Plotting an Invariant Mass Histogram in R

R

R is a language used for statistics and data analysis. In this tutorial, we are introduce data analysis to R using the CMS Open Data from 2011. At the end of this tutorial, you will know how to create histograms of invariant mass and identify the particles that appear in the histograms.

Data types in R

There are different data types in R. Basic data types include the following:

- Logical
- Numeric
- Integer
- Complex
- Character

Variables are assigned to a type without having to specify it. Numeric variables are simply numbers. Integer variables are specified by adding an “L” at the end of the number:

```
> a = 5L
```

Complex variables are complex numbers, they can be declared using the following format:

```
> b = 5 + 3i
```

```
> d = 8 + 0i
```

Logical variables can have values of either TRUE or FALSE. You can also assign a condition to a logical variable:

```
> c = 3 > 5
```

Character variables are either letters or phrases, you can also add numbers and symbols between quotation marks.

```
> cr = “3!”
```

Vectors

Variables can be either scalar or vector. A scalar vector is a variable that contains a Vectors can contain more than one value, and are created in the following form:

```
> a = c(2, 3, 5)
```

Vectors can be any of the basic data types that we mentioned before. We can also apply conditions to vectors to create a logical vector, using the following conditions

```
> a = c(2, 5, 8, 3, 9)
```

```
> b = a > 3
```

we can create a vector with these values:

```
[1] FALSE TRUE TRUE FALSE TRUE
```

To access a particular element of the vector (vector indexing) we can use the name of the vector and the number of the element in brackets, counting from 1.

The command:

```
> a[1]
```

produces the following output:

```
[1]  2
```

Which is the first element of vector “a”.

We can also do logical indexing, apply a condition so that only the elements of the vector that meet the condition are shown:

```
> a[a>3]

[1] 5 8 9
```

Matrices

In R, we can create a matrix from a vector. Matrices are two-dimensional data structures. The way we create a matrix is by specifying the values it will contain, the number of rows and number of columns and if we are going to fill the matrix by column or by row.

```
> a = c(1:9)
```

Creates a vector with values from 1 to 9.

```
> A = matrix(y, nrow=3, ncol=3, byrow=TRUE)

> print(A)

     [,1] [,2] [,3]

[1,]    1    2    3

[2,]    4    5    6

[3,]    7    8    9
```

To access some element of a matrix we have to specify the row and column values in brackets after the matrix name

```
>  A[2,3]
```

Gives the value 6, which is in the second row and the third column.

We can access the values of a whole column at once, by leaving and empty space where the row number would be, and vice versa. A[2,] gives the values of the second row.

Matrices can also be created using conditions for other matrices, and can also be indexed logical. Indexing a matrix logical gives as a result a vector with the values of the matrix for which the condition is true.

```
> a = c(1:25)
```

Creating a vector with values from 1 to 25

```
> A = matrix(a, nrow=5, ncol=5, byrow=TRUE)
```

Creating a matrix from vector a, with 5 rows and 5 columns, and filling it by row.

```
> C = A > 12
```

Creating a logical matrix from a condition

```
> C
      [,1]  [,2]  [,3]  [,4]  [,5]
[1,] FALSE FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE FALSE
[3,] FALSE FALSE  TRUE  TRUE  TRUE
[4,]  TRUE  TRUE  TRUE  TRUE  TRUE
[5,]  TRUE  TRUE  TRUE  TRUE  TRUE
```

Showing the logical matrix, the values are true when the corresponding elements of matrix A meet the condition A>12.

```
> A[C]
 [1] 16 21 17 22 13 18 23 14 19 24 15 20 25
```

Indexing matrix A using matrix C, this gives of us the values of A for which the corresponding values of C are TRUE. That is, the values for which A > 12, in the form of a vector.

Arrays

Arrays are data structures similar to matrices, but they can have more than two dimensions. You can create them from a vector and specify the number of dimensions of the array.

```
> a = c(1:27)
```

Creating a vector with values from 1 to 27

```
> A = array(a, dim=c(3,3,3))
```

Creating an array from vector a, that has 3 matrices of dimension 3x3.

```
> A
```

Showing the array.

```
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18

, , 3

     [,1] [,2] [,3]
[1,]   19   22   25
[2,]   20   23   26
[3,]   21   24   27
```

Lists

Lists are like vectors, but they can contain different data types at the same time, and they can also contain vectors.

```
> l = list(c(1,2,3),'a', 1, 1+5i)
```

Data Frames

Data frames are like lists of vectors that have the same length. They are used to store data in a tabular form. You can create data frames using the following:

```
> data = data.frame(
+ Name = c("James", "David"),
+ Gender = c('M','M'),
+ Age = c(20, 23))
```

Printing the data frame – print(data) – gives the following:

```
   Name Gender Age
1 James      M  20
2 David      M  23
```

If you want to access a particular column of data of a data frame, you use the name of the data frame followed by a $ and the name of the column. In the previous example, if we want to access the names, we can write:

```
> data$Name
```

Which will return a vector with the names written on it. If we want, on the other hand, access only a particular row, we can use the following:

```
> data[1,]
```

Which will give the first row of the data frame.

You can also import previously created data from other files, like CSV files.

Importing Data from CSV files

You can import CSV files (comma separated values) to R, and analyze the data contained in the files. In this tutorial we are going to use as example the open data published by the CMS Collaboration at CERN. The data files can be found in the following link: http://opendata.cern.ch/record/545, in the CERN open data website. You can download the different CSV files and save them in a folder in your computer.

To import the data files in R we use the following command:

```
> data = read.csv("C:/Data/Jpsimumu.csv")
```

Here we are saving the data contained in the Jpsimumu.csv file in a variable called "data". Be sure to use the right path in your computer, depending on where you have saved your files.

The data files contain thousands of entries, so instead of showing the whole file, we can take a peak using the following function

```
> head(data)
```

This will show the 6 first rows of the data frame.

Calculating the Invariant Mass

We are going to use the Jpsimumu data file. This file contains events from the CMS detector where two muons were detected.

First, read the data file to the variable jpsi,

```
> jpsi = read.csv("Jpsimumu.csv")
```

You can type

```
> head(jpsi)
```

To see the first 6 rows of data in the variable, to make sure everything is OK. You can see that there are values shown for the energy (E), the momentum (p), the pseudo-rapidity eta (η) and the angle phi(φ). The value for the mass is not shown, but we can calculate it from the values of the energy and the momentum.

Invariant mass

In relativistic physics, the invariant mass is the mass of a system equivalent to the rest energy of the system. If we calculate the invariant mass of a system of particles that are the product of the decay of one particle, the invariant mass will be the same as the original mass of the decayed particle.

The invariant mass can be calculated with the following equation:

$$M = \sqrt{(\Sigma E)^2 - \|\Sigma \boldsymbol{p}\|^2}$$

where M is the invariant mass, ΣE is the total energy, and $\Sigma \boldsymbol{p}$ is the total momentum.

To calculate the invariant mass in the code, we are going to use the values of px, py and pz and the values of the energy for the two particles. First, we are going to calculate the vector sum of the momentum of particles 1 and 2. To calculate the vector sum, we have to individually sum each component of the vectors:

```
> pxt = jpsi$px1+jpsi$px2

> pyt = jpsi$py1+jpsi$py2

> pzt = jpsi$pz1+jpsi$pz2
```

Then we calculate the magnitude of the vector, with the following code:

```
> ptotal = sqrt( pxt^2 + pyt^2 + pzt^2 )
```

Here we use the function sqrt( ) to obtain the square root of the sum of all momentum components squared, and save it to the variable ptotal.

We can also define a function to do this calculation. You can define your own functions using the following syntax:

```
myfunctionname = function(arg1, arg2…)

{

statements

return(a)

}
```

To define a function to calculate the magnitude of a vector, we need to take three arguments and return one value. We can call this function magnitude( ) and define it like this

```
> magnitude = function(x, y, z) {

+ m = sqrt(x^2 + y^2 + z^2)

+ return(m) }
```

Now to do the calculation using the function we simply pass the arguments, which are the three components of the momentum in the different spatial directions:

```
> ptotal = magnitude(pxt, pyt, pzt)
```

We can also define a function to do the calculation of the invariant mass of a system consisting of two particles.

```
> invmass = function(px1, px2, py1, py2, pz1, pz2, E1, E2){
+ px = px1+px2
+ py = py1+py2
+ pz = pz1+pz2
+ E=E1+E2
+ ptotal = magnitude(px, py, pz)
+ mass = sqrt(E^2 - ptotal^2)
+ return(mass)
+ }
```

Here we first indicate the name of the function and its arguments, then we obtain the sum of p1 and p2 for each individual component, and we do the sum of the energies, then we obtain the magnitude of the total momentum and finally we calculate the invariant mass using the previously stated equation.

We can now calculate the invariant mass value using the function we have just defined:

```
> jpsimass = invmass(jpsi$px1, jpsi$px2, jpsi$py1, jpsi$py2, jpsi$pz1, jpsi$pz2, jpsi$E1, jpsi$E2)
```

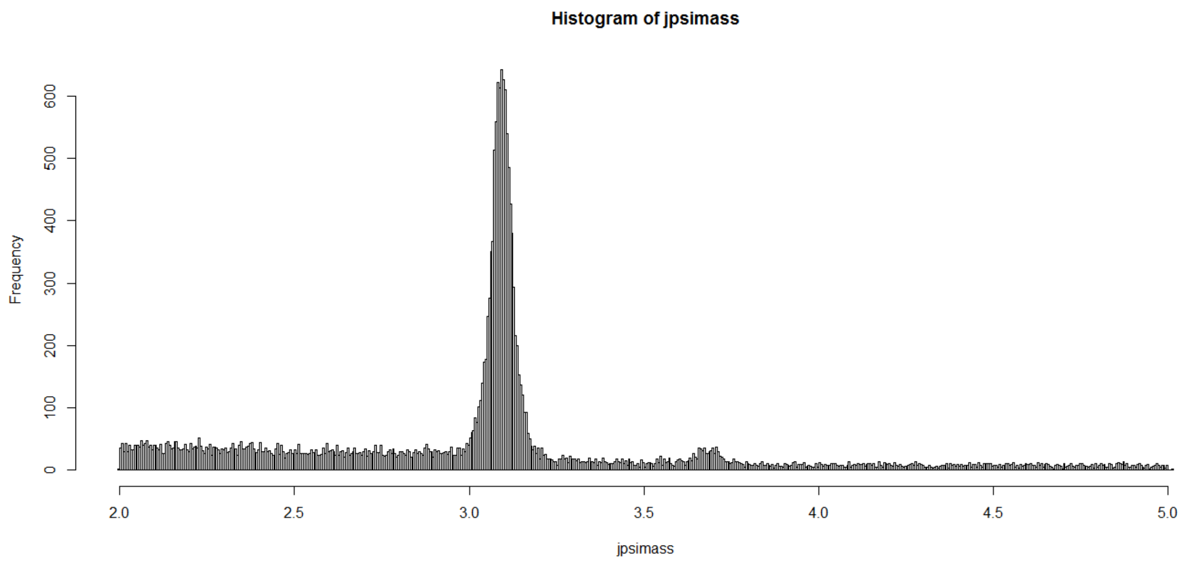
And we can show the first values using the function head(jpsimass).

Plotting a Histogram

Now that we have the values for the invariant mass, we can plot a histogram to see all the data. We can do this in R with the function hist( ).

```
> hist(jpsimass, breaks = 500)
```

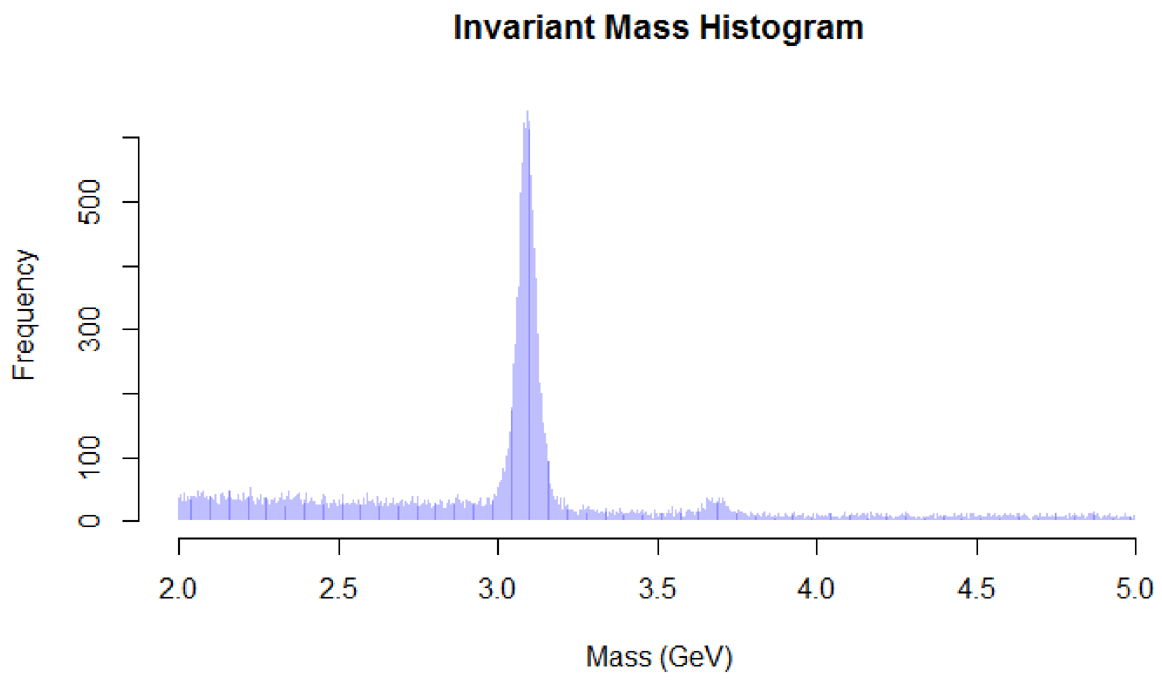
Here, “breaks” indicates the number of bins. You can experiment changing this number to see how you can better visualize the data. The histogram will look more or less like this one:



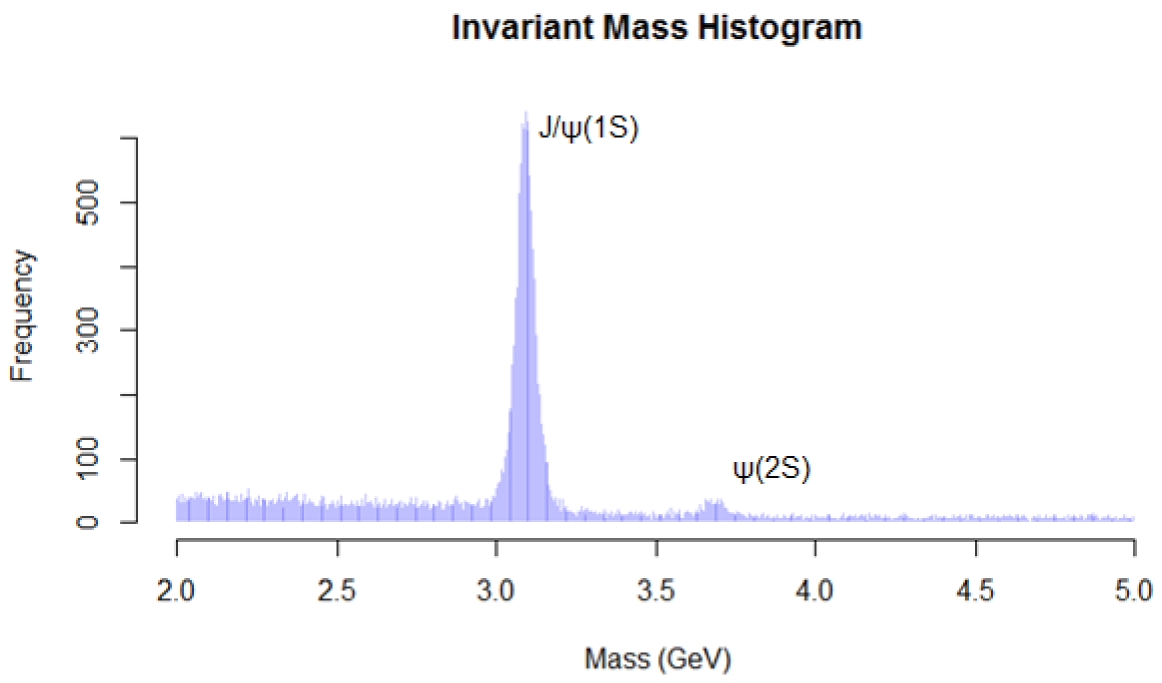
We can change the colors and title of the histogram with different commands. The following command:

hist(jpsimass, breaks = 500, xlab = "Mass (GeV)", lty="blank", col=rgb(0,0,1,1/4), main="Invariant Mass Histogram")

Here 'xlab' is the title for the x axis, lty="blank" just indicates that there's no border for the bins in the histogram, 'main' is the title for the histogram and 'col' indicates the color (the numbers are the values for the red, green, blue and alpha channels). This code will give an output like this:



Now in the histogram you can clearly see a very big peak around 3.1 GeV and a smaller peak around 3.7 GeV. This two peaks correspond to the mass of two particles that have dimuon decay (decay into a muon and an anti-muon). We can search for two such particles in the Particle Data Group database (http://pdg.lbl.gov/), and we find that these two particles are mesons: the J/ψ(1S) meson and the ψ(2S) meson. You can see the two particles in the following graph:



Now you can explore the rest of the files and try to identify the particles that appear in each histogram. To identify the particles, you can take into account the mass and the decay mode and search in the particle data group database.