Almacenamento e análise de datos con ROOT para simulacións de fervenzas de raios cósmicos

Técnicas de Análise e Simulación en Física Nuclear e de Partículas
Curso 2019/2020

21 de Maio de 2020

Lino José Comesaña Cebral

Resumen

Neste documento explícase como implementamos un programa en ROOT para xestionar a saída de datos dun simulador de fervenzas de raios cósmicos (ZHAireS) e como crear arquivos *dinámicos* para transportar gran cantidade de datos consumindo os recursos computacionais mínimos. Ademáis, estudaremos a evolución e comportamento dalgúns parámetros da fervenza como exemplo ilustrativo do potencial das ferramentas de análise implementadas en ROOT.

Índice

1. Simulacións de fervenzas de raios cósmicos empregando ZHAireS

Neste traballo imos tomar como referencia unhas simulacións de fervenzas de raios cósmicos xeradas co software ZHAireS, concretamente unhas simulacións de fervenzas de raios cósmicos iniciadas por neutrinos ν_e de moi altas enerxías na atmosfera. Para entender ben cal vai ser o obxecto do noso análise, imos explicar antes que estamos a simular exactamente e como son os arquivos de saída.

Como xa introducimos, imos supoñer a entrada dun neutrino ν_e de altas enerxías ($\sim 1-10$ EeV) na atmosfera terrestre e, nalgún punto da atmosfera[1] interacciona cun dos núcleos presentes. Concretamente, a interacción que estudaremos vai ser a través de *correntes cargadas*:

$$\nu_e + p \rightarrow e + X$$

onde X é un jet de partículas que se leva ao redor do $\sim 20\,\%$ da enerxía incidente do neutrino.

O noso simulador ZHAireS non é capaz de simular neutrinos, sen embargo, ten unha mecánica de *Special Particles* que nos permite inxectar nalgún punto da atmosfera unha partícula que, á súa vez, é un conxunto de partículas. Así, coa axuda doutro software de simulación (o xerador de eventos Herwig) creamos para distintas enerxías os productos do neutrino ν_e interaccionando por correntes cargadas. Destes productos ou Special Particles seleccionamos só aquelas que teñan unhas enerxías para o primeiro electrón saínte en torno ao $\sim 80\,\%$ da enerxía do neutrino incidente.

Unha vez temos o noso *set* de Special Particles, facemos simulacións con ZHAireS para cada un destes neutrinos chegando á Terra con distintas direccións (isto é, variando os ángulos acimutais e cenitais) e poñendo o vértice da fervenza (o punto da primeira interacción do neutrino) en distintos puntos da atmosfera.

ZHAireS é unha ampliación do software de simulación Aires (AIR-shower Extended Simulations) que emprega o algoritmo ZHS de emisión de radio para calcular os campos eléctricos $s(t)$ en cada instante de tempo nos puntos nos que o usuario defina as antenas. A partires desa información, nós podemos reconstruir

[1] A meirande parte das partículas presentes nos raios cósmicos (protóns, electróns, ...) interaccionan nas primeiras decenas de g·cm^{-2} de atmosfera. Non obstante, como o neutrino ten unha sección eficaz de interacción moi baixa, é dicir, a súa probabilidade de interacción é pequena, non comezará a interaccionar ata que teña atravesado unha cantidade de atmosfera considerable ($\sim$ 300 - 800 g·cm^{-2}).

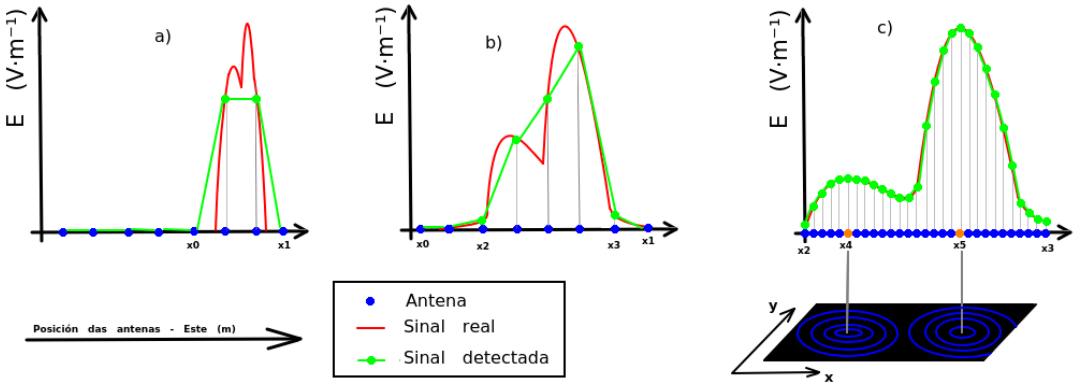


Figura 1: **a)** Localización do sinal: Simulamos cunha resolución auxiliar baixa (poucas antenas repartidas ao longo dunha distancia grande) co obxectivo de detectar onde hai sinal (intervalo [x0,x1]). **b)** Axuste da resolución: Repítese a mesma simulación pero agora coas antenas no intervalo [x0,x1] co obxectivo de almacenar as posicións [x2,x3] correspondentes a antenas rexistrando o 10 % do sinal máximo. **c)** Simulación axustada: Agora que xa temos ben identificado o intervalo de distancias [x2,x3] onde está o sinal, repetimos a simulación pero cunha resolución de antenas boa (aumentamos o número de antenas) e almacenamos as posicións x4 e x5 dos picos (ou pico, de haber un só), que serán os centros nos cales situaremos os arrays de aneis.

o sinal facendo a súa envolvente temporal, a cal se define como o valor absoluto do sinal analítico:

$$u(t) = s(t) + i \cdot H[s(t)] \tag{1}$$

onde $H[s(t)]$ é a transformada de Hilbert. Así, e finalmente, a información coa cal imos traballar son as coordeadas de cada antena e a envolvente de Hilbert do campo eléctrico rexistrado por antena.

Como a duración de cada simulación depende en gran medida nos parámetros impostos (ángulos, punto da primeira interacción, enerxías, número de antenas, ...) empregamos o cluster de nodos do IGFAE. Ademáis, a priori, nós non sabemos cales son as posicións axeitadas das antenas para cada simulación e a súa resolución, polo que empregamos un programa capaz de autoaxustar estas dúas magnitudes e, unha vez acada os valores óptimos, repite as simulacións para arrays de antenas con forma de aneis, como se mostra na Figura 1.

2. Xestionando a saída de datos con ROOT

Dos arquivos que expulsa ZHAireS quedarémonos con aqueles que conteñan información sobre os campos eléctricos en cada instante de tempo t (para facer a envolvente de Hilbert) e as posicións x e y de cada antena. O problema é que, segundo o noso *modus operandi*, temos feitas para cada Special Particle ao redor de ~ 100 simulacións (con ~200 - 300 antenas cada unha), e tendo en conta o tamaño de cada output file a lectura destes arquivos con Python tende a durar ~ 6 minutos (máis o tempo en xerar despois os plots)... En resumo, moito tempo, e se quixésemos facer cambios puntuais nos plots teríamos que reler de novo todo.

Para eliminar este inconveniente imos implementar un programa en ROOT que almacene nun TTree todos estes datos. Así, cada vez que queiramos reler de novo cada simulación xa non teremos que ir a cada output file e consumir minutos do noso tempo, xa que ler as variables directamente dun TTree é un proceso que leva décimas de segundo. Ademáis, esta solución ten outra vantaxe e é o seu tamaño (os arquivos dunha soa simulación ocupa ~ GB e o binario `.root` ocupa só uns poucos KB), o que fai que sexa moi dinámico e portátil.

O programa creado chamámoslle `LECTURA_SIMULACIONS_ROOT.py` (o cal emprega ROOT a través de JupyROOT) e nas seccións seguintes explicaremos as súas funcións.

2.1. Almacén de datos

O punto de partida e principal motivación deste traballo consiste en almacenar os parámetros de cada antena de forma máis *dinámica*, isto é, crear un arquivo lixeiro para transportalo entre distintas CPU e cuxa lectura sexa o máis inmediata posible.

O arquivo `LECTURA_SIMULACIONS_ROOT.py` que imos crear estará formado, en principio, por un TTree[2] organizado en distintas ramas ou TBranches. Cada unha destas ramas serán os parámetros das nosas simulacións: posicións no eixo x, posicións no eixo y, posicións no eixo z e envolventes do campo eléctrico. Estes parámetros obtuvémolos cunha lectura previa en Python e almacenámolos en simples arrays:

```
import ROOT
from array import array
```

[2] Máis adiante imos engadir máis cousas.

```
# Definimos os arrays nos cales almacenaremos os valores de
# todas as antenas que conforman a nosa simulación (todos os
# aneis):
columna_x = []
columna_y = []
columna_z = []
maximos = []

(...LECTURA DOS ARQUIVOS...)
(...ENCHEMOS OS ARRAYS ANTERIORES...)

#????????????????????????????????????????????????????????????

# PARTE CREACIÓN DO TTREE
# Aquí imos almacenar toodos os datos nun TTree para poder
# telos a nosa disposición sen ter que ler de novo toooodas as
#  simulacións (e perder un tempo valioso).

# Defino primeiro as miñas variables auxiliares de almacén:
a = array('d',[0.])
b = array('d',[0.])
c = array('d',[0.])
d = array('d',[0.])

# Creamos o arquivo.root
ARQUIVO = ROOT.TFile("Simulacions_TTree.root","recreate")
# Creamos o TTree:
ARBORE_1 = ROOT.TTree("Almacen_parametros_simulacion","Un Tree simplon")

# Definimos cada rama do noso Tree:
ARBORE_1.Branch("columna_x",a,"a/D")
ARBORE_1.Branch("columna_y",b,"b/D")
ARBORE_1.Branch("columna_z",c,"c/D")
ARBORE_1.Branch("maximos",d,"d/D")

# Agora enchemos todas as ramas coas listas completas que
# queremos:
for i in range(len(maximos)):
    a[0] = columna_x[i]
    b[0] = columna_y[i]
    c[0] = columna_z[i]
```

```
        d[0] = maximos[i]
        ARBORE_1.Fill()

    # Aplicamos os cambios e gardamos:
    ARBORE_1.Write()
    ARQUIVO.Close()
```

Con isto xa temos creado o arquivo `.root` que buscábamos, e se quixésemos facer unha lectura rápida faríamos o seguinte:

```
    # Para volver acceder a este Tree faríamos o seguinte:

    # Primeiro defino as listas onde almacenarei os valores
    # que obteño do TTree:
    maximos = []
    columna_x = []
    columna_y = []
    columna_z = []

    # Abrimos o arquivo.root onde temos o TTree:
    ARQUIVO = ROOT.TFile("Simulacions_TTree.root","update")

    # Xeramos un punteiro ao TTree que buscamos:
    tree_que_busco = ARQUIVO.Get("Almacen_parametros_simulacion")

    # Almacenamos o número total de entradas que ten o TTree, é
    # dicir, o número de filas que conteñe se fixésemos a ins-
    #trucción: tree_que_busco.Scan()

    Numero_de_entradas = tree_que_busco.GetEntries()

    # Agora iteramos en cada TBranch do Tree:
    for i in range(Numero_de_entradas):
        # Metémonos na fila i-ésima:
        entry = tree_que_busco.GetEntry(i)
        maximos.append(tree_que_busco.maximos)
        columna_x.append(tree_que_busco.columna_x)
        columna_y.append(tree_que_busco.columna_y)
    # Pechamos o arquivo:
    ARQUIVO.CLOSE()
```

e xa teríamos de volta os arrays cos parámetros da simulación, e nun instante.

2.2. Analizando as simulacións con ROOT

2.2.1. Creación, almacén e visualización dos plots

Agora que sabemos como ler do noso arquivo `.root` o TTree que queiramos, podemos facer algo similar coas gráficas, é dicir, almacenalas non só como formato de imaxe (`.png`, `.jpg`, `.gif`, `.pdf`, etc) senón que tamén podemos almacenalas con formato `.root`, o cal nos permite modificalas posteriormente de forma interactiva por medio do intérprete de ROOT, sen necesidade de volver a ler os datos e crear de novo as gráficas. A continuación un exemplo da creación do directorio `/plots` e dun exemplo de scatter plot:

```
# Creamos a carpeta /plots no directorio onde temos o arqui-
# vo .root e alí gardaremos en distintas carpetas os plots que
# fagamos:

os.mkdir('plots')
os.chdir('plots')
Ruta_plots = os.getcwd()
os.mkdir('paleta_por_defecto')
os.chdir('paleta_por_defecto')

# Creación de scatter color plots en ROOT

# Por se acaso non tiñamos xa de antes posta a paleta:
ROOT.gStyle.SetPalette(57)

# Defino as dimensións dos canvas que vou crear
altura = 600
anchura = 1500


# Recuperamos os valores que sacamos das simulaciones
x = columna_x
y = columna_y
z = maximos
Number_of_points = len(z)
graph = ROOT.TGraph2D(Number_of_points)
for i in range(Number_of_points): graph.SetPoint(i, y[i], x[i], z[i])
c1 = ROOT.TCanvas("c")
```

```
graph.SetMarkerStyle(20); graph.Draw("PCOL Z")
# A opción "PCOL" fai un scatter plot dos puntos que
# definimos antes. Además, a opción "Z" ao seu carón
# indica que queremos mostrar tamén una barra de co-
# res a modo "lenda"

# Poño eses ángulos para verse mellor
c1.SetTheta(270)
c1.SetPhi(270)
c1.SetWindowSize(anchura + (anchura - c1.GetWw()),
                                altura + (altura - c1.GetWh()))
graph.GetZaxis().SetTitle("Envolvente de Hilbert do
                                campo eléctrico (#muV )")
graph.GetYaxis().SetTitle("Posicion das antenas (m) - Este")
graph.GetXaxis().SetTitle("Posicion das antenas (m) - Norte")
c1.SaveAs("figura_1.png")
c1.SaveAs("figura_1.root")
```

Como o tipo de plot, así como a estética é cuestión de gustos, o código adxunto, a modo de exemplo ilustrativo, garda no directorio `/plots` os distinos tipos de gráficos xerados tanto por ROOT coma por Matplotlib. Poden verse algúns casos nas figuras 2, 3 e 4

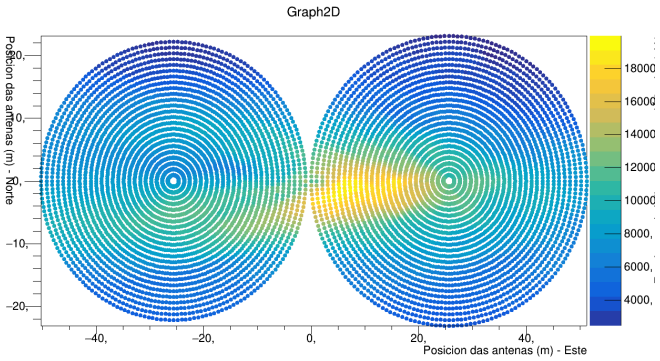


Figura 2: Scatter plot con ROOT

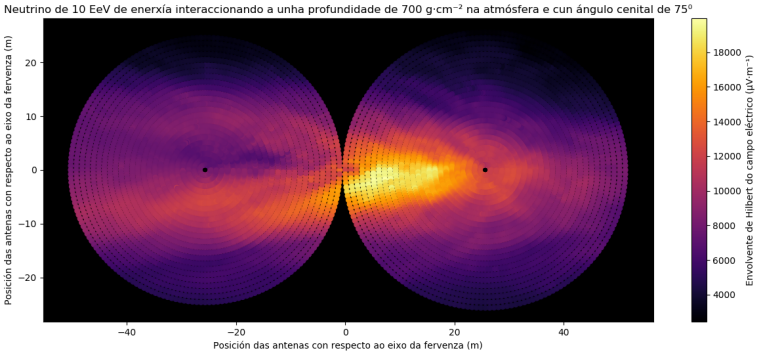


Figura 3: Scatter plot con Matplotlib

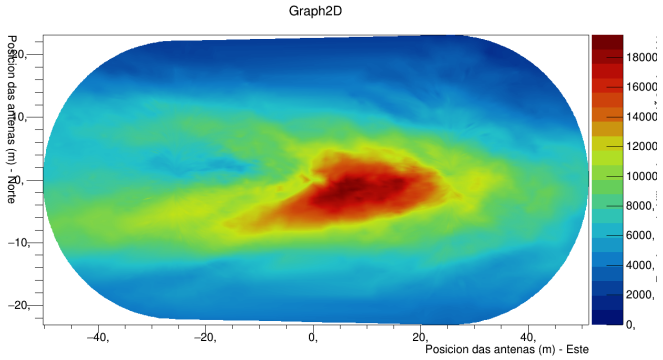


Figura 4: Reconstrucción do sinal con ROOT por triangulación de Delaunay

É interesante a Figura 4, pois a partires do método de triangulación por Delaunay pódese crear unha superficie incluíndo puntos nos cales non temos información. Por suposto, a información non é exacta, pero axúdanos a completar rexións nas cales non esperamos ningún comportamento estraño (como a aparición dun sinal *sorpresa*).

2.2.2. Histogramas

As variables coas que enchemos as ramas do noso TTree foron as coordeadas x, y e z de cada antena e as envolventes de Hilbert do campo eléctrico, tamén para cada antena. Imos engadir no noso código unhas liñas nas cales creamos uns histogramas destas magnitudes:

```
# Abrimos o arquivo.root onde temos o TTree coa opción update:
ARQUIVO = ROOT.TFile("Simulacions_TTree.root","update")
```

```
(...LECTURA DO TTREE...)

# Vou crear a maiores uns histogramas cos cales farei cousas...
histmax = ROOT.TH1F("campos electricos (factor rebinning 28)",
                    "Test data", 350, np.min(maximos), np.max(maximos))
histx = ROOT.TH1F("posicions eixo x (factor rebinning 28)",
                    "Test data", 350, np.min(columna_x), np.max(columna_x))
histy = ROOT.TH1F("posicions eixo y (factor rebinning 28)",
                    "Test data", 350, np.min(columna_y), np.max(columna_y))

for i in range(Numero_de_entradas):
     # Metémonos na fila i-ésima
    entry = tree_que_busco.GetEntry(i)
    histmax.Fill(tree_que_busco.maximos)
    histx.Fill(tree_que_busco.columna_x)
    histy.Fill(tree_que_busco.columna_y)

# Debuxamos os histogramas e os engadimos ao noso arquivo .root
histmax.GetXaxis().SetTitle("Campos electricos")
histmax.Draw()
histmax.Write()

histy.GetXaxis().SetTitle("Posicion das antenas no eixo y")
histy.Draw()
histy.Write()

histx.GetXaxis().SetTitle("Posicions das antenas no eixo x")
histx.Draw()
histx.Write()

ARQUIVO.Close()
```

Os histogramas xerados son os mostrados nas figuras 5, 6 e 7

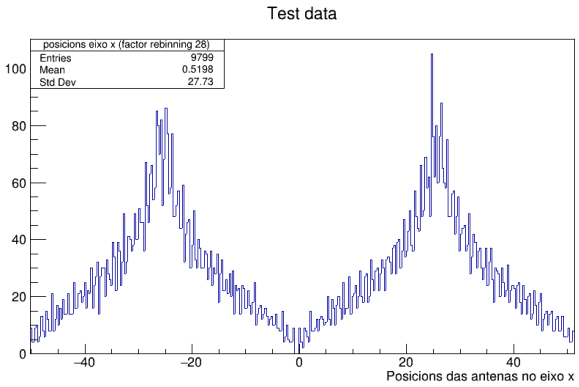


Figura 5: Posición das antenas ao longo do eixo horizontal (Oeste - Este).

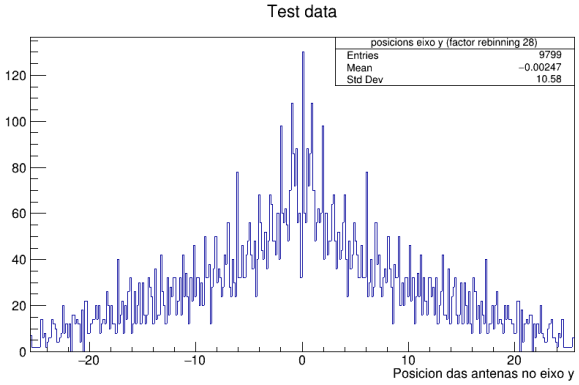


Figura 6: Posición das antenas ao longo do eixo vertical (Sur - Norte).

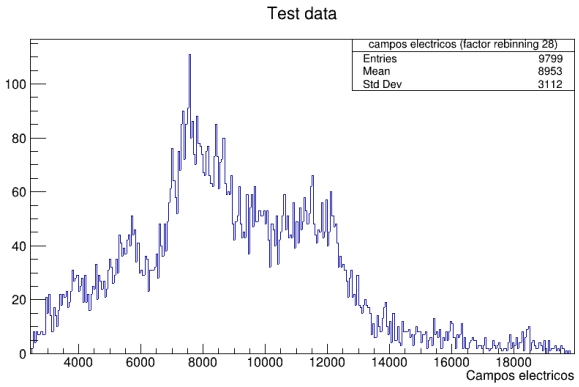


Figura 7: Niveis de activación das antenas.

Destes histogramas sacamos tres estudos interesantes. O primeiro consiste en, a partires do histograma da Figura 7, crear un algoritmo que nos permita **filtrar** o sinal, isto é, que só campos eléctricos con determinada intensidade poidan actuar como trigger das nosas antenas[3]. Outro estudo interesante consiste en averiguar a ***elipticidade*** do noso sinal, que sumado ao ángulo de chegada (obtido a través dos tempos de activación das antenas), pode darnos pistas sobre o vértice da fervenza. Para este último estudo empregaríamos uns axustes nos datos dos histogramas 5 e 6. Finalmente, outro estudo que podemos facer é deducir a **dirección de chegada** do noso neutrino ν_e a partires dos tempos de activación das antenas (recollidos nun dos output files das simulacións).

2.2.3. Creación dun filtro de sinal

Se nos fixamos no histograma da Figura 7, podemos ver un alto nivel de activación de antenas para campos de ~8000 μV·m^{-1}. Se imos a campos máis intensos prodúcese algo similar a un *val* na activación das antenas, cuxo fin está en torno a ~12000 μV·m^{-1}, punto no que comeza a decaer suavemente.

Como sabemos, raios cósmicos menos enerxéticos xeran sinais en radiofrecuencias menos intensas, polo que poderíamos considerar que no caso de ter un background de raios cósmicos menos enerxéticos estaría probablemente no intervalo <12000 μV·m^{-1}, valor que conformaría o noso filtro.

Para ver isto gráficamente, imos crear dentro do mesmo arquivo `Simulacions_TTree.root` outro TTree coas mesmas TBranches pero para aquelas antenas que superen o noso filtro:

```
# Primeiro defino as listas onde almacenarei os valores que obteño
# dunha lectura do TTree orixinal
maximos_filtro = []
columna_x_filtro = []
columna_y_filtro = []
columna_z_filtro = []

# Abrimos o arquivo.root onde temos o TTree:
ARQUIVO = ROOT.TFile("Simulacions_TTree.root","old")

# Xeramos un punteiro ao TTree que buscamos:
tree_que_busco = ARQUIVO.Get("Almacen_parametros_simulacion")
```

[3] Lembremos que estamos a traballar con simulacións baseadas en cálculos teóricos, pero isto sería moi útil na práctica, pois o background de raios cósmicos menos enerxéticos podería suprimirse grazas a esta mecánica.

```
# Almacenamos o número total de entradas que ten o TTree, é
# dicir, o número de filas que contén se fixésemos a instrución:
# tree_que_busco.Scan()
Numero_de_entradas = tree_que_busco.GetEntries()

SINAL_FILTRADO = 12000 # Valor que sacamos vendo o histograma

# Agora iteramos en cada TBranch do Tree:
for i in range(Numero_de_entradas):
    # Metémonos na fila i-ésima
    entry = tree_que_busco.GetEntry(i)
    # Aplicamos o filtro:
    if tree_que_busco.maximos >= SINAL_FILTRADO:
        maximos_filtro.append(tree_que_busco.maximos)
        columna_x_filtro.append(tree_que_busco.columna_x)
        columna_y_filtro.append(tree_que_busco.columna_y)
        columna_z_filtro.append(tree_que_busco.columna_z)
    else:
        continue # Se o sinal non pasa o filtro seguimos

# Xa teríamos o sinal filtrado

ARQUIVO.Close()

# Imos crear outro TTree con estes valores

# Defino primeiro as miñas variables auxiliares de almacén:
a = array('d',[0.])
b = array('d',[0.])
c = array('d',[0.])
d = array('d',[0.])

# Abrimos o arquivo.root onde temos o TTree:
ARQUIVO = ROOT.TFile("Simulacions_TTree.root","UPDATE")
# Poño a opción UPDATE porque vou modificar o arquivo orixinal.
# Creamos o TTree:
ARBORE_2 = ROOT.TTree("Sinal_filtrado","Outro Tree simplon")

# Definimos cada rama do noso Tree:
ARBORE_2.Branch("columna_x",a,"a/D")
```

```
ARBORE_2.Branch("columna_y",b,"b/D")
ARBORE_2.Branch("columna_z",c,"c/D")
ARBORE_2.Branch("maximos",d,"d/D")

# Agora enchemos todas as ramas coas listas completas que
# queremos:
for i in range(len(maximos_filtro)):
    a[0] = columna_x_filtro[i]
    b[0] = columna_y_filtro[i]
    c[0] = columna_z_filtro[i]
    d[0] = maximos_filtro[i]
    ARBORE_2.Fill()

# Aplicamos os cambios e gardamos:
ARBORE_2.Write()
ARQUIVO.Close()
```

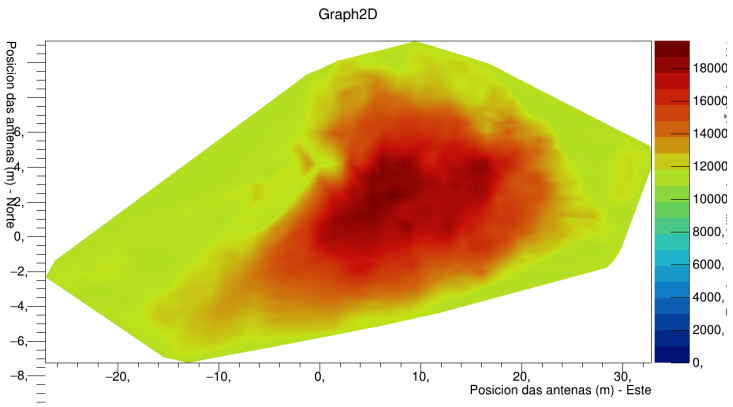


Figura 8: Exemplo do sinal reconstruido a partires da información do filtro.

2.2.4. Estudo da elipticidade do sinal

Os histogramas das figuras 5 e 6 representan a distribución espacial das antenas nas cales *temos datos*. Se lembramos a explicación da sección 1 deste documento (ver Figura 1), estas simulacións seguen unha lóxica implícita, pois os arrays de antenas que temos non foron seleccionados ao azar, senón que houbo un proceso previo cuxo propósito foi o de identificar as rexións onde temos un sinal *nítido* (>10 % do sinal máximo) polo que, se ignoramos as antenas onde se rexistraron sinais máis febles, podemos aproximar o sinal completo á proxección

do cono de Cherenkov, isto é, á sección da fervenza a nivel da terra (no noso caso El Nihuil, Arxentina). En termos xeométricos, isto permítenos coñecer a *elipticidade* do sinal, xa que estamos falando de fervenzas horizontais (canto máis vertical sexa a fervenza menos elíptico será o sinal e máis semellante a un círculo).

Como primeira aproximación, imos facer uns axustes a gaussianas nos datos dos histogramas das figuras 5 e 6 coas funcións da clase `TH1F` de ROOT e gardar os parámetros destes axustes nun arquivo `.txt` externo:

```
(...Recuperamos os histogramas que creamos antes...)

# Customización do histograma nas posicións en Y
c = ROOT.TCanvas("Canvas", "Canvas", 1000, 800)
histy.GetXaxis().SetTitle("Posicion das antenas no eixo y")
histy.Draw()
axuste_y = histy.Fit("gaus","","",np.min(columna_y),np.max(columna_y))

# Engadimos a lenda ao histograma
legend = ROOT.TLegend(0.1,0.7,0.43,0.9)
legend.AddEntry(histy, "Datos", "l")
legend.Draw()

histy.Write()

# Actualizamos o canvas:
c.Draw()
c.Update()
```

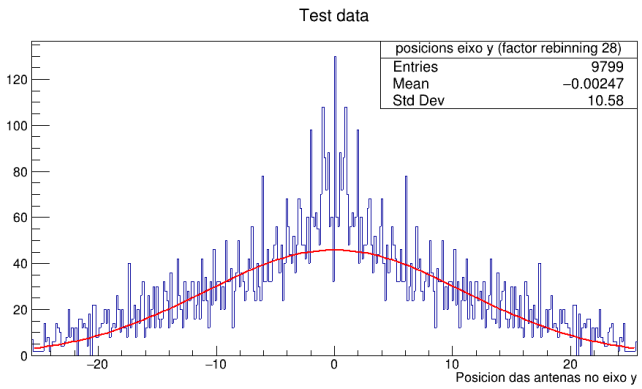


Figura 9: Axuste a unha gaussiana na distribución vertical das antenas

Para a distribución das antenas ao longo do eixo x vemos que podemos facer un fit en distintos rangos, é dicir, facer dous axustes no mesmo histograma para distintos rangos de bins:

```
(...Recuperamos os histogramas que creamos antes...)

# Customización do histograma nas posicións en X
c = ROOT.TCanvas("Canvas", "Canvas", 1000, 800)
histx.GetXaxis().SetTitle("Posicions das antenas no eixo x")
histx.Draw()
# Facemos primeiro o axuste do primeiro pico

g1    =  ROOT.TF1("g1","gaus",np.min(columna_x),0)
axuste_1x = histx.Fit(g1,"R+")

# Engadimos a lenda ao histograma
legend = ROOT.TLegend(0.1,0.7,0.43,0.9)
legend.AddEntry(histx, "Datos", "l")
legend.Draw()

histx.Write()

# Actualizamos o canvas:
c.Draw()
c.Update()

# Rescatamos de novo o histograma e facemos o fit do segundo pico:
histx = ARQUIVO.Get('posicions eixo x (factor rebinning 28)')

# Customización do histograma X
c = ROOT.TCanvas("Canvas", "Canvas", 1000, 800)
histx.GetXaxis().SetTitle("Posicions das antenas no eixo x")
histx.Draw()
# Facemos primeiro o axuste do primeiro pico
g2    =  ROOT.TF1("g2","gaus",0,np.max(columna_x))
axuste_2x = histx.Fit(g2,"R+")

# Engadimos a lenda ao histograma
legend = ROOT.TLegend(0.1,0.7,0.43,0.9)
legend.AddEntry(histx, "Datos", "l")
legend.Draw()
```

```
    histx.Write()

    # Actualizamos o canvas:
    c.Draw()
    c.Update()
```

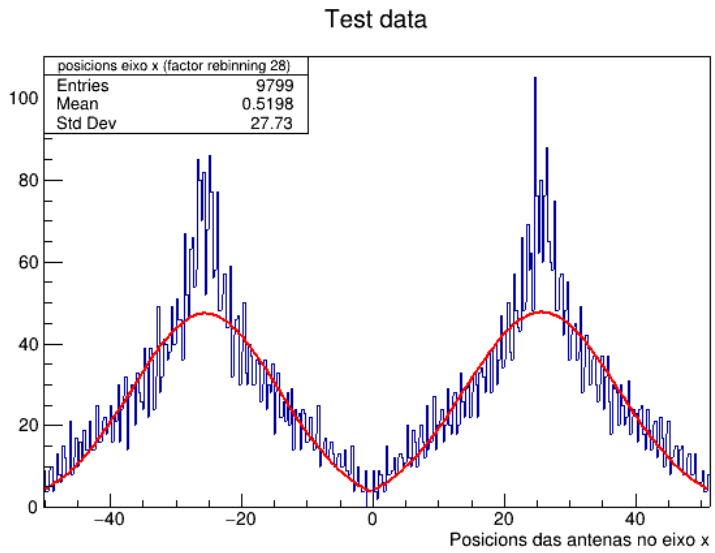


Figura 10: Axuste a unha gaussiana na distribución horizontal das antenas

No mesmo directorio que o arquivo `Simulacions_TTree.root` crease o arquivo `resultados_dos_fits.txt`, no cal temos a seguinte información:

```
############################
    RESULTADOS DOS FITS
############################
Parámetros do axuste da gaussiana da columna y:
45.893317    -0.000164    21.998734

Parámetros dos axustes das gaussianas da columna x:
47.363713    -25.587577    22.533136
47.737100    25.667613    23.366253
```

O primeiro valor de cada axuste correspóndese co valor máis alto da gaussiana, o segundo co seu centroide e o terceiro coa anchura completa a media altura

(FWHM). Interpretando as anchuras destes fits como os eixos dunha elipse podemos facer unha estimación da elipticidade desta fervenza:

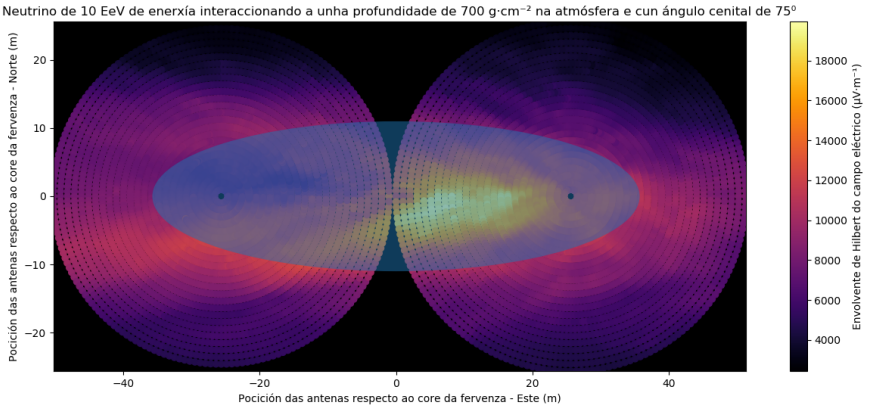


Figura 11: Elipse reconstruida a partires dos axustes ás distribucións espaciais das antenas.

Coa idea de intentar reconstruír o vértice exacto da nosa fervenza precisamos coñecer dous parámetros máis: a dirección da cal procede (verémolo no seguinte apartado) e a altura na cal se produce a primeira interacción. Para este último parámetro, se tivésemos un array dobre de antenas na vertical (é dicir, unha segunda capa a outra altura), estudando a elipticidade do sinal dese outro array e comparándoo co que acabamos de facer permitiríanos ver canto se *agrandou* ou *diminuiu* a elipse e, ao supoñer coñecida a distancia entre ambas superficies de arrays e que o sinal móvese a $v \sim c$, poderíamos facer unha estimación da altura do noso vértice (e incluso do ángulo cenital).

2.2.5. Reconstrucción da dirección de chegada do neutrino ν_e

A parte dos arrays de aneis de antenas tamén temos a simulación inicial, a cal estaba formada por unha liña de antenas horizontal (ver sección 1). Dado que coñecemos os campos eléctricos en cada antena para cada instante de tempo, podemos averiguar a dirección da nosa fervenza no plano ecuatorial terrestre. Lendo os correspondentes output files desta simulación obtemos o seguinte:

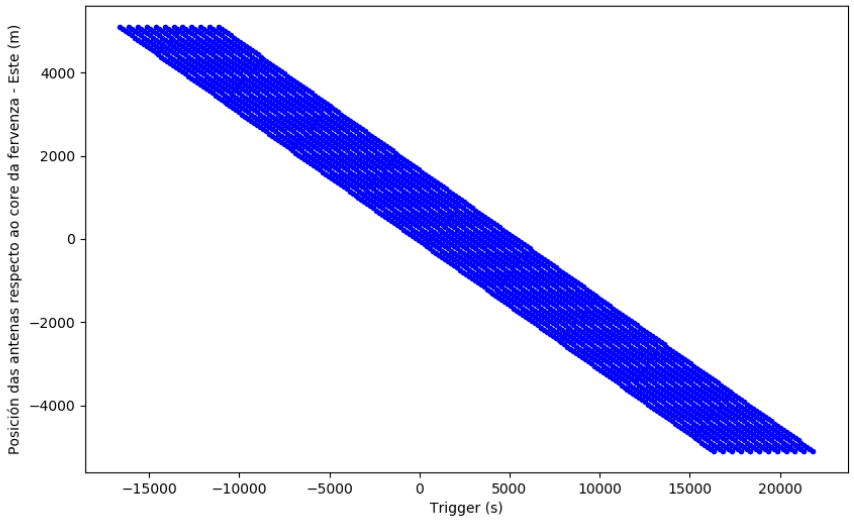


Figura 12: Activación das antenas no plano ecuatorial.

As antenas que se activan primeiro son as que están máis alto no eixo vertical (é dicir, as antenas que están máis ao Este), polo que queda comprobado que a nosa fervenza avanza na dirección Este → Oeste.

Como se pode ver nas imaxes do directorio `/plots` (figuras 2, 3 e 4), ao tratarse dunha fervenza horizontal na *footprint* do sinal intuíase xa como ía ser a súa propagación, pero no caso de fervenzas verticais o sinal perde a elipticidade e está moi concentrado polo que xa non é tan sinxelo, facendo de este método unha ferramenta efectiva.

3. Proxeccións da fervenza nos eixos vertical e horizontal

Imos facer outra implementación interesante de ROOT nas nosas simulacións. Neste caso, co que xa sabemos, imos confeccionar un histograma en dous dimensións (TH2D) do sinal detectado en función das posicións das antenas co obxectivo de, coa axuda dunha macro en C++, facer outros histogramas nunha dimensión (TH1) e mostralos nun Canvas a parte.

```
# PROXECCIONS

antenas = array('f',[])
# Como non podo poñer pares de datos os poño todos seguidos
for i in range(len(columna_x)):
    antenas.append(columna_x[i])
    antenas.append(columna_y[i])
    antenas.append(maximos[i])

factor_rebining = 1
Canvas = ROOT.TCanvas("Canvas", "Canvas", 1000, 800)

histograma = ROOT.TH2D("nombre","",int((np.max(columna_x)-
    np.min(columna_x))/factor_rebining),0,np.abs(
    np.min(columna_x))+np.max(columna_x),int((
    np.max(columna_y)-np.min(columna_y))/
    factor_rebining),0,np.abs(np.min(columna_y))+
    np.max(columna_y))
i = 0

# Recorremos o array que teñe as nosas antenas:
while i < len(antenas):

    # Os histogramas teñen unha cousa que da moitas dores de
    # cabeza, e é a equivalencia bin-valor do array. Nos histogra-
    # mas cada bin ten unhas coordeadas que NON TEÑEN NA-
    # DA QUE VER co plot que estés a facer, simplemente o bin
    # da parte inferior esquerda será sempre o primeiro bin, po-
    # lo tanto non existencoordeadas negativas para un bin (o
    # bin no que temos unha antena nun valornegativo do eixo
    # X será sempre o Bin número i>0. Debido a isto, correre-
    # mos as posicións de todas antenas na vertical e na horizon-
    # tal de modo qie o bin número 1 sexa aquel que teña a coor-
    # deada x máis baixa.
    antenas[i] = antenas[i] + np.abs(np.min(columna_x))
    antenas[i+1] = antenas[i+1] + np.abs(np.min(columna_y))
    # Establecemos o valor en cuestión para ese Bin:
    histograma.SetBinContent(int(antenas[i]),int(antenas[i+1]),
                int(antenas[i+2]))
    # Saltamos á seguinte antena/bin:
    i = i + 3
```

```
histograma.Draw('PCOL Z')
Canvas.Update()
histograma.SetHighlight()
Canvas.Draw()
histograma.SaveAs('histograma_da_simulacion.root')
Canvas.SaveAs('histograma_da_simulacion.png')
```

Agora que temos creado o histograma, creamos unha macro de C++ coa cal poidamos, interactivamente, obter os plots das proxeccións vertical e horizontal do bin sobre o que teñamos situado encima o cursor (facémolo a través da invocación da función `SetHighlight` de TH2). Esta macro chamarémola `proba.C` (adxunto no correo deste documento).

Confeccionada a macro de C++, nunha terminal dirixímonos ao directorio onde o gardamos e nun intérprete de ROOT facemos:

```
$ root
root [0] .x proba.C
```

e aparecerá en pantalla unhas ventanas cos Canvas tanto do histograma 2D coma das proxeccións horizontal e vertical do bin sobre o que teñamos o cursor (Figura 13).

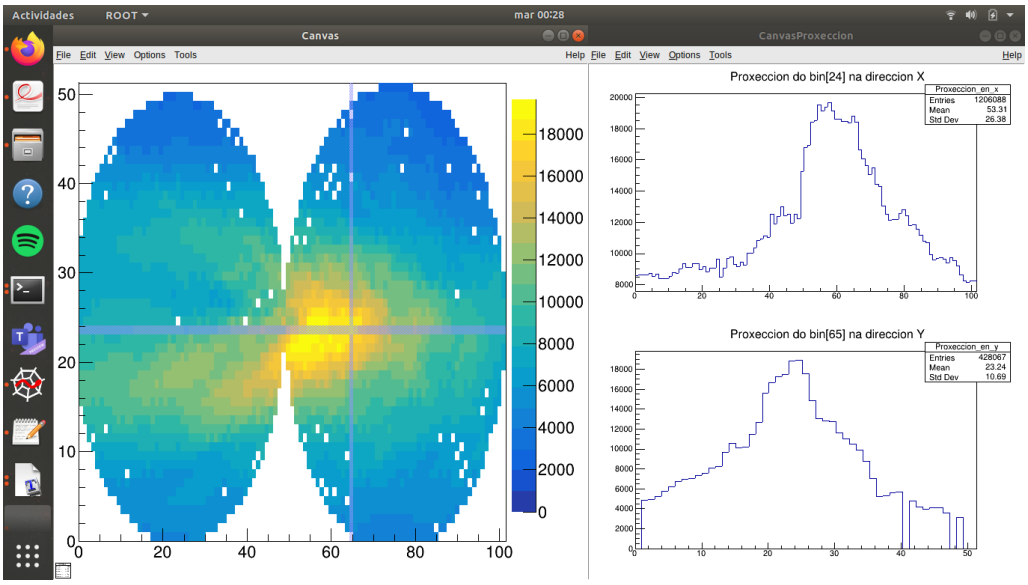


Figura 13: Proxeccións lonxitudinais nas direccións X e Y dun bin sinalado polo noso cursor.

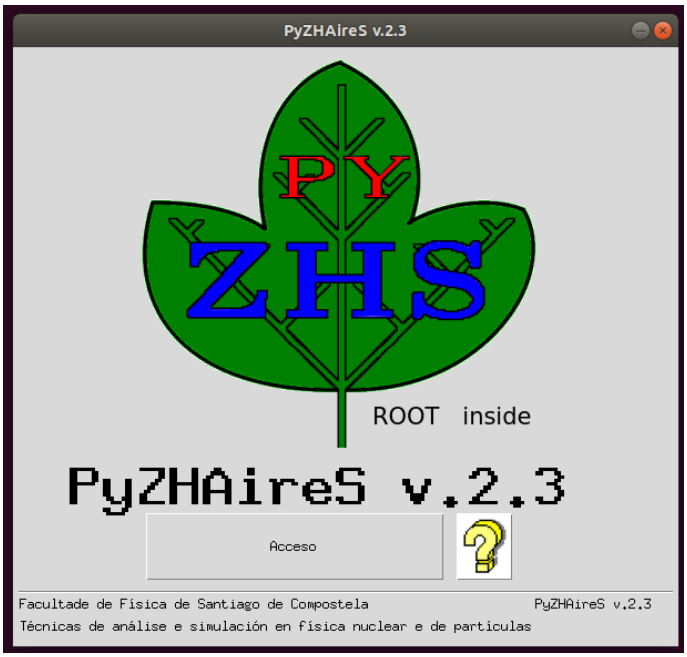
4. Exemplo cunha GUI: ROOZHAireS

Finalmente, só como ferramenta de exemplo para facilitar o seu emprego, implementamos unha aplicación gráfica en Python3 (require ter instalado Tkinter) chamada `ROOZHAireS.py` no cal introducimos todas as funcións do código `LECTURA_SIMULACIONS_ROOT.py` que explicamos na sección anterior.

Para comezar, corremos o código `ROOZHAireS.py` con Python3:

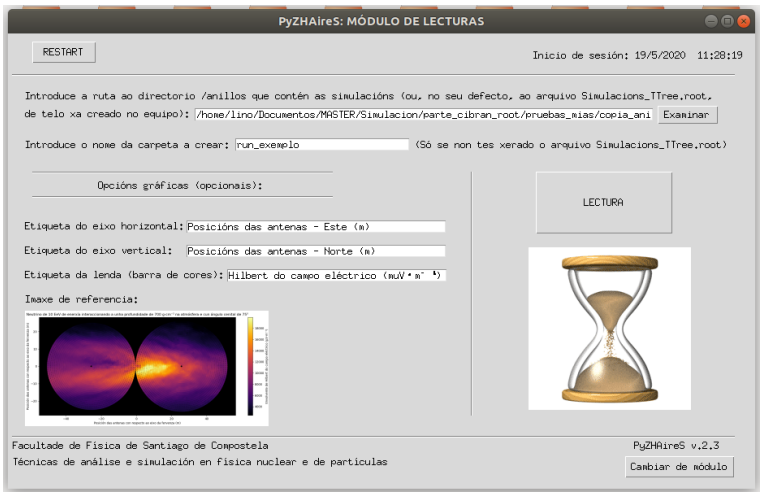
```
$ python3 ROOZHAireS.py
```

e aparecerá en pantalla a interface da aplicación:



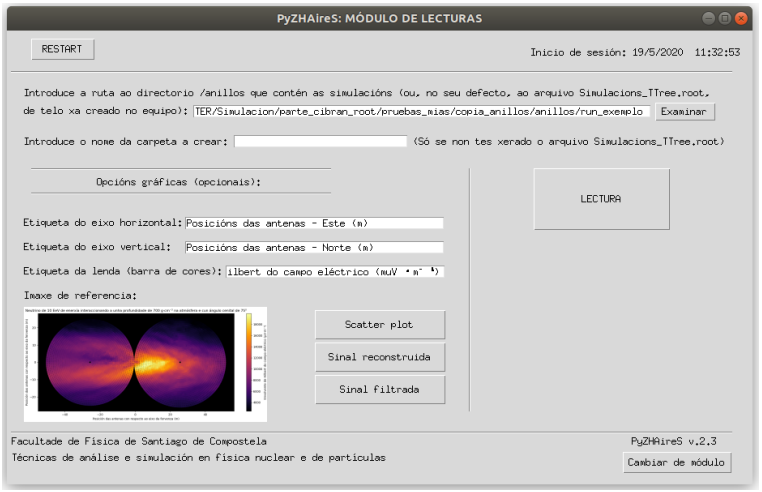
Premendo no botón de *Acceso* entramos ao menú principal da aplicación:

Temos dous módulos de funcionamento[4], pero nós centrarémonos no *Módulo de lectura*. Ao premer no seu botón, aparecerá unha interface na que poderemos poñer o directorio onde temos ubicada a nosa simulación (ou, no seu defecto, o arquivo `Simulacions_TTree.root` se xa o tivésemos creado ou descargado no noso equipo) e, a partires diso, se prememos no botón de *LECTURA*, a aplicación lee o TTree e xera os mesmos arquivos que fai o código `LECTURA_SIMULACIONS_ROOT.py` que explicamos na sección previa:



[4]Esta aplicación está baseada noutra que fixen para automatizar as simulacións no cluster de nodos do IGFAE, por iso aparecen dúas funcións, non obstante, o *Módulo de simulación* non está dispoñible nesta versión para non facer demasiado denso o traballo.

Unha vez remata a lectura teremos creados os directorios e arquivos que xa comentamos e, para facer unha inspección visual rápida dos plots feitos tamén aparecerán na pantalla os seguintes botóns:



5. Conclusións

Neste traballo tomamos como punto de partida as simulacións dunha fervenza de raios cósmicos orixinada por un neutrino ν_e de 10 EeV de enerxía penetrando na atmosfera terrestre baixo un ángulo cenital de 75º e sufrindo a súa primeira interacción a unha profundidade de 700 g·cm^{-2}. Debido ao gran tamaño dos arquivos destas simulacións, empregamos ROOT para almacenar os datos nunha estrutura de TTrees e histogramas, grazas aos cales puidemos facer uns estudos interesantes a maiores para caracterizar esta fervenza.

A implementación de ROOT nas análises das simulacións con ZHAireS permítenos non só crear arquivos lixeiros e dinámicos para almacenar e intercambiar moito contido, senón que tamén abre un dominio moi amplo de posibilidades grazas á facilidade de extracción e tratamento dos datos contidos nun TTree, favorecendo unha lectura dos mesmos nuns intervalos de tempo moi curtos. Deste xeito, o usuario poderá obter os datos de forma eficiente e empregar directamente a gran variedade de ferramentas de análise contidas en ROOT.