Setup root

Η μεταβλητή $ROOTSYS ορίζει τη θέση όπου βρίσκονται τα αρχεία της ROOT.

```
>echo $ROOTSYS
/cern/root
```

Μπορείτε να ορίσετε τις τιμές των μεταβλητών στο αρχείο .myprofile για να εκτελούνται κάθε φορά που μπαίνετε στο σύστημα.

Θα χρειαστείτε να έχετε τουλάχιστον δύο παράθυρα ανοιχτά. Ένα παράθυρο για να τρέχετε τη ROOT και ένα για να τρέχετε Linux εντολές, ή επεξεργαστή κειμένου κλ.

Starting ROOT (5 minutes)

Για να τρέξει πραγματικά η ROOT, πληκτρολογείστε:

```
> root
```

Το παράθυρο το οποίο πληκτρολογείτε αυτήν την εντολή θα γίνει το παράθυρο της ROOT σας. Πρώτα θα δείτε το άσπρο-και-μπλε παράθυρο ROOT να εμφανίζεται στην οθόνη σας. Έπειτα θα εξαφανιστεί και ένα συνοπτικό "Welcome to ROOT" θα γραφτεί στο παράθυρο.
Επίσης μπορείτε να τρέξετε τη ROOT χωρίς το εισαγωγικό γραφικό παράθυρο δίνοντας :

```
> root -l
```

Μπορείτε να πληκτρολογήσετε "?" (ή ".h") για να δείτε μια λίστα από τις εντολές της ROOT (help).
Για να βγείτε από τη ROOT πληκτρολογείτε:

```
>.q
```

Plotting a function

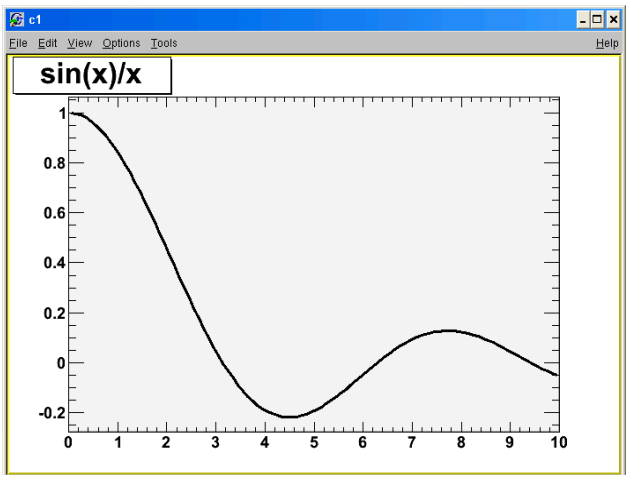
Ας σχεδιάσουμε μια απλή συνάρτηση. Αρχίστε τη ROOT και στη συνέχεια πληκτρολογήστε:

```
[] TF1 f1("func1","sin(x)/x",0,10)
[] f1.Draw()
```

Σημειώστε τη χρήση της σύνταξης C++ για τις εντολές της ROOT.

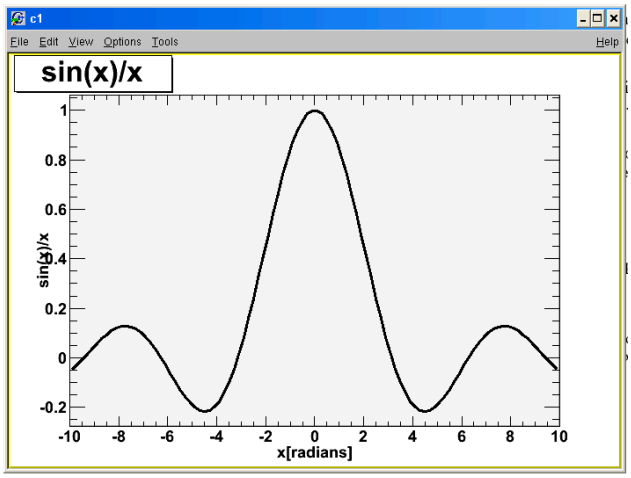
Η πλήρης λειτουργικότητα της ROOT βρίσκεται τεκμηριωμένη στην ιστοσελίδα της ROOT. Πηγαίνετε στην ιστοσελίδα <http://root.cern.ch/> στη συνέχεια πηγαίνετε στο "Reference Guide", και μετά στην "Pro Version…", και μετά στο "TF1".

Με την εκτέλεση της εντολής "f1.Draw()", η ROOT δημιουργεί ένα καμβά (canvas) με όνομα "c1". Ο "canvas" είναι το παράθυρο της ROOT που περιέχει τα γραφικά. Ότι σχεδιάζει η ROOT παρουσιάζονται μέσα σε ένα canvas.



Range

Τοποθετήστε το ποντίκι πάνω στη συνάρτηση (θα μετατραπεί σε δάχτυλο υπόδειξης ή βέλος). Δεξί-κλικ το ποντίκι και επιλέξτε "SetRange". Θέστε την περιοχή σε xmin =-10, xmax=10, και κλικ "OK". Παρατηρήστε πώς αλλάζει η γραφική παράσταση.



Axis Label

Για την εισαγωγή τίτλων στους άξονες κάντε δεξί-κλικ στον x-άξονα της γραφικής παράστασης και επιλέξτε "SetTitle", εισάγετε "το x[radians]", και κλικ "OK". Κεντράρετε τον τίτλο δίνοντας δεξί-κλικ στον x-axis πάλι, επιλέγοντας "CenterTitle", και "OK".

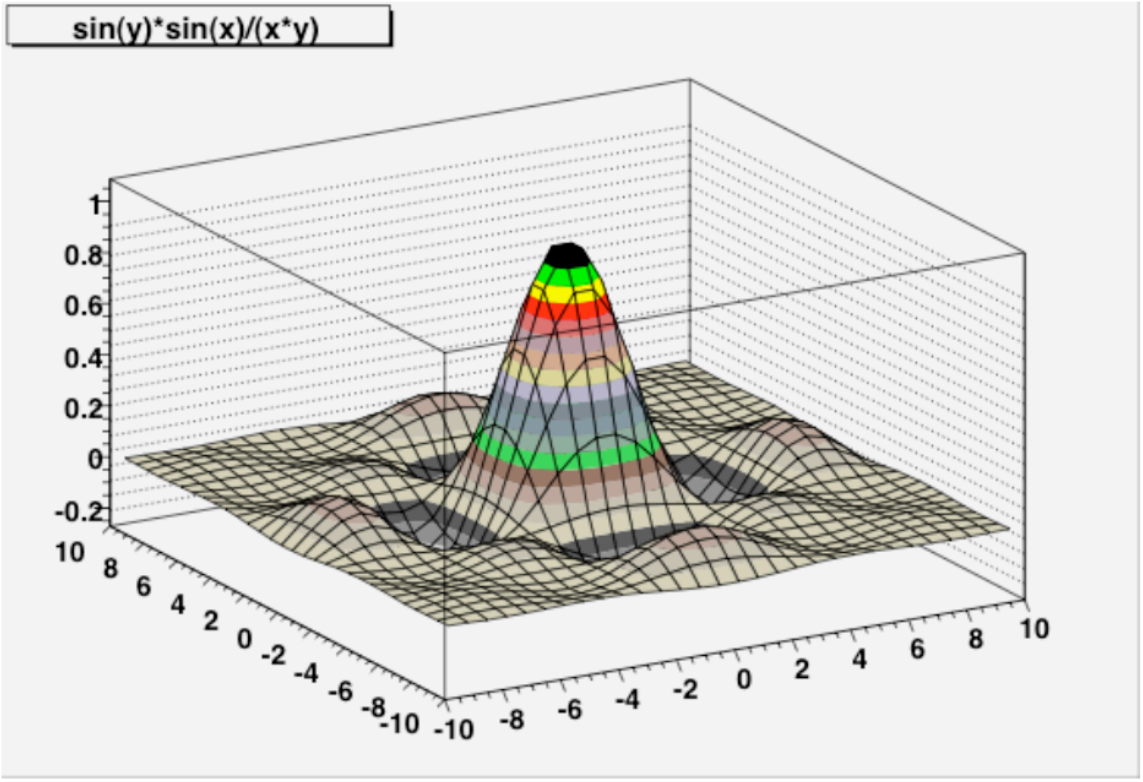
Σημειώστε ότι το δεξί-κλικ στον τίτλο σας δίνει ένα "TCanvas" pop-up μενού και μόνο εάν δώσετε δεξί-κλικ στον άξονα μπορείτε να δράσετε πάνω στον τίτλο. Σε όρους αντικειμενοστραφούς προγραμματισμού ο τίτλος και το κεντράρισμά του είναι μια ιδιοκτησία του άξονα.

Κάνετε το ίδιο πράγμα με τον y-άξονα, με τίτλο "sin(x)/x". Επιλέξτε την ιδιότητα "RotateTitle" των yaxis και δείτε τι συμβαίνει.

Zoom
Μπορείτε να μεγεθύνετε έναν άξονα διαδραστικά (interactively). Αριστερό-κλικ στον αριθμό "2" στον x-άξονα, και σύρσιμο (drag) μέχρι τον αριθμό "4". Η γραφική παράσταση θα επεκταθεί. Μπορείτε να την μεγεθύνετε όσο εείς θέλετε. Όταν έχετε τελειώσει, δεξί-κλικ στον άξονα και επιλέγετε "UnZoom."

Μπορείτε να κάνετε πολλά πράγματα για την αλλαγή της μορφής ενός διαγράμματος. Από την επιλογή “View” του μενού επιλέξτε “Edit”. Μπορείτε να αλλάξετε διάφορες ιδιότητες του γραφήματος. Επίσης μπορείτε να δώσετε View->Toolbar και Edit->Style για να δείτε τις δυνατότητες.
Δεν υπάρχει απλή εντολή Undo αλλά μπορείτε να κάνετε δεξί-κλικ σε κάποιο αντκελιμενο και στη συνέχεια να το διαγράψετε (Delete).

ΑΣΚΗΣΗ
Προσπαθήστε τώρα να κάνετε το παρακάτω γράφημα.



Από την εντολή TF1 της προηγούμενης σελίδας, η οποία παράγει συνάρτηση μιας διάστασης, τι εντολή θα δώσει συνάρτηση δύο διαστάσεων;
Τι ορίσματα θα πρέπει να έχει αυτή η εντολή;
Η μορφή του διαγράμματος είναι τύπου “surf1”, και είναι όρισμα που δέχεται η εντολή Draw(“surf1”).

Working with Histograms (15 minutes)

Δημιουργήστε ένα απλό ιστόγραμμα:

```
[] TH1D h1("hist1","Histogram from a gaussian",100,-3,3)
```

Τα ορίσματα της συνάρτησης:
Το όνομα του ιστογράμματος είναι "hist1".
Ο τίτλος του ιστογράμματος που εμφανίζεται είναι "Histogram from a gaussian".
Ο αριθμός των bins είναι 100 και τα όρια του ιστογράμματος είναι από -3 μέχρι 3.

Ερώτηση: Ποιό είναι το πλάτος του bin αυτού του ιστογράμματος?

```
[] h1.GetBinWidth(0)
```

Σημειώστε ότι πρέπει να προσδιορίσουμε ποιου bin θέλουμε το πλάτος (bin 0 σε αυτήν την περίπτωση), επειδή μπορείτε να καθορίσετε τα ιστόγραμμα με τα διάφορα πλάτη bin.

Δίνοντας

```
[] h1.Draw()
```

δεν θα δείτε πολλά γιατί το ιστόγραμμα είναι άδειο. Ας το γεμίσουμε με τυχαίους 10,000 αριθμούς που ακολουθούν συγκεκριμένη κατανομή:

```
[] h1.FillRandom("gaus",10000)
[] h1.Draw()
```

Η συνάρτηση "gaus" είναι προ-ορισμένη (pre-defined) στη ROOT. Η εξ ορισμού κατανομή Gaus έχει πλάτος 1 και μέση τιμή 0.
Προσέξτε τη στατιστική του ιστογράμματος πάνω δεξιά.

Ερώτηση: Γιατί δεν είναι ο μέσος όρος ακριβώς 0, ή το πλάτος ακριβώς 1;
Προσθέστε άλλα 10,000 γεγονότα στο ιστόγραμμα h1 με τη μέθοδο FillRandom (χρησιμοποιείστε τα βελάκια προς τα πάνω μέχρι να δείτε ξανά το "h1.FillRandom("gaus",10000)"). Κάντε διπλό-κλικ πάνω στον canvas. Ανανενώνεται το ιστόγραμμα ή χρειάζεται να ξαναδώσετε την εντολή "Draw"?

Working with Histograms (continued) (10 minutes)

Ας βάλουμε στο ιστόγραμμα και σφάλματα (error bars). Επιλέξτε "View->Editor", και κάντε κλικ στο ιστόγραμμα. Από το μενού "Error" επιλέξτε "Simple". Προσπαθήστε κάνοντας κλικ στο "Simple Drawing" και δείτε τις αλλαγές στο διάγραμμα.
Το μέγεθος του σφάλματος (error bars) είναι ίσο με την τετραγωνική ρίζα του αριθμού γεγονότων στο αντίστοιχο bin. Με το βέλος-επάνω από το πληκτρολόγιοστο παράθυρο της ROOT δώστε την εντολή FillRandom μερικές φορές ακόμη
Ερώτηση: Γιατί τα σφάλματα γίνονται μικρότερα;

Συχνά θα θελήσετε να κάνετε το ιστόγραμμα με σφάλματα (error bars). Για μελλοντική αναφορά, θα μπορούσατε να χρησιμοποιήσετε την παρακάτω εντολή αντί του Editor:

```
[] h1.Draw("e")
```

Ας δημιουργήσουμε μια δική μας συνάρτηση:

```
[] TF1 myfunc("myfunc","gaus",0,3)
```

Η συνάρτηση "gaus" (ή gaussian) είναι στην πραγματικότητα η

$$P_0\, e^{\left(\frac{(x-P_1)}{P_2}\right)^2}$$

όπου P_0, P_1, and P_2 are είναι "παράμετροι" της συνάρτησης. Ας δώσουμε στις τρεις παραμέτρους δικές μας τιμές, να "ζωγραφίσουμε" το αποτέλεσμα, και να δημιουργήσουμε ένα ιστόγραμμα με τη νέα μας συνάρτηση:

```
[] myfunc.SetParameters(10.,1.0,0.5)
[] myfunc.Draw()
[] TH1D h2("hist2","Histogram from my function",100,-3,3)
[] h2.FillRandom("myfunc",10000)
[] h2.Draw()
```

Σημειώστε ότι θα μπορούσαμε επίσης να θέσετε τις τιμές στις παραμέτρους της συνάρτησης χωριστά για την καθεμμιά:

```
[] myfunc.SetParameter(1,-1.0)
[] h2.FillRandom("myfunc",10000)
```

Working with multiple plots (5 minutes)

Έχουμε τώρα πολλά διαφορετικά ιστογραμματα και συναρτήσεις, αλλά τα σχεδιάζουμε όλα στον ίδιο καμβά, έτσι δεν μπορούμε να δούμε περισσότερα του ενός διαγράμματα κάθε φορά. Υπάρχουν δύο τρόποι για να το αποφύγει κανείς αυτό.
Κατ' αρχάς, μπορούμε να δημιουργήσουμε έναν νέο καμβά με την επιλογή του "New Canvas" από τις επιλογές του υπάρχοντος καμβά μας, αυτό θα δημιουργήσει έναν νέο καμβά με ένα όνομα όπως "c1_n2". Δοκιμάστε αυτό τώρα.
Δεύτερον, μπορούμε να διαιρέσουμε έναν καμβά σε "μαξιλάρια". Στο νέο καμβά, με δεξί κλικ στη μέση και επιλέξτε "Divide". Εισάγετε nx=2, ny=3, και δώστε "OK".
Κάντε κλικ σε διάφορα pads και καμβάδες

```
[] h2.Draw()
[] myfunc.Draw()
```

Saving and printing your work (15 minutes)

"Save->*canvas-name*.C"
```
> less c1.C
```

"Save->c1.pdf"
"Save->c1.root"

```
[] .x c1.C
```

```
> root c1.C
```

Fitting a histogram

Χρησιμοποιείστε το αρχείο "**histogram.root**",
> root –l histogram.root
Ανοίξτε ένα Browser
Double-click στο "histogram.root".
"hist1" and "hist2".
Double-click στο "hist1".
Right-click στο histogram και επιλέξτε "FitPanel".
“Fit Function”, "gaus"
Click on "Fit"
(P_0, P_1, and P_2); "Constant", "Mean", and "Sigma" on the fit output.

click on "landau" on the FitPanel's “Fit Function” pop-up menu and click on "Fit" again; δοκιμάστε "expo" and fit again.

Fitting a histogram (continued)

Ας κάνουμε το ίδιο για το "hist2".

"FitPanel".
"gaus" and then on "Fit". Uggh -- that's a terrible fit!

Ορίστε μια συνάρτηση με την εντολή:
```
[] TF1 func("mydoublegaus","gaus(0)+gaus(3)")
```
P_0, P_1, and P_2 are the "constant","mean", and "sigma" of the first gaussian, and

P_3, P_4, and P_5 are the "constant", "mean",and "sigma" of the second gaussian.

Now try to do a fit by going to the FitPanel, select “User Func” under “Fit Function”, select "mydoublegaus" from the pop-up next to it, then clicking on "Fit".

Αν δεν αποκρίνεται σωστά ο Browser μπορείτε να δώσετε τις εντολές.
```
[] TF1 func("mydoublegaus","gaus(0)+gaus(3)")
```

```
[] hist2.Fit("mydoublegaus")
```

ΑΠΟΤΥΧΙΑ

Και για το απλούστερο Fit, η ROOT χρειάζεται κάποιες αρχικές τιμές για τις παραμέτρους του Fit.

```
[] func.SetParameters(5.,5.,1.,1.,10.,1.)
[] hist2.Fit("mydoublegaus")
```

Το αποτέλεσμα δεν είναι ικανοποιητικό. Δοκιμάστε άλλες αρχικές τιμές μέχρι να το βρείτε.

```
[] func.SetParameters(5.,5.,1.,1.,10.,2.)
[] hist2.Fit("mydoublegaus")
```

Δοκιμάστε ένα καλύτερο set παραμέτρων:

```
[] func.SetParameters(5.,2.,1.,1.,10.,1.)
```

Αν και δεν μοιάζουν πολύ διαφορετικές το αποτέλεσμα είναι σωστό

```
[] hist2.Fit("mydoublegaus")
```

Αν θέλετε να δείτε πως δημιουργήθηκαν τα ιστογράμματα βγείτε από τη ROOT και δώστε:

```
> less CreateHist.C
```

Saving your work, part 2 (15 minutes)

```
[] TFile file1("histogram.root","UPDATE")

[] hist2.Draw()

[] TF1 func("user","gaus(0)+gaus(3)")
[] func.SetParameters(5.,2.,1.,1.,10.,1.)
[] hist2.Fit("user")

[] hist2.Write()
[] func.Write()

[] file1.Close()
```

Quit ROOT, start it again, and use the ROOT browser to open "histogram.root".

Accessing variables in ROOT NTuples/Trees

experiment.root

[] TBrowser b

Με τη βοήθεια του Browser ανοίξτε το αρχείο "experiment.root",
Υπάρχει ένα αντικείμενο το "tree1",
Ένα ROOT Tree (n-tuple) με 100,000 γεγονότα προσομοίωσης.

Δεξί-click στο "tree1" και επιλογή "**Scan**", (και μετά "OK").
Δείτε στο παράθυρο της ROOT, (πολλοί αριθμοί ….)
Στη κορυφή του παραθύρου τα ονόματα των μεταβλητών του δένδρου.

Σε αυτό το υπερβολικά-απλό παράδειγμα, ένα φανταστικό σωματίδιο ταξιδεύει σε μια θετική κατεύθυνση κατά μήκος του άξονα z με την ενέργεια "ebeam". Χτυπά σε έναν στόχο στο z=0, και ταξιδεύει μια απόσταση "zv" προτού να εκτραπεί από την αρχική διεύθυνση από το υλικό του στόχου. Η νέα τροχιά του σωματιδίου δίνεται από τις τελικές ορμές "px", "py", και "pz", και στις κατευθύνσεις x, y, z αντίστοιχα. Η μεταβλητή "chi2" παριστάνει ένα επίπεδο εμπιστοσύνης στη μέτρηση της ορμής του σωματιδίου

Simple analysis using the Draw command

Μπορεί η ανάλυση που θα σας ζητηθεί να μπορεί να γίνει χρησιμοποιώντας την εντολή Draw, το TreeViewer, το FitPanel και άλλες απλές τεχνικές που πραγματεύονται στον οδηγό χρηστών της ROOT. Εντούτοις, είναι πιθανότερο ότι αυτές οι απλές εντολές θα είναι χρήσιμες μόνο για ξεκίνημα, πχ μπορείτε να κάνετε ένα ιστόγραμμα μιας μεταβλητής για να δείτε τα όρια του ιστογράμματος.
Ας αρχίσουμε με απλά τεχνάσματα, και κατόπιν πάμε σε πιο ρεαλιστικές τεχνικές.

```
[] TFile myFile("experiment.root")
[] tree1->Scan()
```

Πως ξέρει η ROOT ότι υπάρχει η μεταβλητή "tree1", χωρίς να την έχετε ορίσει;
Η απάντηση είναι ότι όταν διαβάζετε ένα αρχείο που περιέχει αντικείμενα της ROOT, η ROOT εξετάζει αυτόματα τα αντικείμενα στο αρχείο και δημιουργεί μεταβλητές με το ίδιο όνομα με τα αντικείμενα.

Μπορείτε επίσης να εμφανίσετε το TTree με έναν διαφορετικό τρόπο που δεν παρουσιάζει τα δεδομένα, αλλά εμφανίζει τα ονόματα των μεταβλητών και το μέγεθος του TTree:

```
[] tree1->Print()
```

Μπορείτε να δείτε ότι οι μεταβλητές που είναι αποθηκευμένες στο TTree είναι "event", "ebeam", "px", "py","pz", "zv", και "chi2".

```
[] tree1->Draw("ebeam")
```

Pointers: An all-too-brief explanation (for those who don't know C++ or C)

Προσέξτε ότι στην προηγούμενη σελίδα χρησιμοποιήσαμε τα σύμβολα του δείκτη (pointer) "->" αντί της περιόδου "." για να δώσουμε τις εντολές στο TTree. Αυτό έγινε επειδή η μεταβλητή "tree1" δεν είναι πραγματικά το ίδιο το TTree, είναι ένας "δείκτης" στο TTree. Η διαφορά μεταξύ ενός αντικειμένου και ενός δείκτη στη C++ (και στη ROOT) είναι μια βασική έννοια προγραμματισμού.

```
[] TH1D hist1("h1","a histogram",100,-3,3)
```

Αυτό δημιουργεί ένα νέο ιστόγραμμα στη ROOT (αντικείμενο), και το όνομα του "αντικειμένου ιστογράμματος" είναι "hist1".
Πρέπει να χρησιμοποιήσω μια περίοδο για τις εντολές στο ιστόγραμμα:

```
[] hist1.Draw()
```

Εδώ το ίδιο πράγμα, αλλά με χρησιμοποίηση ενός δείκτη αντ' αυτού:

```
[] TH1D *hist1 = new TH1D("h1","a histogram",100,-3,3)
```

Σημειώστε τη χρήση του αστερίσκου "*" όταν καθορίζω τη μεταβλητή, και τη χρήση της λέξης κλειδί "new". Σε αυτό το παράδειγμα, "hist1" δεν είναι ένα "αντικείμενο ιστόγραμμα" αλλά είναι ένας "δείκτης σε ιστόγραμμα." Πρέπει να χρησιμοποιήσω τα σύμβολα δεικτών στις εντολές:

```
[] hist1->Draw()
```

Πρέπει επίσης να χρησιμοποιήσετε τους δείκτες για να εκμεταλλευθείτε την κληρονομικότητα και τον πολυμορφισμό των αντικειμένων στη C++, και η ROOT στηρίζεται σε μεγάλο ποσοστό στην κληρονομικότητα των αντικειμένων (μερικοί θα έλεγαν πάρα πολύ βαριά).

Για τη διαφορά μεταξύ μιας κλάσης και ενός αντικειμένου μπορούμε να πούμε απλοϊκά ότι μια κλάση καθορίζει μια *αφηρημένη* άποψη μιας έννοιας, ενώ ένα αντικείμενο είναι ένα *συγκεκριμένο* πράγμα.
Υποθέστε ότι καθορίζω την έννοια ενός κύκλου. Αυτό που είναι οι ιδιότητες του κύκλου; Έχει μια ακτίνα, και τη θέση του κέντρου του. Δεδομένου ότι μιλάμε για υπολογιστές, να σκεφτούμε ότι δίνουμε στον κύκλο μερικές εντολές για να εκτελέσει: δώσε μας την περιφέρεια, και το εμβαδόν σου. Οι εντολές (κώδικας C++) είναι οδηγίες που έχουν να κάνουν με τις ιδιότητες του κύκλου (π.χ., $A=4\pi r^2$).
Αν λοιπόν ορίσω μια κλάση για τον κύκλο σε κώδικα C++ και τον σώσω σε ένα αρχείο με όνομα CircleClass.C. Έστω ότι η κλάση λέγεται Circle και περιέχει και μια εντολή για τον υπολογισμό της επιφάνειας: Area.
Οι εντολές που πρέπει να δώσω στη ROOT για να υπολογίσω την επιφάνεια ενός συγκεκριμένου κύκλου c είναι:

```
[] .L CircleClass.C
[] Circle c
[] c.Area()
```

Simple analysis using the Draw command, part 2

```
[] tree1->Draw("ebeam:px")
```
Scatterplot, η εντολή Draw χειρίζεται τις μεταβλητές ως ("x:y") για να αποφασίσει ποιους άξονες να χρησιμοποιήσει.
Να είστε προσεκτικοί, είναι εύκολο να πέσετε στην παγίδα ότι κάθε (x,y) σημείο σε ένα scatterplot αντιπροσωπεύει δύο τιμές στο n-tuple σας. Στην πραγματικότητα, το scatterplot είναι ένα πλέγμα (grid) και κάθε τετράγωνο μέσα το πλέγμα είναι τυχαία εποικημένο με μια πυκνότητα των σημείων η οποία είναι ανάλογη προς τον αριθμό των τιμών σε εκείνο το πλέγμα. (Είναι ιστίγραμμα 2D, όχι πραγματικό scatterplot)

```
[] tree1->Draw("px:py:pz")
```
Ας δημιουργήσουμε ένα "cut" (ένα όριο στη περιοχή που σχεδιάζεται μια μεταβλητή):
```
[] tree1->Draw("zv","zv<20")
```
Συγκρίνετε με το
```
[] tree1->Draw("zv")
```
Σημειώστε ότι μια μεταβλητή σε ένα cut δεν είναι απαραίτητο να είναι μια από τις μεταβλητές που σχεδιάζετε:
```
[] tree1->Draw("ebeam","zv<20")
```
Το σύμβολο για λογικό AND στη C++ είναι "&&". Δοκιμάστε σε ένα cut, πχ.:
```
[] tree1->Draw("ebeam","px>10 && zv<20")
```

Using C++ to analyze a Tree (10 minutes)
Μπορείτε να ξοδέψετε μια ζωή μαθαίνοντας όλα τα του αντικειμενοστραφούς προγραμματισμού σε C++. Ευτυχώς, εσείς χρειάζεται μόνο ένα μικρό υποσύνολο αυτού για να εκτελέσετε την ανάλυση με ROOT. Το πρώτο βήμα είναι η ROOT να γράψει το σκελετό μιας κλάσης ανάλυσης για το ntuple σας.

Αυτό γίνεται με την εντολή MakeClass.
Βγείτε από τη ROOT και αρχίστε την και πάλι.
```
[] TFile myFile("experiment.root")
```

Δημιουργήστε μια macro για το "tree1" με τη MakeClass, και οναμάστε την 'Analyze'.
```
[] tree1->MakeClass("Analyze")
```

Στο παράθυρο του Linux δείτε τα αρχεία που δημιουργήθηκαν:
```
> less Analyze.h
> less Analyze.C
```

Οι περισσότερες αναλύσεις αποτελούνται από τρια βασικά βήματα:

- **Set-up** (open files, define variables, create histograms, etc.).
- **Loop** (για κάθε γεγονός στην n-tuple ή στο Tree, πραγματοποίησε κάποιες ενέργειες: υπολόγισε τιμές, αφάρμασε περιορισμούς (apply cuts) γέμισε ιστογράμματα κλπ).
- **Wrap up** (display results, save histograms, etc.)

Ο C++ κώδικας από το Analyze.C ειναι παρακάτω.

```
#define Analyze_cxx
#include "Analyze.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
void Analyze::Loop()
{
// In a ROOT session, you can do:
// Root > .L Analyze.C
// Root > Analyze t
// Root > t.GetEntry(12); // Fill t data members with entry number 12
// Root > t.Show(); // Show values of entry 12
// Root > t.Show(16); // Read and show values of entry 16
// Root > t.Loop(); // Loop on all entries
//
// This is the loop skeleton where:
// jentry is the global entry number in the chain
// ientry is the entry number in the current Tree
// Note that the argument to GetEntry must be:
// jentry for TChain::GetEntry
// ientry for TTree::GetEntry and TBranch::GetEntry
//
// To read only selected branches, Insert statements like:
// METHOD1:
// fChain->SetBranchStatus("*",0); // disable all branches
// fChain->SetBranchStatus("branchname",1); // activate branchname
// METHOD2: replace line
// fChain->GetEntry(jentry); //read all branches
//by b_branchname->GetEntry(ientry); //read only this branch
if (fChain == 0) return;
// The Set-up code goes here.
Long64_t nentries = fChain->GetEntriesFast();
Long64_t nbytes = 0, nb = 0;
for (Long64_t jentry=0; jentry<nentries;jentry++) {
Long64_t ientry = LoadTree(jentry);
if (ientry < 0) break;
nb = fChain->GetEntry(jentry); nbytes += nb;
// if (Cut(ientry) < 0) continue;
// The Loop code goes here.
}
// The Wrap-up code goes here.
}
```

Running the Analyze macro (10 minutes)

Πως τρέχουμε τον κώδικα.

Quit ROOT, start it again, and enter the following lines:

```
[] .L Analyze.C
[] Analyze a
[] a.Loop()
```

Ρίξτε άλλη μια ματιά στο Analyze.h. Εάν δείτε μέσα σ'αυτό, θα δείτε C ++ εντολές που αναφέρονται ρητά στο όνομα του αρχείου, το όνομα του δέντρου, και τις μεταβλητές του. Τώρα πηγαίνετε πίσω και δείτε στην κορυφή του Analyze.C τη γραμμή
`#include Analyze.h`
Αυτό σημαίνει ότι η ROOT θα περιλαμβάνει τα περιεχόμενα του αρχείου Analyze.h όταν φορτώνει το **`Analyze.C`** .

`.L Analyze.C` λέει στη ROOT να φορτώσει τον κώδικα μέσα στο αρχείο **Analyze.C**, και να "μεταφράσει" τον κώδικα για να δημιουργήσετε μία κλάση C ++. Το όνομα αυτής της κλάσης θα είναι "Analyze" δείτε κοντά στην κορυφή της Analyze.h, και θα δείτε τις λέξεις-κλειδιά **" class Analyze".**

`Analyze a,` δημιουργεί ένα αντικείμενο του οποίου το όνομα είναι "a" .
a.Loop ( ) λέει στη ROOT να εκτελέσει την εντολή Loop του αντικειμένου "α". Δείτε πάλι το Analyze.C. Κοντά στην αρχή, θα δείτε τη γραμμή " void Analyze::Loop ".
Ο κώδικας σε αυτό το αρχείο, καθορίζει την εντολή Loop.