Analiza generiranih podatkov

Tadej Tomažič

23. september 2024

Kazalo

Slike

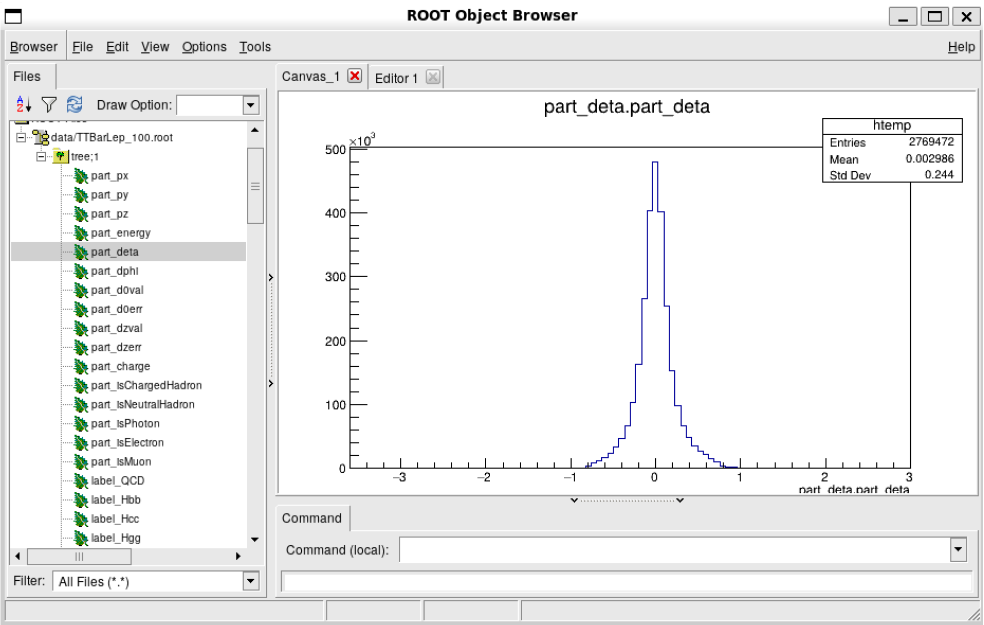
1 Uvod

Set podatkov je bil vzet iz `https://zenodo.org/records/6619768`. Koda katero sem poganjal pa je objavljena na repozitoriju `blabla`. Grafi so bili risani za nek evropski projekt. Zakaj so bili grafi sploh risani, ve samo profesor in Bog.

2 Priprava okolja

Podatki so shranjeni v formatu `.root`. To je format, ki ga je razvil CERN za shranjevanje in analizo velikih količin podatkov v fiziki delcev. Te datoteke so zelo učinkovite pri obdelavi in shranjevanju kompleksnih podatkovnih struktur, kot so histogrami, drevesa in grafi. Za odpiranje datotek `.root` potrebuješ programsko opremo `ROOT`, ki je javno dostopna na portalu CERN-a.

Izkaže se, da za enostavno branje datotek v `pythonu`, ne potrebuješ celotnega modula, ampak le pomožno knjižico `uproot`, ki je veliko manjša od celotnega `ROOT`-a, kar ne pomeni, da ni bil uporaben v mojem primeru. Namreč dani podatki so imeli izredno slabo dokumentacijo in ker sem se sam prvič srečal s takim formatom in knjižicami, mi je prav prišel ukaz `TBrowser` v okolju `root`, ker je skoraj razčistil pomen podatkov.



Slika 1: Grafični prikaz datoteke TTBarLep_100.root

Grafična upodobitev zavaja, saj drevo `tree;1` ima veje, ki jih interpretira kot liste. To se vidi s pomočjo `ipython`-a in knjižico `uproot`, a to ni tako ključnega pomena.

Zagon programov je odvisen od nekaterih knjižnic. Priporočam, da se knjižnice inštalira v virtualnem okolju `venv`, ki ga ustvarimo z ukazom `python3 -m venv ./venv`. Vstop v okolje pa je odvisen od sistema uporabnika, recimo za `linux` lahko uporabimo ukaz `source ./venv/bin/activate`. Sedaj lahko inštaliramo potrebne knjižnice za delovanje programov `pip install numpy matplotlib uproot tqdm scipy`. Za nekaterih knjižnic `pip` ne bo znal sam namestiti, zato jih bo potrebno ročno.

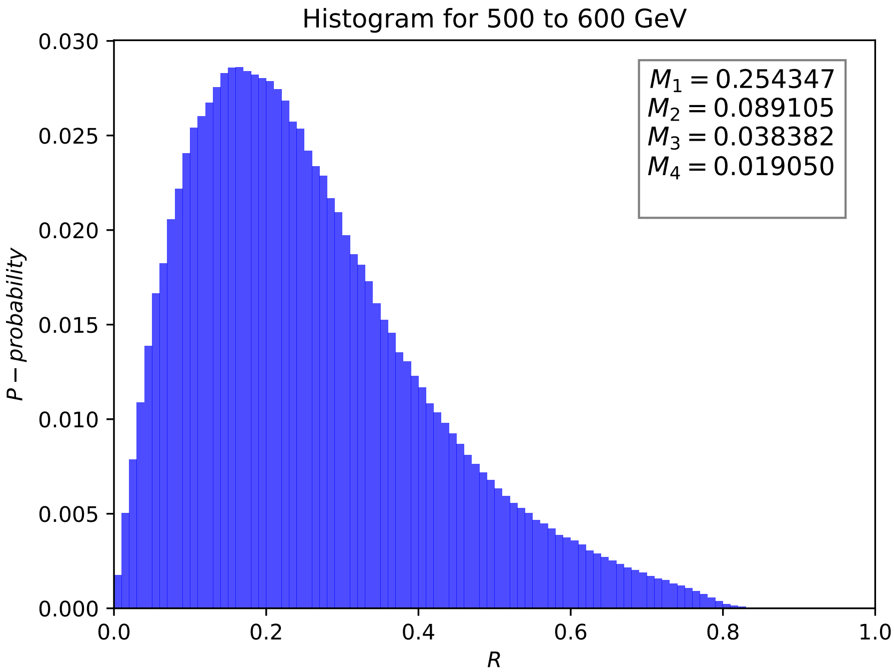
Za branje podatkov sem uporabljal tole kodo. Koda je precej počasna, zaradi osme vrstice,

```
1 tree = uproot.open(file)['tree']
2 jets = {}
3 components = ['px', 'py', 'pz']
4
5 for component in components:
6     part = 'part_'+component
7     branch = tree[part]
8     jets[component] = [jet for jet in branch.array().tolist()]
```

ker prevaja podatke iz `.root` reprezentacije v `numpy` in nato še v navaden seznam. Za pohitritev bi morali uporabljati jezik C++ in knjižnico `ROOT`, ki ne potrebuje prevajanja v treh korakih ampak samo v enem. Kompleksnost in berljivost programske kode se veliko bolj poslabša.

3 Histogrami $P(R)$ v energijskih pasovih

Skupna gibalna količina v podatkih je bila vedno večja od 500 GeV/c in manjša od 1500 GeV/c. Histograme sem najprej risal po energijskih pasovih zato, da sem lahko preveril legitimnost grafov $M(p_T)$. Graf $P(R)$ zgleda takole:



Slika 2: Graf $P(R)$ na energijskem pasu 500 do 600 GeV/c

Za definicijo R potrebujemo definirati rapidnost η in kot ϕ. Za rapidnost jeta označimo η_J in rapidnost posameznega delca v jetu η_i, podobno velja za kot ϕ. Rapidnost izračunamo s pomočjo velikosti celotne gibalne količine p in velikosti komponente p_z po tej formuli:

$$\eta(p,\, p_z) = \frac{1}{2}\log\left(\frac{p + p_z}{p - p_z}\right) \tag{1}$$

Kot ϕ pa je kot med p_x in p_y. V sami kodi sem uporabljal funkcijo `numpy.arctan2()`

$$\phi(p_x,\, p_y) = \arctan\left(\frac{p_x}{p_y}\right) \tag{2}$$

Sedaj lahko končno definiramo R_i, ki predstavlja R posameznega delca v jetu.

$$R_i = \left((\eta_J - \eta_i)^2 + (\phi_J - \phi_i)^2\right)^{\frac{1}{2}} \tag{3}$$

Histogram je bil narisan takole, da je se je vrednost P na intervalu, na katerega je spadal R_i povečala za $\frac{p_{T_i}}{p_{T_J} N_J}$, kjer p_T označuje transverzalno gibalno količino, kjer indeks i predstavlja posamezen delec, indeks J pa celoten jet. in N_J število jetov. Tak pristop nam zagotovi normaliziran graf. Za vsak graf so bili filtrirani jeti, ki niso bili v pravem energijskem pasu.

V grafu so v zgornjem desnem okvirčku izračunani momenti distribuciji. Momenti so definirani takole:

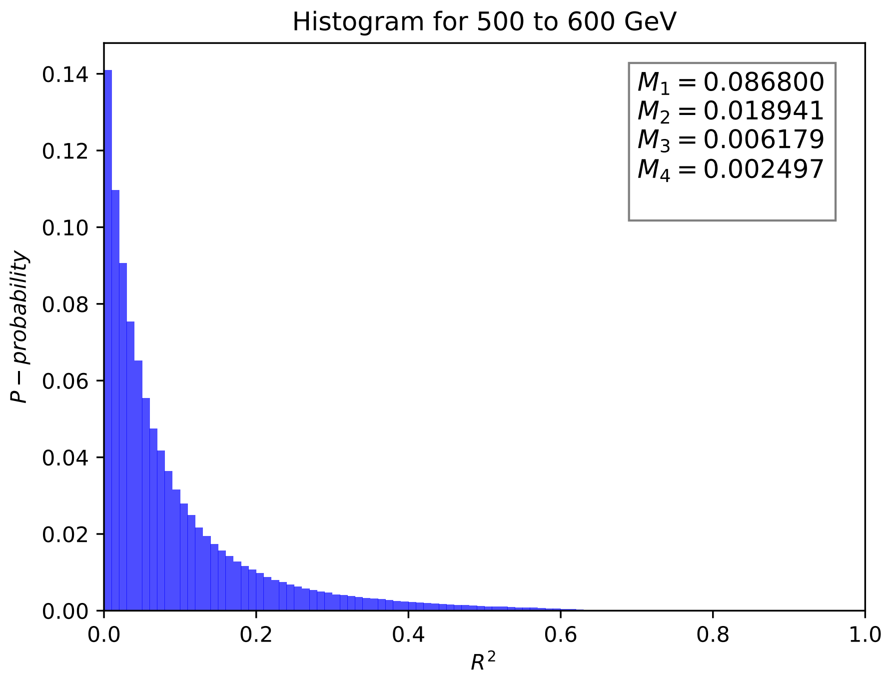
$$M_i = \sum_{k=0}^{N_b} P(R_k) R_k^i \tag{4}$$

Kjer je N_b število intervalov, k pa predstavlja k-ti interval.

Koda tega pristopa je v prilogi pod imenom `R_pt.py` grafi so pa shranjeni v `i.pdf`, kjer je $i \in \{1, 2, 3, 4, 5\}$.

3.1 Histogrami $P(R^2)$

Podobno sem narisal distribucije $P(R^2)$, kjer je namesto R argument R^2. Graf izgleda takole:



Slika 3: Graf $P(R^2)$ na energijskem pasu 500 do 600 GeV/c

Nova definicija se glasi:

$$R_i^2 = (\eta_J - \eta_i)^2 + (\phi_J - \phi_i)^2 \tag{5}$$

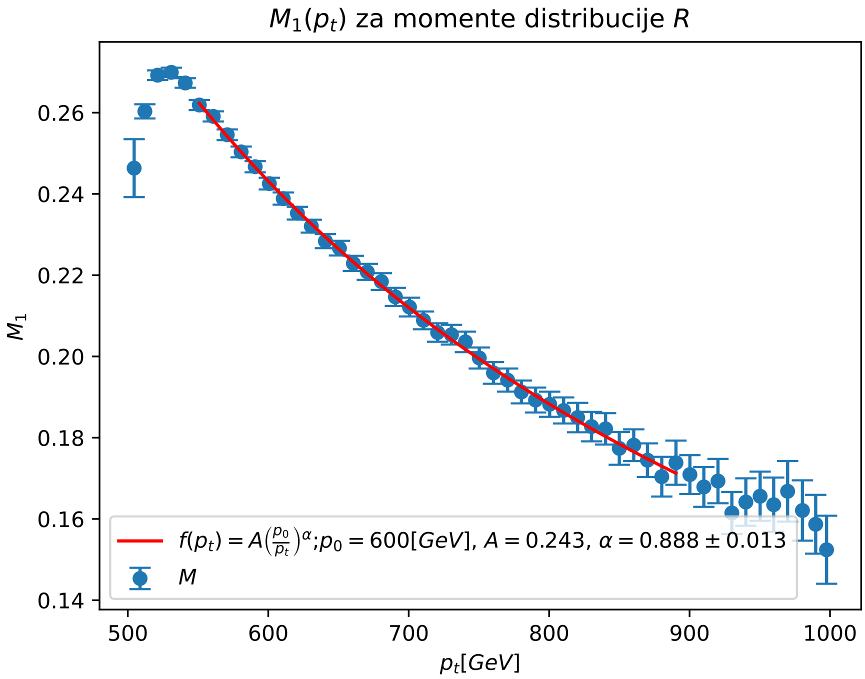
Grafi so shranjeni v prilogi `iR2.pdf`, kjer je ponovno $i \in \{1, 2, 3, 4, 5\}$, koda pa je v `R2_pt.py`

4 Graf momentov $M(p_T)$

Naslednja naloga je bila izrisanje grafov momentov posameznih jetov razporejenih na intervalih transverzalnih gibalnih količin jetov. Znotraj vsakega jeta je bil izračunan moment, in je bil podeljen intervalu. Moment jeta je bil izračunan po formuli:

$$M_{i_J} = \sum_k^{N_J} R_k^i \frac{p_{T_k}}{p_{T_J}} \tag{6}$$

Indeks i predstavlja tip momenta. Znotraj vsakega intervala sem izračunal povprečen moment in ga grafično upodobil z modro piko, kot je prikazano na sliki:



Slika 4: Graf $M(p_T)$ za momente distribucije R, s fitano funkcijo oblike $f(p_T) = A\left(\frac{p_0}{p_T}\right)^\alpha$ od 550 do 900 GeV/c

Grafu sem dorisal absolutno napako, ki je bila izračunana s pomočjo standardne devijacije.

$$\Delta M_j = \sqrt{\frac{\sum_k^{N_j} \left(M_{jk} - \overline{M_j}\right)^2}{N_j^2}} \cdot 1.96 \tag{7}$$

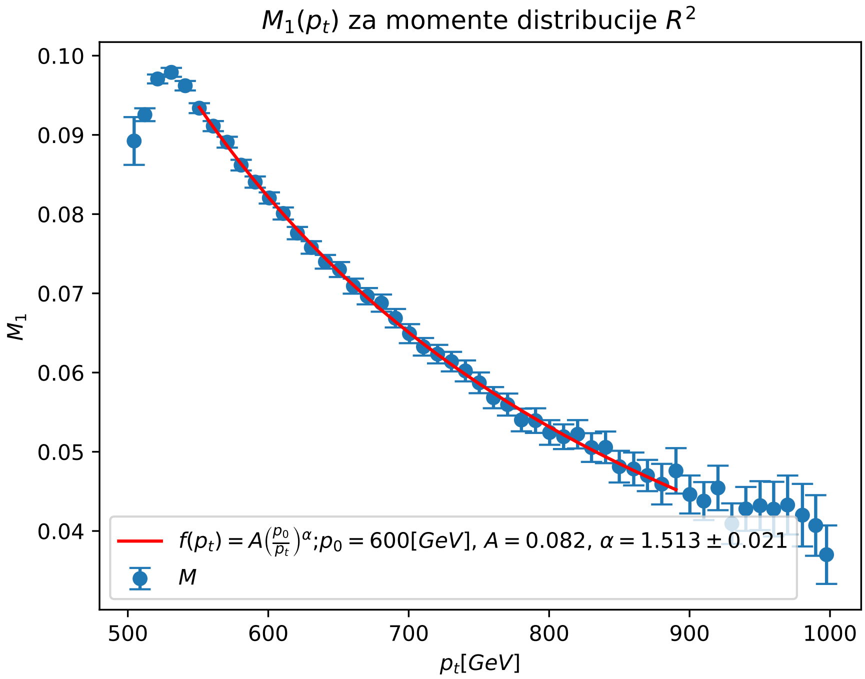
Indeks j označuje zaporedni interval (bin), indeks k pa označuje posamezen moment v intervalu j, $\overline{M_j}$ je povprečna vrednost M znotraj intervala j, ter N_j označuje število momentov na intervalu. Število 1.96 nastopa v enačbi zato, da napaka zajame 95% podatkov.

Fit funckcije je bil narejen s pomočjo `scipy.optimize.curve_fit()`.

Slike so shranjene v datotekah `CCM_i(P_t).pdf`, kjer je i spet število momenta. Sama koda, ki zgenerira tako sliko, je shranjena v `CCM(pT).py`

4.1 Graf momentov $M(p_T)$ za distribucije R^2

Podobno kot smo v pogavlju 3 najprej narisali graf za R in potem R^2 lahko sedaj naredimo isto. Za R^2 vemo formulo iz enačbe (5).

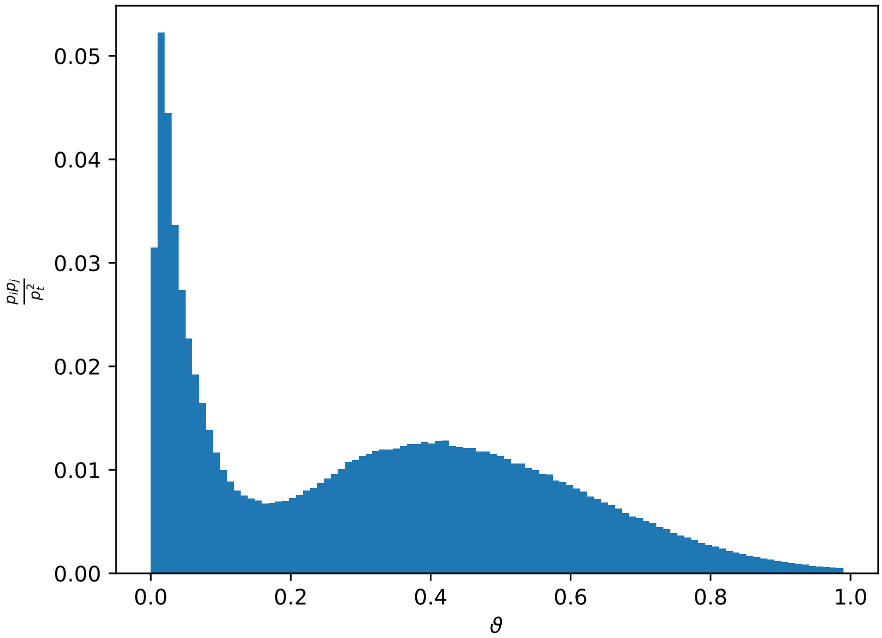


Slika 5: Graf $M(p_T)$ za momente distribucije R^2, s fitano funkcijo oblike $f(p_T) = A\left(\frac{p_0}{p_T}\right)^\alpha$ od 550 do 900 GeV/c

Grafi za momente distribucije R^2 so shranjeni pod imenom `M_i(P_t).pdf`, koda pa v `M(Pt).py`

5 EFP graf

Predzadnji graf, ki sem ga risal je bil EFP graf.



Slika 6: Graf EFP

Kot je vidno iz slike, imamo dimenzijo ϑ in $\frac{p_i p_j}{p_T^2}$. Spremenljivka p_T predstavlja transverzalno gibalno količino danega jeta, p_i pa transverzalno gibalno količino delca v danem jetu.

Poglejmo si, kako se izračuna ϑ.

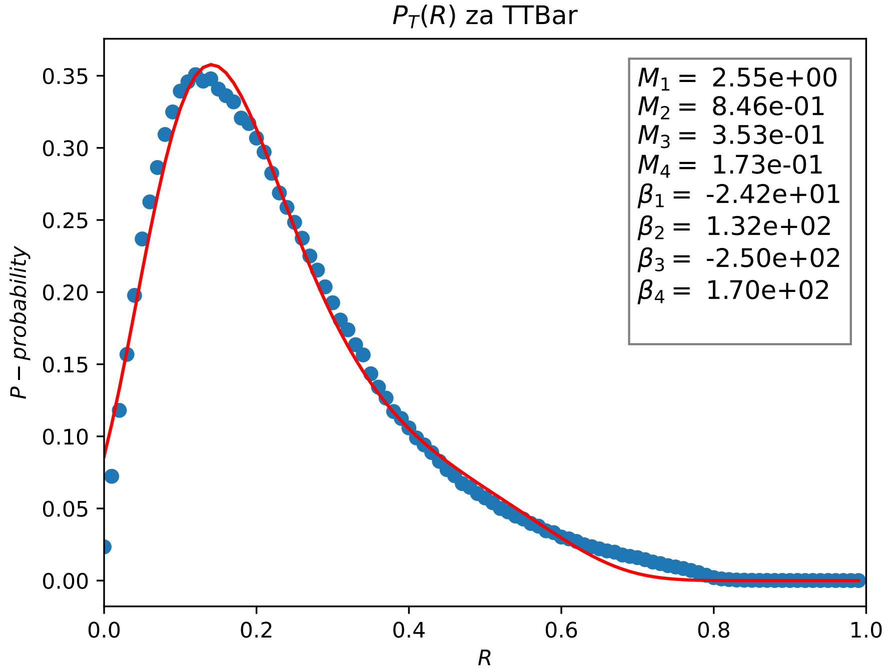
$$\vartheta_{ij}^2 = (\eta_i - \eta_j)^2 + (\phi_i - \phi_j)^2 \tag{8}$$

Spremenljivki ϑ sem dopisal indekse, da je bolj očitno, iz katerih dveh delcev i in j je izračunana. Podobno kot prej je treba graf normalizirat s številom jetov, ki je bil risan. Pomembno je vedeti, da delca i in j pripadata istemu jetu.

Koda, ki izriše graf je v `MAIN/main.py`, funkcije in grafi so pa iz moje strani poimenovani `ERPG`. Zato tudi v kodi obstaja funkcija `ERPG_2()`.

6 Grafi $P(R)$ za vse energijske pasove

Kot samo ime sugestira, so grafi narejeni na podoben način kot v poglavju 2, s to razliko, da ne ločujemo grafov po energiskih pasovih.

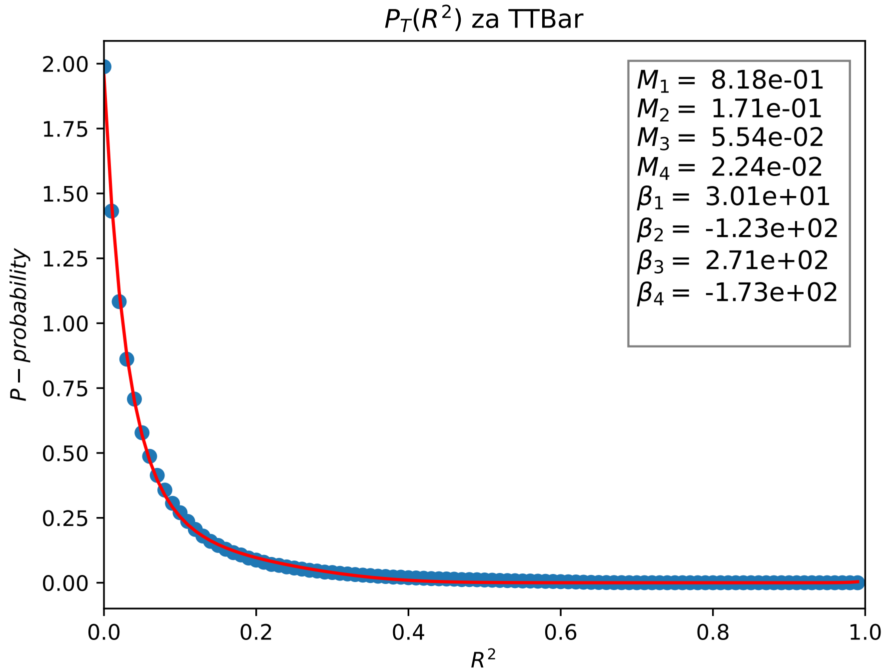


Slika 7: Graf $P(R)$, s fitano funkcijo oblike $A \cdot \prod_i e^{\beta_i}$

Na graf sem še fital funkcijo oblike:

$$f(x) = A \cdot \prod_i e^{\beta_i} \tag{9}$$

Fitani so bili parametri A in β_i. V levem zgornjem okvirčku so še izračunani momenti. Podobno sem risal grafe funkcije R^2. Koda je shranjena v `MAIN/functions.py`, grafi pa v `MAIN/pdfs`. Vsak graf pripada različni datoteki in zato so shranjeni na način, ki sam po sebi pove, kaj predstavlja. Podatki so bili vzeti iz vseh oblik istih datotek. Kot primer: `TTBarLep_100.root TTBarLep_101.root ...`



Slika 8: Graf $P(R^2)$, s fitano funkcijo oblike $A \cdot \prod_i e^{\beta_i}$