexes.
   En résume la POO est basée sur :
   – l'abstraction de données caractéristiques d'un objet,
   – l'encapsulation (on peut utiliser un objet sans connaitre tous ces détails),
   – l'héritage ou dérivation, c.-à-d. la création de sous-classes à partir d'une autre classe et
   – la construction d'une classe déscendante, qui remplace des classes ancêtres (le polymorphisme).
6. **Décrivez la Méthode d'Euler pour résoudre des EDOs.**
   L'idée à la base de la Méthode d'Euler est de remplacer la dérivée éxacte par la différence finie calculée à droite à chaque itération, c.-à-d. utiliser la dérivée donnée au point t pour extrapoler la fonction jusqu'au point $t+\Delta t$. On dit que il s'agit d'une méthode de premier ordre, parce que l'éxpansion de la fonction (en série de Taylor) est coupé en premier ordre.
7. **Qu'est-ce qu'il faut faire pour résoudre une EDO de deuxième ordre ? Et de troisième ordre ?**
   D'abord, il faut transformer l'EDO de ordre p en un système de p EDOs de premier ordre.
8. **Expliquez la Méthode de Runge pour résoudre des EDOs.**
   La Méthode de Runge est une amélioration de la Méthode d'Euler : on utilise la dérivée calculée dans le point $t+\Delta t/2$ pour extrapoler la fonction jusqu'au point $t+\Delta t$, donc on évalue la dérivée au milieu de l'intervalle $(t,\, t+\Delta t)$. On peut montrer que cette Méthode tient aussi compte des termes de deuxième ordre dans l'approximation de la fonction. Donc, elle est plus précise que la Méthode d'Euler.

2 Exercices

Exercice 1 : Intégration d'une EDO de premier ordre
Voici la solution !
Boisson.cpp

```
1  //etude du refroidissement d'une boisson
2  #include <iostream>
3  #include <cmath>
4  #include "dislin.h"      //bibliotheque DISLIN
5
6  using namespace std;
7
8  int main() {
9    const int STEPMAX = 100;  //nombre de pas
10   double dt;                //pas d'integration
11
12   double tempExt = 10.;   //temperature exterieur
13   double temp[STEPMAX];   //tableau des temperatures
14   double t[STEPMAX];
15   double k = 0.7;         //coefficient de refroidissement degres / h
16
17   //conditions intiales
18   temp[0] = 20.;
19   t[0] = 0.;
20
21   //solution methode de Runge
22   cout << "Temperature exterieur constante" << endl;
23   dt = 0.1;
24   double tempM;
25   bool cond1 = false;
26   for (int step=0; step<STEPMAX-1; step++) {
27     //pour calculer temp dans (t_i + Delta t/2) on utilse la methode d'
         Euler
28     tempM = temp[step] - k*(temp[step]-tempExt)*dt/2.;
29     temp[step+1] = temp[step] - k*(tempM-tempExt)*dt;
30     t[step+1] = t[step] + dt;
31     if (!cond1 && temp[step+1] < 12.) {
32       cout << "Apres t = " << t[step+1] <<
33         " heurs la temperature de la boisson a baisse a 12 degres !" <<
             endl;
34       cond1 = true;
35     }
36   }
37
38   //initialisation de DISLIN
39   metafl("XWIN");    //XWIN ou PDF
40   disini();
41   smxalf("GREEK", "{", "}", 1);
42   name("temps (h)", "X");
43   name("T(t)", "Y");
44   titlin("Refroidissement", 1);
45   titlin("dT/dt = -k*(T(t)-T_ext)",2);
46   titlin("T(0) = 20, k = 0.7 / h", 3);
47   graf(0., 10., 0., 1., 5., 25., 5., 1.);
48   title();
49
50   //dessin
51   color("RED");
52   incmrk(-1);    //symboles au lieu de la curbe
```

```
53    marker(21);    //choix du symbole
54    curve(t,temp,100);
55
56    //temperature externe variable
57    cout << "\nTemperature exterieur variable" << endl;
58    dt = 0.1;
59    double tempB[STEPMAX];     //temperatures
60    //variation de la temperature exterieur :
61    //elle chute de 4 degres sur 8 heures
62    double a = 4./8.;
63
64    //conditions intiales
65    tempB[0] = 20.;
66    t[0] = 0.;
67
68    bool cond2 = false;
69    bool cond3 = false;
70    for (int step=0; step<STEPMAX-1; step++) {
71      //pour calculer temp dans (t_i + Delta t/2) on utilse la methode d'
          Euler
72      tempM = tempB[step] - k*(tempB[step]-(tempExt-a*t[step]))*dt/2.;
73      tempB[step+1] = tempB[step] - k*(tempM-(tempExt-a*t[step]))*dt;
74      t[step+1] = t[step] + dt;
75      if (!cond2 && t[step+1]>0.333) {
76        cout << t[step+1]*60 << " min apres 22h00, la temperature est de "
77           << tempB[step+1] << " degres" << endl;
78        cond2 = true;
79      }
80      if (!cond3 && t[step+1]>8.) {
81      cout << "Apres 06h00 la temperature de la boisson a chute a "
82           << tempB[step+1] << " degres" << endl;
83      cond3 = true;
84      }
85    }
86
87    //dessin
88    color("GREEN");
89    incmrk(-1);    //symboles au lieu de la curbe
90    marker(21);    //choix du symbole
91    curve(t,tempB,100);
92
93    //temperature exterieur variable - methode d'EULER
94    cout << "\nTemperature exterieur variable - methode d'Euler" << endl;
95    dt = 0.1;
96    double tempE[STEPMAX];     //temperatures
97
98    //conditions intiales
99    tempE[0] = 20.;
100   t[0] = 0.;
101
102   cond2 = false;
103   cond3 = false;
104   for (int step=0; step<STEPMAX-1; step++) {
105     tempE[step+1] = tempE[step] - k*(tempE[step]-(tempExt-a*t[step]))*dt;
106     t[step+1] = t[step] + dt;
107     if (!cond2 && t[step+1]>0.333) {
108       cout << t[step+1]*60 << " min apres 22h00, la temperature est de "
109            << tempE[step+1] << " degres" << endl;
110       cond2 = true;
111     }
```

```
112       if (!cond3 && t[step+1]>8.) {
113         cout << "Apres 06h00 la temperature de la boisson a chute a "
114              << tempE[step+1] << " degres" << endl;
115       cond3 = true;
116       }
117     }
118
119     //dessin
120     color("BLUE");
121     incmrk(-1);    //symboles au lieu de la curbe
122     marker(21);    //choix du symbole
123     curve(t,tempE,100);
124
125     //fin de dislin
126     disfin();
127
128     return 0;
129 }
```

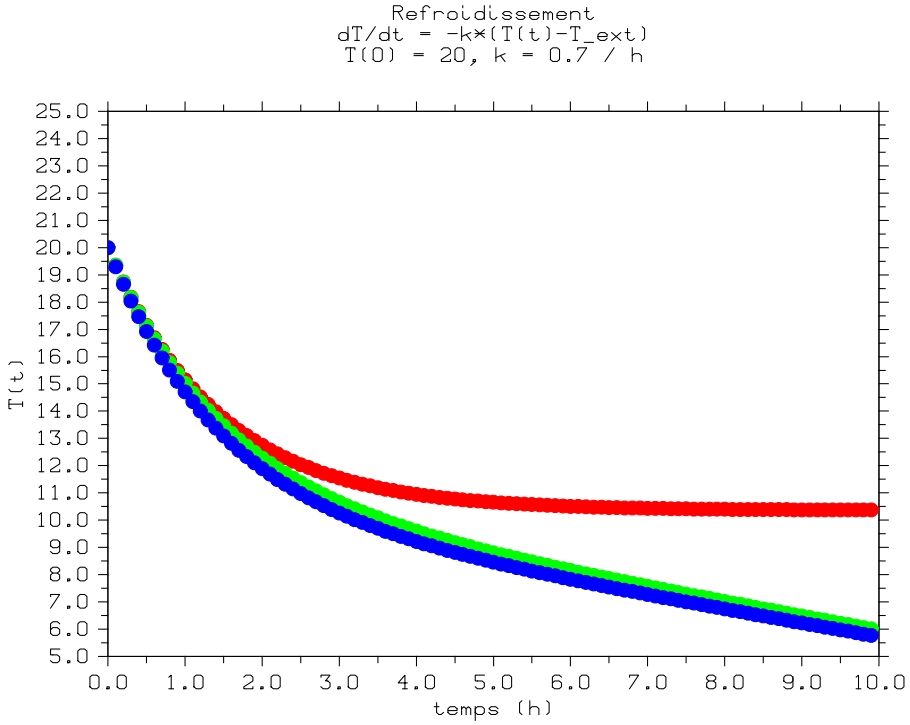

FIGURE 1 – Variation de la température de la boisson en fonction du temps : température externe constante (en rouge avec la méthode de Runge), température externe variable (en vert avec la méthode de Runge et en bleu avec la méthode d'Euler).

Exercice 2 : Développement d'une classe

Voici le programme !

Lorentz.cpp

```
1 #include<iostream>
2 #include<cmath>
3
4 using namespace std;
5
6 class LorentzVector {
7   public:
```

```
8       //Constructor:
9       LorentzVector(double t, double x, double y, double z)
10          {_t = t; _x = x; _y = y; _z = z;}
11
12      //Access functions:
13      double GetT() {return _t;}
14      double GetX() {return _x;}
15      double GetY() {return _y;}
16      double GetZ() {return _z;}
17
18      //Methods:
19      void Print();
20      LorentzVector operator+(LorentzVector vec);
21      double Contract();
22      LorentzVector LorentzTransformation(double beta);
23
24    private:
25      //Encapsulation of space-time coordinates:
26      double _t, _x, _y, _z;
27  };
28
29  int main() {
30    //Make a couple of Lorentz vectors and print them
31    LorentzVector A(-5,6,-1,2);
32    A.Print();
33    LorentzVector B(4,-6,4,-5);
34    B.Print();
35
36    //Test the overloaded + operator
37    LorentzVector C = A + B;
38    C.Print();
39
40    //Test the Lorentz boost method
41    cout << "Contraction before boost = " << C.Contract() << endl;
42    LorentzVector D = C.LorentzTransformation(0.7);
43    D.Print();
44    cout << "Contraction after boost = " << D.Contract() << endl;
45
46    return 0;
47  }
48
49  //Print space-time coordinates
50  void LorentzVector::Print() {
51    cout << "Lorentz vector is (";
52    cout << "t=" << _t;
53    cout << ", x=" << _x;
54    cout << ", y=" << _y;
55    cout << ", z=" << _z;
56    cout << ")" << endl;
57  }
58
59  //Overload + operator
60  LorentzVector LorentzVector::operator+(LorentzVector vec) {
61    return LorentzVector(vec.GetT()+_t, vec.GetX()+_x, vec.GetY()+_y, vec.
        GetZ()+_z);
62  }
63
64  //Contract Lorentz 4-vector
65  double LorentzVector::Contract() {
66    return _t*_t  - _x*_x - _y*_y - _z*_z;
```

```
67 }
68
69 //Lorentz boost - return a boosted Lorentz vector
70 //(Alternative: boost the class itself)
71 LorentzVector LorentzVector::LorentzTransformation(double beta) {
72   double gamma = 1. / sqrt(1. - beta*beta);
73   double T = gamma * (_t - beta*_x);
74   double X = gamma * (_x - beta*_t);
75   return LorentzVector(T, X, _y, _z);
76 }
```