Introduction à ROOT

Licence de Physique

[Home|Syntaxe du C++|Fichiers|Classes I|Classes II|Graphisme]

Contents

1 Introduction

ROOT est une bibliothèque de classes disponible gratuitement et facile à installer sur la plupart des machines. Les classes de **ROOT** offrent de très nombreuses possibilités pour gérer une interface graphique, gérer des bases de données énormes et la représentation de ces données. Nous utiliserons essentiellement ces dernières possibilités. **ROOT** permet de travailler soit *en mode compilé* (i.e., comme vous l'avez fait jusqu'à maintenant) soit *en mode interprété* ; cette dernière possibilité peu être parfois pratique mais nous ne l'utiliserons pas.

Dans ce petit chapitre, nous allons nous contenter de donner des exemples particuliers afin d'illustrer quelques modes de représentations dont vous pourriez avoir besoin. La liste des classes et leur description est accessible sur le site de ROOT

1.1 Philosophie générale

- Chaque objet créé d'une classe de **ROOT** a un nom donné sous forme de chaîne de caractères et qui est le premier argument dans le constructeur. Par convention, on prend souvent comme chaîne de caractères le nom de l'objet.
- Il existe souvent plusieurs constructeurs avec plus ou moins d'arguments pour une classe, en fonction de nos besoin.

- A chaque fois qu'on utilise une classe particulière, il est nécessaire (en général) d'inclure le fichier d'en-tête de la classe. Par exemple, pour utiliser la classe **TApplication**, il faudra faire

```
#include <TApplication.h>
```

- Une fenêtre graphique s'appelle un **TCanvas**. Elle peut contenir un ou plusieurs **TPad** qui sont des zones de tracé.

2 Compilation d'un programme utilisant les classes de ROOT

- Pour compiler un programme utilisant les classes de **ROOT**, vous devez utiliser le compilateur qui a servit à compiler **ROOT** (ici c'est g++) et taper la commande

```
g++ `root-config ---cflags` -o name name.C `root-config ---glibs`
```

si votre programme s'appelle *name.C* ; cela génère une exécutable dont le nom est *name* (-o *name*).

Remarque

- vous avez intérêt à écrire un script shell (appelez-le *Comp*) que vous copierez dans votre *$HOME/bin* ; ce script pourrait être simplement

```
g++ `root-config ---cflags` -o $1 $1.C `root-config ---glibs`
```

$1 étant le premier argument passer à *Comp*. N'oublier pas de le rendre executable (chmod a+x Comp).

3 Interface de gestion des fenêtres

La première chose à savoir est qu'il faut une interface pour la gestion des fenêtres. Une classe prend ceci en charge, c'est la classe **TApplication**.

```
#include <TROOT.h>
#include <TApplication.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
...
   MonApp.Run();
   return 0;
}
```

Remarque

- La classe **TROOT** est la classe de base. On doit toujours définir un objet de cette classe avant de pouvoir en utiliser d'autre ; ici, l'objet est *root* ; le constructeur nécessite un nom (*"root"*) et un titre. De même pour l'objet *MonApp*.
- Les 2 autres arguments du constructeur de **TApplication** sont *argc* et *argv* qui permettent de prendre en compte des éventuels arguments au lancement du *main()* (voir leur définition dans le *main()*)
- Seule la méthode **TApplication::Run()** permet de lancer les fenêtres graphiques.
- Noter les 2 includes.

4 Fenêtre graphique

Une fenêtre graphique appartient à la classe TCanvas. On construit un **TCanvas** par

```
TCanvas c1("c1","Titre de la fenêtre",width, heigth);
```

Cette fenêtre a une largeur *width* et une hauteur *height* (en pixel, c'est-à-dire des entiers). On peut aussi spécifier l'endroit (x0,y0) où le **TCanvas** apparaît en utilisant le constructeur

```
TCanvas c1("c1","Titre de la fenêtre",x0,y0,width, heigth);
```

- Parfois, il est utile de définit un système de coordonnées qui nous est propre. Cela se fait par la méthode **TCanvas::Range**(*xmin,ymin,xmax,ymax*) où cette fois *xmin*, ... sont des **double**.
- Si nous ayons 2 **TCanvas** différents *c1* et *c2*. Pour tracer dans le **TCanvas** *c1*, on fera ***c1.cd()*** avant de tracer quelque chose ; de même avant de tracer dans le **TCanvas** *c2*, on fera ***c2.cd()***.
- Enfin, lorsqu'on trace un objet sur un **TCanvas**, il est nécessaire de mettre le **TCanvas** à jour pour voir l'objet effectivement tracé par la méthode **TCanvas::Update()**.

4.1 Exemple

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   MonApp.Run();
   return 0;
}
```

5 Que peut-on tracer ?

Selon que l'on veuille tracer un courbe (ou surface) à partir d'une fonction ou d'un tableau de points, on utilisera des méthodes assez différentes ; une fonction mathématique sera tracé par un objet "fonction" et un tableau de point par un "graphe" ou un "histogramme".

5.1 Une fonction

5.1.1 fonction 1D

Une fonction 1D (*y=f(x)*) est décrite par la classe TF1.

- une fonction sans paramètre

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TF1.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   TF1 f1("f1","sin(x)/x",0.,10.);
   f1.Draw();
   MonApp.Run();
   return 0;
}
```

  - **TF1** est une classe pour les fonction à une dimension.
  - Les 2 derniers arguments sont les valeurs *xmin* et *xmax* de *x*.
  - Noter l'include.

- une fonction avec paramètres

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TF1.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   TF1 f1("f1","[0]*sin([1]*x)",-3.,3.);
   f1.SetParameter(0,4.5);  //parametre 0
   f1.SetParameter(1,3.14); //parametre 1
   f1.SetParName(0,"Amplitude");
   f1.SetParName(1,"Omega");
   f1.Draw();
   MonApp.Run();
   return 0;
}
```

Les paramètres sont représentés entre [ ] ; Il faut leur donner un valeur avec la méthode **TF1::SetParameter()** et il est possible de leur donner un nom avec la méthode **TF1::SetParName()**.

5.1.2 fonction 2D

Une fonction 2D (*z=f(x,y)*) est décrite par la classe TF2.

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TF2.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   TF2 f2("f2","sin(x)*sin(y)/(x*y)",0.,5.,0.,5.);
   f2.Draw("surf4");
   MonApp.Run();
   return 0;
}
```

- Le paramètre "**surf4**" est une option de Draw() ; vous pouvez avoir la liste complète de ces options.

5.2 Un histogramme

La classe TH1 sert à construire des histogrammes à 1D (*X,Poids*) ; la classe TH2, très analogue à la première sert pour les histogrammes de dimension 2 (*X,Y,Poids*) où *Poids* est le contenu du canal (**bin**) de l'histogramme. Il existe plusieurs type d'histogramme selon la nature des données que l'on entre dans celui-ci. Par exemple, un **TH1F** est un histogramme de **float** ; un **TH1D** est un histogramme de **double** (et de même à 2D).

5.2.1 Histogramme 1D

Il y a deux principaux constructeur pour un histogramme 1D :

- canaux de largeur constante

```
TH1F h1("h1","Titre de l'histo",Nbin,Xmin,Xmax);
```

où *Nbin* est un entier donnant le nombre de canaux, *Xmin* (**float** ou **double**) est la borne inférieure du canal 1 et *Xmax* la borne supérieure du canal *Nbin*. En fait un histogramme possède deux canaux supplémentaires pour les "*underflow*" (canal 0 pour les valeurs < *Xmin*) et les "*overflow*" (canal *Nbin+1* pour les valeurs > *Xmax*)

- canaux de largeur variable

```
TH1F h1("h1","Titre de l'histo",Nbin,X);
```

où *X* est un tableau de float ou de double de dimension *Nbin+1* contenant les bornes inférieures de chacun des canaux.

Il y a deux façons de remplir un histogramme :

- **TH1::Fill()**

```
h1.Fill(x,Poids);
```

ajoute au le canal correspondant à la valeur *x*, la valeur *Poids* ; si cette dernière est omise, 1 est ajouté au canal.

- **TH1::SetBinContent()**

```
h1.SetBinContent(n,contents);
```

remplace le contenu du canal *n* par la valeur *contents*.

- Enfin on trouvera les options de TH1::Draw() ou TH2::Draw() ici.

Exemple

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TH1.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   TH1F h1("h1","Mon histo",101,0.,10.);
   double dx=0.1;
   for( double x=0; x<10. ; x+=dx) h1.Fill(x,x*x);
   h1.Draw();
   MonApp.Run();
   return 0;
}
```


5.2.2 Histogramme 2D

Un histogramme 2D possède les mêmes méthodes de base qu'un histogramme à 1D ; il suffit juste de rajouter la 2ème dimension. Par exemple, le constructeur sera

```
TH2F h1("h1","Titre de l'histo",NbinX,Xmin,Xmax,NbinY,Ymin,Ymax);
```

On le remplira de même avec

```
h1.Fill(x,y,Poids);
```

où *Poids* sera mis dans le canal correspondant à (*x,y*).

5.3 Un point

Un TMarker est un point (*x,y*) représenté par un "marker" (voir les différents types de markers). Les coordonnées du point sont des doubles. Le constructeur d'un **TMarker** a 3 arguments, 2 **doubles** pour les coordonnées du point et un **int** pour le type de **TMarker**.

- On peut modifier la taille et la couleur d'un **TMarker** avec les méthodes **TAttMarker::SetMarkerSize()** et **TAttMarker::SetMarkerColor()**.

Exemple

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TMarker.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   c1.Range(-3.5,-2.1,3.5,2.1);
   TMarker *m=new TMarker(0.,0.,8);
   m->SetMarkerColor(2);
   m->SetMarkerSize(0.3);
   m->Draw();
   m=new TMarker(-1.8,1.2,30);
   m->SetMarkerColor(4);
   m->SetMarkerSize(0.8);
   m->Draw();
   MonApp.Run();
   return 0;
}
```

5.4 Une ligne

Une TLine permet de créer une ligne. Le constructeur prend 4 arguments, les coordonnées du point de départ (*x0,y0*) et du point d'arriver (*x1,y1*).

- Il est possible de modifier les attributs d'une **TLine** (couleur, épaisseur et style) grâce aux méthodes **TAttLine::SetLineColor()**, **TAttLine::SetLineWidth()** et **TAttLine::SetLineStyle()**.

Exemple

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TLine.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   c1.Range(-3.5,-3.5,3.5,3.5);
   TLine *L1=new TLine(-3.,0.,3,0.);
   L1->SetLineColor(2);
   L1->Draw();
   TLine *L2=new TLine(-3.,-2.,1.5,1.8);
   L2->SetLineColor(4);
   L2->SetLineWidth(2.);
   L2->SetLineStyle(2);
   L2->Draw();
   MonApp.Run();
   return 0;
}
```

5.5 Une ellipse

Une TEllipse permet de créer une ellipse. Le constructeur prend 3 arguments obligatoires, les coordonnées du centre de l'ellipse et le premier rayon (demi-axe). Les 4 autres arguments sont optionnels et correspondent à l'autre demi-axe, aux angles de départ et de fin si on veut tracer une portion d'ellipse et enfin un angle correspondant à la rotation éventuelle du demi-grand axe.

- Il est possible de modifier les attributs du contour et les attributs du fond de l'ellipse.

Exemple

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TEllipse.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   c1.Range(-3.5,-3.5,3.5,3.5);
   TEllipse *ellipse=new TEllipse(0.,0.,1.,0.5);
   TEllipse *cercle=new TEllipse(-2.,0.,1.);
   ellipse->SetLineColor(2);
   ellipse->Draw();
   cercle->SetLineColor(7);
   cercle->SetLineWidth(5.);
   cercle->SetFillColor(4);
   cercle->Draw();
   MonApp.Run();
   return 0;
```



```
}
```

5.6 Un rectangle

Un TPave permet de créer un rectangle. Le constructeur prend 5 arguments obligatoires, les coordonnées du coin supérieur gauche, celles du coin inférieur droit et l'épaisseur du contour (un **int**). Le dernier argument est une option de tracer (facultatif).

- Il est possible de modifier les attributs du contour et les attributs du fond du rectangle.

Exemple

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TPave.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   c1.Range(-3.5,-3.5,3.5,3.5);
   TPave *rectangle=new TPave(-1.,-1.,1.5,0.5);
   rectangle->SetLineColor(2);
   rectangle->SetFillColor(4);
   rectangle->Draw();
   MonApp.Run();
   return 0;
}
```

5.7 Un texte

Un TText permet d'afficher un texte sur une fenêtre graphique. Le constructeur prend 3 arguments, les coordonnées de la position du texte (2 **doubles**) et le texte lui même sous la forme d'une chaîne de caractères.

Exemple

```
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TText.h>
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   TCanvas c1("c1","Titre de la fenêtre",400, 400);
   c1.Range(-3.5,-3.5,3.5,3.5);
   TText *montexte=new TText(-1.,-1.,"Un petit texte");
   montexte->Draw();
   MonApp.Run();
   return 0;
}
```

6 Gestion de la souris et du clavier

Il est possible de gérer le clavier et la souris en utilisant la classe **TCanvas** de ROOT. Pour cela, nous devons **créer une classe** (dans l'exemple, la classe *Fenetre*) **dérivant de la classe TCanvas**. Examiner attentivement l'exemple ci-dessous.

Exemple

- Ici la classe *Fenetre* est très simple ; elle ne consiste qu'en un **TCanvas** dont le **Range()** est fixé.
- La méthode **TCanvas::ToggleEventsStatus()** permet l'affichage de la position de la souris dans la fenêtre.
- La méthode permettant de récupérer les information concernant le clavier et la souris est **HandleInput()**. Il est utile (mais pas indispensable) d'appeler **TCanvas::HandleInput()** afin d'utiliser les fonctionnalités d'un **TCanvas** dans la méthode **Fenetre::HandleInput()**.
- La méthode **TCanvas::AbsPixeltoXY()** permet de convertir les coordonnées de la souris (pixel) en coordonnées utilisateur définis par **TCanvas::Range()**.
- Noter enfin qu'il est possible de provoquer l'appel à la méthode **HandleInput()** grâce à la commande **gSystem->ProcessEvents();** (il faudra ajouter l'include **TSystem.h**).

```
#include <iostream>
using namespace std;
#include <TROOT.h>
#include <TApplication.h>
#include <TCanvas.h>
//
//      Prototype de la classe Fenetre
//
class Fenetre : public TCanvas
{
 public:
        Fenetre(char *nom,char *titre,int width,int height);
        void HandleInput(EEventType event, Int_t px, Int_t py);
};
//
//      Implementation de la classe Fenetre
//

Fenetre::Fenetre(char *nom,char *titre,int width,int height):TCanvas(nom,titre,width,height)
{
        Range(-3.5,-3.5,3.5,3.5);
        ToggleEventStatus();  //Affichage de la position de la souris en bas de la fenetre
}

void Fenetre::HandleInput(EEventType event, Int_t px, Int_t py)
{
        TCanvas::HandleInput(event,px,py);
        double x,y;
        switch (event)
        {
        //-----------------------------------------------------------------------
        //      La souris : px et py sont les coordonnées pixel de la souris
        //-----------------------------------------------------------------------
```



```
                case kMouseMotion:break;  //si la souris bouge dans la fenetre
                case kButton1Down:
                        AbsPixeltoXY(px,py,x,y);
                        cout<<"clic avec le bouton gauche en "<<x<<" "<<y<<endl;
                        break;
                case kButton2Down:
                        AbsPixeltoXY(px,py,x,y);
                        cout<<"clic avec le bouton du milieu en "<<x<<" "<<y<<endl;
                        break;
                case kButton3Down:
                        AbsPixeltoXY(px,py,x,y);
                        cout<<"clic avec le bouton droit en "<<x<<" "<<y<<endl;
                        break;
        //-----------------------------------------------------------------------
        //      Le clavier : px contient le code (ascii) de la touche
        //-----------------------------------------------------------------------
                case kKeyPress:
                        switch(px)
                        {
                                case 'a': cout<<"vous avez appuyer sur 'a'"<<endl; break;
                                default: cout<<"la touche est : "<<char(px)<<endl;

                         }
                         break;
         }

}
int main(int argc, char **argv)
{
   TROOT root("root","titre");
   TApplication MonApp("MonApp",&argc, argv);
   Fenetre *mywin=new Fenetre("mywin","Titre de la fenêtre",400,400);
   MonApp.Run();
   return 0;
}
```

[Home|Syntaxe du C++|Fichiers|Classes I|Classes II|Graphisme]