UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES

Méthodes informatiques pour physiciens

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

Corrigé 3

Professeur : Alessandro Bravar
Alessandro.Bravar@unige.ch

Université de Genève
Section de Physique

Semestre de printemps 2015

Références :

M. Michelou et M. Rieder
Programmation orientée objets en C++

J.C. Chappelier et F. Seydoux
C++ par la pratique

B. Stroustrup
Programmation *Principes et pratique avec C++*

`http://dpnc.unige.ch/~bravar/C++2015/L3` :
pour les notes du cours, les exercices et les corrigés

3.1 Questions

1. **Quelle est la différence entre la déclaration et la définition d'une fonction ?**
   La déclaration d'une fonction, que l'on appelle également *prototype* de la fonction, spécifie simplement le type renvoyé par la fonction et la liste des paramètres associés. La déclaration ne dit pas ce que la fonction fait. La déclaration de la fonction est obligatoire avant l'utilisation de la fonction (comme pour les variables) et ne doit pas être contenue dans les autres fonctions (y compris la fonction `main`).

   La définition de la fonction contient le corps de la fonction décrivant son action (suite d'instructions). Comme cette définition doit correspondre à la fonction déclarée, on spécifie encore la signature de la fonction (le type de renvoi et la liste de paramètres). La définition peut être placée partout en-dehors du `main` (d'habitude en fin de programme) ou dans une bibliothèque quelconque.
2. **Quelle est la différence entre le passage par valeur et le passage par référence ?**
   Le passage d'une variable par valeur ne fait qu'affecter la valeur de celle-ci à une copie locale de la variable dans la fonction. La fonction agit sur cette copie dont la portée est limitée au corps de la fonction. Ainsi, la variable passée à la fonction ne peut pas être modifiée en-dehors de la fonction.
   Par contre, lors d'un passage par référence, on passe l'adresse mémoire de la variable. Ainsi la fonction agit directement sur la variable passée et peut donc modifier la valeur stockée à l'emplacement mémoire correspondant.
3. **Quel peut être l'avantage d'un passage par référence constante ?**
   Le passage par référence constante ne permet pas la modification de la variable dans la fonction. La variable se comporte comme une variable déclarée constante. Le deuxième avantage d'un passage par référence est que celui-ci évite la création d'une copie locale de la variable passée (pas de duplication). Ceci est donc très utile lorsque la variable passée est par exemple un long tableau nécessitant beaucoup de mémoire dans l'espace alloué au programme.
4. **Qu'entend-on par paramètre en** *lecture seule / lecture-écriture* **?**
   Par *lecture seule* on fait référence au fait que lors d'un passage par valeur, on ne peut pas modifier la variable passée à la fonction. C'est comme si la fonction ne pouvait que lire la valeur de la variable et utiliser seulement sa valeur. Par *lecture-écriture* on signifie que lors d'un passage par référence la fonction peut modifier la variable passée, donc aussi sa valeur dans la mémoire ; c'est comme si la fonction pouvait lire la valeur et récrire par-dessus.
5. **Quentend-on par** *récursivité* **?**
   En C++ une fonction peut aussi appeler soi-mme, c.-à-d. que la valeur renvoyé par la fonction est passée à la même fonction une nouvelle fois.
6. **A quoi servent les directives des prétraitement ?**
   Les directives des prétraitement sont traitées par le pré-compilateur et ils permettent p. ex. d'inclure les fichiers en-têtes. Il ne s'agit pas des programmation C++, mais de manipulation du texte.

3.2 Trouvez l'erreur !

Les programmes corrigés sont donnés ci-dessous. Les erreurs sont les suivantes :

1. Le prototype (ainsi que la définition) de la fonction annonce un type de renvoi `void` dans les deux programmes, c'est-à-dire que la fonction n'est pas censée retourner des valeurs.

Or la fonction retourne la valeur `4.*x` (`x*x`) où `x` est déclaré comme `double`. Le prototype de la fonction doit donc lui aussi annoncer un type de renvoi `double`.

2. La variable `a` est passée par valeur à la fonction et la valeur retournée est affectée à la variable `b`. Lors du passage à la fonction (par valeur, référence ou pointeur) le type de la variable passée ne doit pas être à nouveau spécifié. Dans le deuxième programme la variable `a` elle-même n'est pas passée.
3. Dans le deuxième programme il n'y a pas de `;` entre la parenthèse `)` et l'accolade { dans la définition de la fonction.

```
1  //erreur 1
2  #include <iostream>
3
4  using namespace std;
5
6  double maFonction(double x);
7
8  int main() {
9    double a=10., b;
10   b = maFonction(a);
11   cout << a << " " << b << endl;
12   return 0;
13 }
14
15 double maFonction(double x) {
16   return 4.*x;
17 }
```

```
1  //erreur 2
2  #include <iostream>
3
4  using namespace std;
5
6  double maFonction(double x);
7
8  int main() {
9    double a=10., b;
10   b = maFonction(a);
11   cout << a << " " << b << endl;
12   return 0;
13 }
14
15 double maFonction(double x) {
16   return x*x;
17 }
```

3.3 Exercices

Exercice 1 : Le programme utilise la fonction `double log10(double x)` définie dans la bibliothèque `cmath`. Pour inclure cette bibliothèque il faut ajouter la commande `#include <cmath>` au début du programme. L'utilisateur entre au clavier la valeur dont on veut calculer le logarithme en base 10 en utilisant la fonction `cin`. Le programme vérifie ensuite que la valeur introduite est positive et calcule la valeur du logarithme en appelant la fonction `double log10(double x)`. Ces instructions sont mises dans une boucle *do-while* pour répéter le calcul jusqu'à ce que l'utilisateur choisisse de sortir en répondant

`OUI` à la question posée par l'ordinateur.

Ex_1.cpp

```
1  //exemple utilisant une fonction mathematique
2  #include <iostream>
3  #include <cmath>      //bibliotheque mathematique
4
5  using namespace std;
6
7  int main() {
8    bool reponse = false;   //variable de controle pour la boucle do
9    do {
10     double x;
11     cout << "Entrez un nombre positif : ";
12     cin >> x;
13
14     if (x<=0.) {
15       cout << "Le nombre est <= 0 ! STOP" << endl;
16       system("PAUSE");
17       return 0;
18     }
19
20     cout << "Son logarithme en base 10 est : " << log10(x) << endl;
21
22     cout << "Voulez-vous calculer un autre logarithme (OUI=1, NO=0) ? ";
23     cin >> reponse;
24     cout << endl;
25   } while (reponse);
26
27   return 0;
28 }
```

Exercice 2 : Le programme utilise la fonction `int maximum(int n, int n2)` construite par le programmeur. Pour introduire cette fonction il faut déclarer la fonction au début du programme avec l'instruction `maximum(int n, int n2);`. La définition de la fonction est donnée au dessous de la fonction `int main()` : la fonction renvoie la valeur de `n1` si `n1` est plus grand de `n2`, autrement elle renvoie la valeur de `n2`.

Maximum.cpp

```
1  //exemple utilisant une fonction
2  #include <iostream>
3
4  using namespace std;
5
6  //declaration de la fonction maximum
7  int maximum(int n1, int n2);
8
9  int main() {
10   int n1, n2;
11   cout << "Entrez deux nombres entiers !" << endl;
12   cout << "n1 = "; cin >> n1;
13   cout << "n2 = "; cin >> n2;
14
15   int max;
16   max = maximum(n1,n2);
17   cout << "Le maximum des deux nombres est : " << max << endl;
18   //on peut aussi ecrire
19   //cout << "Le maximum des deux nombres est : " << maximum(n1,n2) << endl;
20
```

```
21    return 0;
22  }
23
24  //corps de la fonction maximum
25  int maximum(int n1, int n2) {
26    if (n1>n2) return n1;
27  //on n'a pas besoin de l'instruciton alternative (if - else)
28  //avec return on sort de la fonction et on revien au programme appelant
29    return n2;
30  }
```

Exercice 3 : Le programme utilise deux fonctions différentes `double surface(double r)` et `double volume(double r)` déclarées avant la fonction principale `int main()`. Le corps des ces fonctions, placés à la fin du programme, renvoient les valeurs de la surface et du volume.

Sphere_v2.cpp

```
1   //exemple utilisant des fonctions
2   #include <iostream>
3   #include <cmath>      //bibliotheque mathematique
4
5   using namespace std;
6
7   //declaration de la fonction surface
8   double surface(double rayon);
9   //declaration de la fonction volume
10  double volume(double rayon);
11
12  int main() {
13    double rayon;
14    cout << "Entrez le rayon de la sphere : ";
15    cin >> rayon;
16
17    //calcul de la surface avec la fonction surface
18    cout << "La surface de la sphere est : " << surface(rayon) << endl;
19
20    //calcul du volume avec la fonction volume
21    double vol = volume(rayon);
22    cout << "Le volume de la sphere est :  " << vol << endl;
23
24    return 0;
25  }
26
27  //corps de la fonction surface
28  double surface(double rayon) {
29    double x = 4. * M_PI * pow(rayon,2.);
30    return x;
31  }
32
33  //corps de la fonction volume
34  double volume(double rayon) {
35    double x = 4./3. * M_PI * pow(rayon,3.);
36    //on peut  crire  aussi return 4./3. * M_PI * pow(rayon,3.); }
37    return x;
38  }
```

Exercice 4 : Le programme utilise la fonction `double fahren2kelvin(double t)` déclarée en tête du programme. Le *prototype* de la fonction est introduit avec l'instruction

`double fahren2kelvin(double t);`. La déclaration de la fonction, placée au dessous de la fonction `int main()`, convertit la température (valeur de la variable `temp` introduite par l'utilisateur) avec la formule :

$$K = \frac{5.}{9.} \times (F - 32.) + 273.15 \, .$$

Far2Kel.cpp

```
1  //conversion de degres Fahrenheit en Kelvin
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  //declaration de la fonction de conversion F -> K
8  double fahren2kelvin(double t);
9
10 int main() {
11   //saisi des donnees
12   double temp;
13   cout << "Entrez la temperature en Fahrenheit : ";
14   cin >> temp;
15
16   //conversion F -> K
17   double kelvin;
18   kelvin = fahren2kelvin(temp);
19   cout << "La temperature en Kelvin est : " << kelvin << endl;
20   //on peut aussi ecrire
21   //cout << "La temperature en Kelvin est : " << fahren2kelvin(temp) <<
         endl;
22
23   return 0;
24 }
25
26 //corps de la fonction de conversion F -> K
27 double fahren2kelvin(double t) {
28   double kelvin;
29   kelvin = 5./9.*(t-32.) + 273.15;
30   return kelvin;
31   //on peut aussi ecrire
32   //return 5./9*(t-32.) + 273.15;
33 }
```

Exercice 5 : Le programme utilise la fonction `double volumeCylindre(double rayon, double hauteur)` déclarée en-tête du programme. Le *prototype* de la fonction est introduit avec l'instruction `double volumeCylindre(double rayon, double hauteur);`.

VolumeCylindre.cpp

```
1  //parametres par defaut
2  #include <iostream>
3  #include <cmath>      //bibliotheque mathematique
4
5  using namespace std;
6
7  //declaration de la fonction volume avec des valeur par defaut
8  double volumeCylindre(double rayon = 10., double hauteur = 50.);
9
10 int main() {
```

```
11    cout << "volumeCylindre(20.,40.)    " << volumeCylindre(20.,40.) << endl;
12    //resultat : pi * 20**2 * 40
13
14    cout << "volumeCylindre(20.)    " << volumeCylindre(20.) << endl;
15    //resultat : pi * 20**2 * 50
16
17    cout << "volumeCylindre()    " << volumeCylindre() << endl;
18    //resultat : pi * 10**2 * 50
19
20    return 0;
21  }
22
23  //corps de la fonction volume
24  double volumeCylindre(double rayon, double hauteur) {
25    double vol = M_PI * pow(rayon,2) * hauteur;
26    return vol;
27  }
```

Exercice 6 : Voir le programme **Surcharge.cpp** introduit pendant la leçon.

Surcharge.cpp

```
1   //exemple de surcharge d'une fonction
2   #include <iostream>
3   #include <iomanip>
4
5   using namespace std;
6
7   //declarations de la fonctionne cube
8   //la fonction est surchargee 3 fois
9   int cube(int n);
10  float cube(float n);
11  double cube(double n);
12
13  int main() {
14    int x = 5;
15    cout << "Le cube d'un entier est " << cube(x) << endl;
16
17    float y = 5.5555;
18    cout << "Le cube d'un float est " << setw(15) << setprecision(10) << cube
          (y) << endl;
19
20    double z = 5.5555;
21    cout << "Le cube d'un duble est " << setw(15) << setprecision(10) << cube
          (z) << endl;
22
23    return 0;
24  }
25
26  //definition de la fonction volume : int
27  //les calcules sont effectues entre entiers et le resultat est un entier
28  int cube(int n) {
29    return n*n*n;
30  }
31
32  //definition de la fonction volume : float
33  //les calcules sont effectues entre floats et le resultat est un flaot
34  float cube(float n) {
35    return n*n*n;
36  }
37
```

```
38 //definition de la fonction volume : double
39 //les calcules sont effectues entre doubles et le resultat est un double
40 double cube(double n) {
41   return n*n*n;
42 }
```

Exercice 7 : Le programme **Fibonacci_rec.cpp** affiche les premiers `n` termes de la suite de Fibonacci en utilisant la fonction `int fibonacci(int n)`, qui calcule la suite par récursivité selon sa définition :

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-2} + F_{n-1} \quad (n \geq 2) \end{cases}$$

Fibonacci_rec.cpp

```
1  //calcul de la suite de Fibonacci par recursivite
2  #include <iostream>
3
4  using namespace std;
5
6  //declaration de la fonction
7  int fibonacci(int n);
8
9  int main() {
10   int n;
11   cout << "Entrez le nombre de termes de la suite : ";
12   cin >> n;
13   cout << "Les premiers " << n << " terms de la suite de Fibonacci sont : "
         << endl;
14
15   int f = 0;
16   for (int i=0; i<n; i++) {
17     f = fibonacci(i);
18     cout << f << endl;
19   }
20
21   return 0;
22 }
23
24 //definition de la fonction
25 int fibonacci(int n) {
26   if (n==0) return 0;
27   if (n==1) return 1;
28   if (n>=2) return fibonacci(n-2) + fibonacci(n-1);
29 }
```

La fonction récursive `fibonacci` n'est pas du tout efficace et le programme *tourne* très lentement. On peut obtenir le même résultat p.ex. avec une boucle, comme dans **Fibonacci.cpp** où on a réécrit la fonction `fibonacci` en utilisant une boucle `for`. Comparez le temps d'exécution de ces deux programmes, p.ex. pour `n = 40`.

Fibonacci.cpp

```
1  //calcul de la suite de Fibonacci
2  #include <iostream>
3
4  using namespace std;
5
```

```
6  //declaration de la fonction
7  int fibonacci(int n);
8
9  int main() {
10   int n;
11   cout << "Entrez le nombre de termes de la suite : ";
12   cin >> n;
13   cout << "Les premiers " << n << " terms de la suite de Fibonacci sont : "
         << endl;
14
15   int f = 0;
16   for (int i=0; i<n; i++) {
17     f = fibonacci(i);
18     cout << f << endl;
19   }
20
21   return 0;
22 }
23
24 //definition de la fonction
25 int fibonacci(int n) {
26   int fib0 = 0;
27   int fib1 = 1;
28   int fib2;
29   if (n==0) return 0;
30   if (n==1) return 1;
31   for (int i=2; i<=n; i++) {
32     fib2 = fib0 + fib1;
33     fib0 = fib1;
34     fib1 = fib2;
35   }
36   return fib2;
37 }
```

Exercice 8 : Le programme **Pascal.cpp** affiche sur l'écran le triangle de Pascal jusqu'à la ligne choisie par l'utilisateur. En-tête du programme, on déclare les fonctions qui calculent la factorielle d'un nombre et le coefficient binomial $\begin{pmatrix} n \\ k \end{pmatrix}$ avec les instructions `int fact(int n);` et `int comb(int n, int k);`. Les fonctions sont définies à la fin du programme. La fonction `fact` calcule la valeur de la factorielle avec une boucle `for`. La fonction `comb` utilise la fonction `fact` pour calculer le nombre des combinaisons selon la formule :

$$\begin{pmatrix} n \\ k \end{pmatrix} = \frac{n!}{(k!)(n-k)!}$$

Les deux boucles `for` dans la fonction `main` affichent les valeur calculées en les organisant sous forme triangulaire. On utilise la fonction `setw(n)` pour la mise en page.

Pascal.cpp

```
1  //traingle de Pascal
2  #include <iostream>
3  #include <iomanip>
4
5  using namespace std;
6
7  int fact(int n);      //declaration de la fonction factorielle
8  int comb(int n, int k);   //declaration de la fonction combinatoire
9
```

```
10 int main() {
11   int lignes;
12   cout << "Triangle de Pascal " << endl;
13   cout << "Entrez le nombre de lignes : ";
14   cin >> lignes;
15   cout << endl;
16
17   //boucle sur le lignes
18   for (int i=0; i<lignes; i++) {
19     //imprime un espace pour former un tableau triangulaire
20     for (int j=1; j<lignes-i; j++) cout << "    " ;
21     //calcule les element du triangle pour la ligne i
22     for (int j=0; j<=i; j++) cout << setw(6) << comb(i,j);
23     cout << endl;
24   }
25   cout << endl;
26
27   return 0;
28 }
29
30 //corps de la fonction factorielle
31 int fact(int n) {
32   int f = 1;
33   for (int i=2; i<=n; i++)
34     f = f * i;
35   return f;
36 }
37
38 //corps de la fonction combinatoire
39 int comb(int n, int k) {
40   if (n<0 || k<0 || k>n) return 0;
41   int cnk;
42   cnk = fact(n) / (fact(k)*fact(n-k));
43   return cnk;
44 }
```

Exercice 9 : Voir le programme **Reference.cpp** introduit pendant la leçon.

Exercice 10 : La fonction `void swap(int n1, int n2)` dans **Swapp.cpp** ne produit pas le résultat souhaité parce que les variables `a` et `b` sont passées par valeur : la fonction copie la valeur des ces deux variables dans des variables locales de la fonction (des cellules de mémoire différentes) et agit sur ces variables locales. Quand on sort de la fonction `swap`, les variables locales sont détruites et la fonction `main` agit sur les variables `a` et `b` qui n'ont été jamais modifiées. Au contraire, dans le programme **Swap_ref.cpp** les variables `a` et `b` sont passées par référence à la fonction `void swap(int &n1, int &n2)` : l'adresse mémoire des variables `a` et `b` est transmise à la fonction `swap`. Aucune copie locale de `a` et `b` n'est créée. Les changements sont effectués sur les cellules de mémoire de `n1` et `n2`, qui correspondent aux cellules mémoire de `a` et `b`. Les modifications sont donc définitives.

3.4 Problème

Plans inclinés : Le programme **PlansInclines.cpp** calcule la position d'une balle lâchée d'une hauteur `h` fournie par l'utilisateur sur un plan incliné avec inclination α. La balle rebondit entre deux plans symétriques. Les rebonds sont parfaitement élastique. A chaque rebond le programme affiche la position du point de rebond, la vitesse de la balle et le temps entre deux rebonds.

Le programme utilise les cinq fonctions suivantes pour étudier le rebond :

```
double angleRebond(double theta, double alpha, double x)

double solutionGauche(double v, double theta, double alpha, double x0,
double y0)

double solutionDroite(double v, double theta, double alpha, double x0,
double y0)

double intersect(double v, double theta, double alpha, double x0, double
y0)

double angleIncidence(double v, double theta, double alpha, double x0,
double y0, double x)
```

déclarées en-tête du programme.
Au début le programme saisit la hauteur initiale h, la position horizontal initiale x_0, l'inclination des plans et le nombre de rebonds que l'on souhaite calculer. Ensuite le programme calcule la position y_0 du premier rebond et la vitesse de la balle et affiche les résultats. Si y_0 est en-dessous du plan incliné le programme saisit des nouvelles valeurs initiales h et x_0. On utilise la fonction `angleRebond` pour calculer l'angle du rebond θ_{Out}. On considère séparément le cas du rebond sur le plan à droite ($x > 0$) et le plan à gauche ($x < 0$). La trajectoire de la balle après le rebond est décrite par une parabole d'équation

$$y = y_0 + \tan(\theta_{Out})(x - x_0) - g(x - x_0)^2/(2v_0^2\cos(\theta_{Out})^2) \ .$$

Ensuite on calcule l'intersection de cette parabole avec les plans inclinés

$$y = \tan\alpha \mid x \mid \ .$$

Les solutions sont de la forme

$$x = \frac{-B_i \pm \sqrt{(B_i^2 - 4AC)}}{(2A)}$$

avec

$$A = g/(2v_0^2\cos(\theta_{Out})^2)$$
$$B_1 = -x_0 g/(v_0^2\cos(\theta_{Out})^2) - \tan(\theta_{Out}) - \tan(\alpha)$$
$$B_2 = -x_0 g/(v_0^2\cos(\theta_{Out})^2) - \tan(\theta_{Out}) + \tan(\alpha)$$
$$C = gx_0^2/(2v_0^2\cos(\theta_{Out})^2) + \tan(\theta_{Out})x_0 - y_0$$

où B_1 correspond à la solution à droite et B_2 à la solution à gauche.
La fonction `intersect` résout ces équations pour chaque plan et choisit la bonne intersection. Pour calculer les intersections on utilise les fonction `solutionGauche` et `solutionDroite`. Enfin la fonction `angleIncidence` calcule le valeur de l'angle de la trajectoire quand la balle touche le plan incliné.
Dans le programme on fait attention que l'angle du rebond ne soit pas $\pi/2$. Dans ce cas le rebond est vertical et le programme considère séparément cette possibilité. Ensuite il calcule l'angle d'incidence suivant θ_{In} et le temps entre deux rebonds. Puis il calcule l'angle de rebond θ_{Out} et la vitesse de la balle et il met à jour les valeurs de x_0, y_0, v_0. Enfin le programme affiche ces résultats.
PlansInclines.cpp

```
1  //etude de rebonds d'une balle sur deux plans inclines
2  #include <iostream>
3  #include <cmath>
4  #include <iomanip>
5
6  using namespace std;
7
8  //declarations des fonctions
9  double angleRebond(double theta, double alpha, double x);
10 double solutionGauche(double v, double theta, double alpha, double x0,
       double y0);
11 double solutionDroite(double v, double theta, double alpha, double x0,
       double y0);
12 double intersect(double v, double theta, double alpha, double x0, double y0
       );
13 double angleIncidence(double v, double theta, double alpha, double x0,
       double y0, double x);
14
15 //on utilise des unites MKS
16 const double GN = 9.81;
17
18 int main() {
19   //saisi des parametre initials
20   double h;
21   cout << "Entrez la hauteur initiale h en metres : ";
22   cin >> h;
23   double x0;
24   cout << "\nEntrez la position horizontale x en metres : ";
25   cin >> x0;
26   double alpha;
27   cout << "\nEntrez l'angle d'inclination des plans en degres : ";
28   cin >> alpha;
29   alpha *= M_PI/180.;   //conversion de l'angle en radians
30   int nrebond;
31   cout << "\nEntrez le nombre de rebonds : ";
32   cin >> nrebond;
33   cout << endl;
34
35   while(h <= fabs(tan(alpha)*x0)) {
36     cout << "La hauteur initiale est au dessous du plan." << endl;
37     cout << "Essayez des nouvelles donnees (meme inclination)" << endl;
38     cout << "Entrez la nouvelle hauteur initiale h : ";
39     cin >> h;
40     cout << "\net la position horizontale x : " << endl;
41     cin >> x0;
42   }
43
44   //premier rebond
45   double y0, v0, t0;
46   y0 = fabs(tan(alpha)*x0);
47   v0 = sqrt(2.*GN*(h-y0));
48   t0 = sqrt(2.*(h-y0)/GN);
49
50   //affichage
51   cout << endl << endl;
52   cout << "rebond   position x0   position y0    vitesse v0         temps";
53   cout << endl << endl;
54   cout << setw(6) << 1
55        << setw(14) << setprecision(7) << x0
```

```
56          << setw(14) << setprecision(7) << y0
57          << setw(14) << setprecision(7) << v0
58          << setw(14) << setprecision(7) << t0 << endl;
59
60    double thetaIn, thetaOut;
61    thetaIn = -M_PI/2.;    //angle d'incidence initial en radians
62    thetaOut = angleRebond(thetaIn,alpha,x0);
63
64  /*
65  Ensuite, il faut determiner l'intersection d'une parabole :
66  y = y0 + tan(thetaOut)*(x-x0) - g*(x-x0)^2/(2*v0^2*cos(thetaOut)^2)
67  avec la fonction qui decrit les plans y = tan(alpha)*|x|.
68
69  On doit donc resoudre deux equations quadratiques :
70  (1) x < 0 :
71  g*(x-x0)^2/(2*v0^2*cos(thetaOut)^2)-tan(thetaOut)*(x-x0)-y0-tan(alpha)*x=0
72    pour y = -tan(alpha)*x    --> x < 0
73  ou
74  (2) x > 0 :
75  g*(x-x0)^2/(2*v0^2*cos(thetaOut)^2)-tan(thetaOut)*(x-x0)-y0+tan(alpha)*x=0
76    pour y = tan(alpha)*x    --> x > 0
77
78  Les solutions de ces equations sont de la forme :
79    x = (-B_i +- sqrt( B_i^2- 4*A*C ))/(2*A)      i=1,2
80  avec
81    A =  g/(2*v0^2*cos(thetaOut)^2),
82    B1= -x0*g/(v0^2*cos(thetaOut)^2) - tan(thetaOut) -tan(alpha)
83    B2= -x0*g/(v0^2*cos(thetaOut)^2) - tan(thetaOut) +tan(alpha)
84    C =  g*x0^2/(2*v0^2*cos(thetaOut)^2) + tan(thetaOut)*x0 -y0
85  */
86
87    double x, t;
88    for(int compt=1; compt<nrebond; compt++) {
89      if (fabs(thetaOut-M_PI/2.)>0.00001) {
90        x = intersect(v0,thetaOut,alpha,x0,y0);
91        thetaIn = angleIncidence(v0,thetaOut,alpha,x0,y0,x);
92      }
93      else {
94        x = x0;
95        thetaIn = thetaOut;
96      }
97
98      if (fabs(x0-x)>0.00001)
99        t = fabs((x0-x) / (v0*cos(thetaOut)));
100     else
101       t = 2.*v0*fabs(sin(thetaOut)) / GN;
102
103     thetaOut = angleRebond(thetaIn,alpha,x);
104     x0 = x;
105     y0 = tan(alpha)*fabs(x);
106     v0 = sqrt(2.*GN*(h-y0));
107
108     //affichage
109     cout << setw(6) << compt+1
110          << setw(14) << setprecision(7) << x0
111          << setw(14) << setprecision(7) << y0
112          << setw(14) << setprecision(7) << v0
113          << setw(14) << setprecision(7) << t << endl;
114   }
115
```

```
116   return 0;
117 }
118
119 //definition de la fonction qui calcule l'angle de rebond
120 double angleRebond(double theta, double alpha, double x) {
121   if (x<0.)      //rebond sur le plan a gauche
122     return -2.*alpha-theta;
123   else if (x>0.)      //rebond sur le plan a droite
124     return 2.*alpha-theta;
125   else      //rebond entre les deux plans (x=0): approxime a un plan
        horizontal
126     return M_PI-theta;
127 }
128
129 //definition de la fonction qui calcule l'intersection entre
130 //la trajectoire de la balle et le plan incline
131 double intersect(double v, double theta, double alpha, double x0, double y0
      ) {
132   double x1 = solutionGauche(v,theta,alpha,x0,y0);
133   double x2 = solutionDroite(v,theta,alpha,x0,y0);
134   if (x1>0 && x2>=0)
135     return x2;
136   else if (x2<0 && x1<0)
137     return x1;
138   else {
139     cout << "Il y a quelque probleme, deux solutions !? " << endl;
140     cout << "x1 = " << setw(10) << setprecision(7) << x1
141          << ", x2 = " << setw(10) << setprecision(7) << x2 << endl;
142     return -99999.99;
143   }
144 }
145
146 //definition de la fonction qui calcule l'intersection a gauche (x < 0)
147 double solutionGauche(double v, double theta, double alpha, double x0,
      double y0) {
148   double den = v*v*cos(theta)*cos(theta);
149   double a = GN/(2.*den);
150   double b1= -tan(theta) - GN*x0/den - tan(alpha);
151   double c = GN*x0*x0/(2.*den) + tan(theta)*x0 -y0;
152   double x1, x2;
153   double disc = b1*b1-4.*a*c;
154
155   if (disc>0) {
156     x1 = (-b1 + sqrt(disc))/(2.*a);
157     x2 = (-b1 - sqrt(disc))/(2.*a);
158   }
159   else if(disc==0)
160     x1 = x2 = -b1/(2.*a);
161   else {
162     //cout << "Pas de solution possible pour x < 0" << endl;
163     return 99999.9;
164   }
165
166   //choisir une solution x < 0, qui ne soit pas la position initiale
167   if (x1<0 && 0.01<fabs(x1-x0))
168     return x1;
169   else if (x2<0 && 0.01<fabs(x2-x0))
170     return x2;
171   else {
172     //cout << "Pas de solution possible pour x < 0" << endl;
```

```
173       return 99999.9;
174     }
175   }
176
177   //definition de la fonction qui calcule l'intersection a droite (x > 0)
178   double solutionDroite(double v, double theta, double alpha, double x0,
          double y0) {
179     double den = v*v*cos(theta)*cos(theta);
180     double a = GN/(2.*den);
181     double b2= -tan(theta) - GN*x0/den + tan(alpha);
182     double c = GN*x0*x0/(2.*den) + tan(theta)*x0 -y0;
183     double x1, x2;
184     double disc = b2*b2-4.*a*c;
185
186     if (disc>0) {
187       x1 = (-b2 + sqrt(disc))/(2.*a);
188       x2 = (-b2 - sqrt(disc))/(2.*a);
189     }
190     else if (disc==0)
191       x1 = x2 = -b2/(2.*a);
192     else {
193       //cout << "Pas de solution possible pour x > 0" << endl;
194       return -99999.9;
195     }
196
197     //choisir une solution x > 0, qui ne soit pas la position initiale
198     if (x1>=0 && fabs(x1-x0)>0.001)
199       return x1;
200     else if (x2>=0 && fabs(x2-x0)>0.001)
201       return x2;
202     else {
203       //cout << "Pas de solution possible pour x>0" << endl;
204       return -99999.9;
205     }
206   }
207
208   //definition de la fonction qui calcule l'angle d'incidence avant le rebond
209   double angleIncidence(double v0, double thetaOut, double alpha, double x0,
          double y0, double x) {
210     double sign;
211     double den = v0*v0*cos(thetaOut)*cos(thetaOut);
212     double derive = -GN*(x-x0)/den + tan(thetaOut);
213
214     if (fabs(x-x0)<0.0001)    //pour eviter une division par zero
215       sign = 1.0;
216     else
217       sign = (x-x0)/fabs(x-x0);
218
219     double reponse = atan2(derive,sign);    //l'angle est entre -Pi et Pi
220
221     return reponse;
222   }
```