FAKULTÄT FÜR PHYSIK
Institut für Experimentelle Teilchenphysik

Virtualisierung von Rechnerplätzen mit JupyterHub

(Virtualization of computer workstations with Jupyterhub)

ETP-Bachelor-KA/2020-04

Bachelorarbeit
von

Jonas Eppelt

am Institut für Experimentelle Teilchenphysik

Referent:	Prof. Dr. Günter Quast
Korreferent:	Dr. Manuel Giffels

Bearbeitungszeit: 01.02.2020 – 08.05.2020

Erklärung zur Selbstständigkeit

Ich versichere, dass ich diese Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der gültigen Fassung vom 24.05.2018 beachtet habe.

Karlsruhe, den 14.05.2020, _____

Jonas Eppelt

Als Prüfungsexemplar genehmigt von

Karlsruhe, den 14.05.2020, _____

Prof. Dr. Günter Quast

Inhaltsverzeichnis

1. Einleitung

Seit ihrem Erscheinen sind *Jupyter Notebooks* immer mehr zu einem elementaren Werkzeug in vielen Bereichen der Datenanalyse geworden [KRKP+16]. Mit ihrer Kombination von Programmcode, dessen Ausgaben und Text sorgen sie für eine leicht nachvollziehbare Darstellung von Datenanalysen und -auswertungen. Die im Notebook erarbeiteten Schritte lassen sich leicht reproduzieren oder mit vergleichbaren Datensätzen validieren. Durch ihre große Verbreitung sind sie bereits zu einem Quasi-Standard im Bereich der Datenanalyse geworden.

Die zu Grunde liegende Technologie ermöglicht breite Einsatzmöglickeiten und eine großflächige Bereitstellung an viele Nutzer. So hat z. B. ein Zusammenschluss verschiedener kanadischer Organisationen im Bereich der Digitalisierung ein Cloud-basiertes System für Notebooks aufgebaut. Es wird von sechzehn Instituten, beziehungsweise 11.000 Personen genutzt [LABZ19].

Auch in der Lehre erfreuen sich Notebooks großer Beliebtheit. So wird z. B. an der University of California at Berkeley ein Kurs zur Datenwissenschaft vollständig mit *Jupyter* veranstaltet, der 2018 von rund 1400 Studierenden besucht wurde [LABZ19].

Aber auch in Großforschungsprojekten kommen sie inzwischen zum Einsatz. Ein bemerkenswertes Beispiel dafür ist ihre Verwendung bei der Entdeckung der Gravitationswellen am LIGO (Laser Interferometer Gravitational-Wave Observatory). Die Daten und die konkreten Analyseschritte wurden der Öffentlichkeit mit einem Notebook vorgestellt und zugänglich gemacht [AAA+16] [KRKP+16].

In dieser Arbeit wird demonstriert, wie Notebooks im Rahmen des Physikstudiums am Karlsruher Institut für Technologie verwendet werden können und welche Vorteile sie für Lehrende und Lernende bringen. Dazu wird zum einen mit *EnJoY* eine Infrastruktur vorgestellt, die Notebooks mit virtuellen Umgebungen und Cloud-Computing verbindet, um:

- die Reproduzierbarkeit der Berechnungen und Analysen zu erleichtern,
- die Nutzung von Notebooks von der Hardware der Studierenden unabhängig zu machen
- und das Angebot an Notebooks mit entsprechender Software leichter auf die erforderliche Größe zu skalieren.

Zum anderen wird anhand der Übung zur Bestimmung der Top-Quark-Masse der Teilchenphysik I Vorlesung gezeigt, dass selbst komplexere und umfangreiche Übungen sich gut und einfach als Notebook realisieren lassen.

Abschließend wird ein Ausblick gegeben, wie sie in weiteren Bereichen des Physikstudiums verwendet werden können und wie *EnJoY* weiterentwickelt werden kann. Insbesondere wird der Austausch von Notebooks zwischen Lehrenden und Studierenden betrachtet und

mit dem *nbgrader* eine mögliche Erweiterung von *EnJoY* vorgestellt, die speziell für die Lehre entwickelt wurde. Auch wird ein kurzer Blick darauf geworfen, wie Notebooks im Physikstudium abseits von Übungen eingesetzt werden können.

2. Project Jupyter

2.1. Überblick

> "Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages."[Jupc]

Project Jupyter [Jupc] stellt Entwicklungsumgebungen, Dateiformate und weitere Standards für interaktives Arbeiten mit einer ganzen Reihe an Programmiersprachen bereit. Geprägt wird das *Project Jupyter* dabei von folgenden Merkmalen:

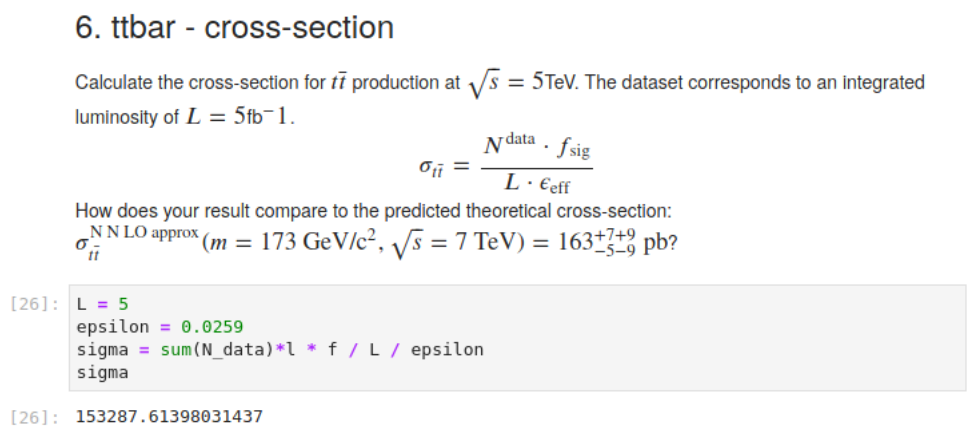
- Quelloffenheit und offene Standards
- Browserbasiertheit
- aktive Community
- breite Anwendung der Software auf mehrere Sprachen wie *Python*, *R*, *Julia*, *C++* oder *Mathematica*

Der Kern des Projekts wird von dem Dokumentenformat *Jupyter-Notebook* gebildet. In einer Zellenstruktur werden in ihm Programmcode und Text zu einem Dokument verbunden, wie es in Abbildung 2.1 beispielhaft dargestellt ist. Dabei können Zellen entweder Code oder Text enthalten. Ausgaben, die von Code produziert werden, werden unterhalb der entsprechenden Zelle dargestellt und ebenfalls gespeichert.

Der Text wird mit *Markdown* dargestellt [Gru]. Diese Syntax zur Formatierung von Text zielt darauf ab, sowohl eine gute Lesbarkeit zu bieten, als auch mit simplen Strukturelementen diesen mit Bildern, Listen oder Überschriften zu versehen. Dazu werden Formatierungsanweisungen im Klartext einbezogen und können dann von entsprechenden Programmen interpretiert oder in *HTML*- oder *LaTeX*-Darstellungen überführt werden. In Notebooks wird die Syntax um *LaTeX*-Elemente erweitert, sodass z. B. Formeln in *LaTeX*-Syntax auch genutzt werden können.

Zur Bearbeitung dieser Dokumente existieren zwei browserbasierte Softwarelösungen: Das gleichnamige *Jupyter-Notebook*[1] und dessen Nachfolger, *JupyterLab* [Jupa]. Die technische Grundlage bilden in beiden Lösungen vor allem die Kernel. Sie übernehmen die Interpretation des Programmcodes eines Notebooks und stehen inzwischen für viele Sprachen zur Verfügung. *IPython* [PG07] ist dabei der Standard-Kernel zur Interpretation von Notebooks in *Python* und wird von *Project Jupyter* selbst unterhalten. Die Kernel anderer Sprachen dagegen werden überwiegend von der Community bereitgestellt und weiterentwickelt. Mit *Jupyter-Notebook* ist man darauf beschränkt, nur ein Notebook zur gleichen Zeit bearbeiten zu können. Dagegen bringt *JupyterLab*, neben weiteren Verbesserungen

[1] Zur Unterscheidung der Software *Jupyter-Notebook* und dem Dokumentenformat, wird in dieser Arbeit mit Notebook stets das Dokumentenformat bezeichnet. *Jupyter-Notebook* wird im Folgenden für die Software verwendet.



6. ttbar - cross-section

Calculate the cross-section for $t\bar{t}$ production at $\sqrt{s} = 5$TeV. The dataset corresponds to an integrated luminosity of $L = 5\text{fb}^{-}1$.

$$\sigma_{t\bar{t}} = \frac{N^{\text{data}} \cdot f_{\text{sig}}}{L \cdot \epsilon_{\text{eff}}}$$

How does your result compare to the predicted theoretical cross-section:
$\sigma_{t\bar{t}}^{\text{N N LO approx}}(m = 173\ \text{GeV/c}^2, \sqrt{s} = 7\ \text{TeV}) = 163^{+7+9}_{-5-9}$ pb?

[26]:
```
L = 5
epsilon = 0.0259
sigma = sum(N_data)*l * f / L / epsilon
sigma
```

[26]:
```
153287.61398031437
```

Abbildung 2.1.: Beispiel zur Zellenstruktur eines Notebooks. Der Ausschnitt aus der neuen Version des Übungsblattes zur Bestimmung der Top-Quark-Masse zeigt oben eine *Markdown*-Zelle mit der Beschreibung der Aufgabe. Darauf folgt eine mit Code, unter der dann dessen Ausgabe steht. Die Nummern neben den Code-Zellen indizieren die Reihenfolge, in der sie ausgeführt wurden.

der Nutzeroberfläche, ein Tab-System und das parallele Arbeiten an mehreren Notebooks mit sich.

Die Nutzung der *Jupyter*-Software durch mehrere Nutzer kann mit *JupyterHub* organisiert werden. Mit ihm können mehrere Nutzer Zugriff auf eigene Instanzen von *JupyterLabs* oder *Jupyter-Notebooks* erhalten. Diese Zugriffe erfolgen über das Netzwerkprotokoll *HTTPS*, wodurch *JupyterHub* sowohl lokal auf einem Rechner betrieben werden kann, als auch in lokalen Netzwerken und dem Internet. Er unterstützt gängige Authentifizierungsverfahren wie *Linux PAM*, *LDAP* oder *OAuth* und startet Instanzen der beiden Softwarelösungen mit sogenannten *Spawnern*. Eine weitere Personalisierung seiner Funktionen lässt sich leicht über Konfigurationsdateien erreichen. Für darüber hinausgehende Anpassungen muss auf den offenen Programmcode zurückgegriffen werden.

Obwohl der Fokus des Projektes bisher auf den namensgebenden Sprachen **Ju**lia **Pyt**hon und **R** liegt, wurden auch zusätzliche Projekte aufgenommen wie *Xeus* (eine *C++*-Implementierung des *Jupyter* Kernelprotokolls [Mab20]) oder *Viola* (eine Erweiterung zur Präsentation von Notebooks, [Cor19]).

2.2. Vorteile der Nutzung von Notebooks in der Lehre

Das Buch „Teaching and Learning with Jupyter“[LABZ19] liefert eine ausführliche Beschreibung der didaktischen und technischen Vorteile von Notebooks in der Lehre. Hier soll eine kurze Zusammenfassung der Kapitel 2 („Why we use Jupyter notebooks“) und 3 („Notebooks in teaching and learning“) die wichtigsten Vorteile im Physikstudium darlegen.

Notebooks stellen ein sehr flexibles Dokumentenformat dar. Neben den Funktionen aus *Markdown*, sorgen viele Erweiterungen für zahlreiche unterschiedliche Verwendungen. Übungen bis hin zu ganzen Kursen können ebenso realisiert werden, wie Multimediaplattformen mit Videos und Bildern geschaffen werden können. Erweiterungen wie interaktive Widgets und auf konkrete Gebiete spezialisierte Darstellungsformen können Notebooks hin zu einer Applikation umgestalten.

Das macht Notebooks auch zu einem interaktiven Medium. Die Analysen mit eigenen Datensätzen durchzuführen, sie mit anderen Parametern zu wiederholen oder eigene Untersuchungen selbst hinzuzufügen, ermöglicht eine Art Interaktion mit Daten und Analysemethoden. Dadurch wird aus dem reinen Konsumieren von Methodiken und Ergebnissen ein

„Learning-by-doing“. Große Vorteile bringt dies vor allem Programmieranfängern. Neben der Möglichkeit, ausgehend von vorgegebenen, funktionierenden Beispielen, selbst Lösungen zu entwickeln und zu experimentieren, bieten Notebooks einen leichten Einstieg, in dem sich Studierende auf die wesentlichen Konzepte des Programmierens konzentrieren können. Der Weg vom Schreiben des Codes zu dessen Ergebnis ist sehr kurz und kommt aus Sicht des Nutzers ohne Compiler oder zusätzliche Software aus.

Auch die Reproduzierbarkeit der Notebooks bringt Vorteile im Bereich des Studiums, insbesondere in den Laborpraktika. Statt nur die Methoden der Analyse und deren Ergebnisse zu präsentieren, werden mit ihnen die konkreten Rechenschritte gezeigt. Dies bietet die Möglichkeit gezielteres Feedback zur Arbeit der Studierenden zu geben. So sind nicht nur Formfehler einfacher von inhaltlichen Fehlern zu trennen, auch Übertragungsfehler können leichter entdeckt werden. Ebenso kann ohne großen Aufwand geprüft werden, ob unerwartete Ergebnisse, wie große Abweichungen, in den Daten begründet sind oder von den Rechenmethoden verursacht wurden.

3. EnJoY - virtuelle Umgebungen für Notebooks

E-Lear**n**ing with **J**upy**t**er, kurz *EnJoY*, ist ein *JupyterHub*, der virtuelle Umgebungen, in Form von sogenannten Containern, jedem Nutzer bereitstellt. Es wurde mit den Zielen entwickelt Studierenden Zugang zu einer einheitlicher Software-Umgebung zu ermöglichen und Rechenkapazität für Kurse bereitszustellen. Mit *JupyterLab* in virtuellen Umgebungen stellt es, durch Verwendung von Container-Orchestrierungs-Werkzeugen wie „*docker swarm*“, ein leicht skalierbares System für den Einsatz von Notebooks in der Lehre dar. Ursprünglich wurde es von Manuel Giffels für Kurse der GridKa School entwickelt [Gif19].

3.1. Motivation zur Virtualisierung

Um die Reproduzierbarkeit von Notebooks voll ausnutzen zu können, müssen Lesern die gleichen Versionen der verwendeten Softwarepakete wie den Erstellern zur Verfügung stehen. Dies bringt eine Reihe an Herausforderungen mit sich.

Studierende beginnen das Studium mit unterschiedlichen Hardware-Voraussetzungen und Vorkenntnissen im Bereich des Programmierens. Entsprechend wählen sie ihre Werkzeuge und Programme für z. B. Laborpraktikas oder Computer-Übungen. Daraus resultiert ein breites Spektrum an Software von einfach zu bedienenden Anwendungen wie Tabellenkalkulationsprogrammen, über spezialisierte Programme wie *originlab* [Cor], hin zu den *Python*-Bibliotheken wie *numpy* [vCV11] oder *kafe* [Să13]. Damit sind Analysen und Auswertungen, die spezielle Software nutzen, nicht auf jedem Computer lauffähig und Ergebnisse können unter Umständen nicht nachvollzogen werden. Zusätzlich erschwert diese die Zusammenarbeit der Studierenden im Rahmen der Laborpraktika.

Eine Lösung dafür bieten die Computer-Pools der Fakultät. Diese werden zentral verwaltet und dadurch ist auf jedem Computer auch die gleiche Software installiert. Die Anzahl an verfügbaren Computer-Plätzen ist aber auf etwa 30 beschränkt. Dem stehen rund 600 [Kit] Studierende im Bachelor-Studiengang Physik gegenüber.

Des Weiteren musste im Zuge der COVID-19-Pandemie 2020 der Poolraum geschlossen werden. Zwar ist ein Zugriff auf die Computer dort prinzipiell per *ssh*-Protokoll möglich, dies ist aber nicht komfortabel und insbesondere nicht einsteigerfreundlich.

Eine Virtualisierung dieser Computer-Arbeitsplätze bietet sich als Lösung an. Die Idee der Virtualisierung ist physische Ressourcen zusammenzufassen und gemeinsam zu verwalten.[Bau11] Dafür werden Abstraktionsebenen wie Hypervisoren eingeführt, die diese Reorganisation übernehmen. Die Vorteile dieser Herangehensweise sind unter anderem eine bessere Auslastung der physischen Ressourcen, deren dynamische Anpassung entsprechend des Bedarfs und Platzersparnis durch Nutzung weniger, leistungsstarker Server anstatt vieler einzelner Computer [Bau11].

3.1.1. Zielsetzungen der Virtualisierung

Mit der Virtualisierung werden im Fall von *EnJoY* folgende Ziele angestrebt:

- Notebooks können im Rahmen eines Kurses voll reproduziert und nachvollzogen werden. Dazu wird angestrebt, dass
  - Datenanalysen und -auswertungen von den Studierenden unabhängig ihrer Hardware- und Softwarevorraussetzungen in angemessener Qualität durchgeführt werden können.
  - Allen Studierenden die gleichen Bibliotheken in der selben Version zur Verfügung stehen.
- Die Anzahl an verfügbaren Computer-Arbeitsplätze wird erhöht.
- Den Studierenden steht persistenter Speicherplatz in der virtuellen Umgebung zur Verfügung.

3.2. Virtualisierung durch Containerisierung mit *Docker*

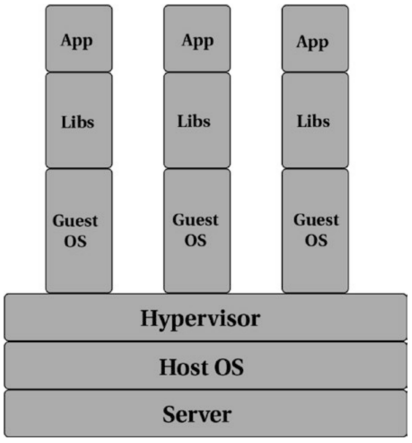


Abbildung 3.1.: [Bha18]: Repräsentation dreier Applikationen in drei virtuellen Maschinen. Auf einem Server wird ein Betriebssystem ausgeführt (Host OS). Dieses startet einen Hypervisor, der wiederum stellt virtuelle Ressourcen einem weiteren Betriebssystem (Guest OS) zur Verfügung. In dem befinden sich wieder eigene Bibliotheken, die den Betrieb der Applikation ermöglichen.

Auf Grundlage des Buches "Practical Docker with Python"[Bha18] wird im Folgenden das Konzept der Container in *Docker* vorgestellt und die in *EnJoY* verwendeten Funktionen erklärt.

Container sind eine leichtgewichtige Form der Virtualisierung auf Ebene des Betriebssystems [Bha18]. Sie führen keine Hardwareemulation durch, wie es die bekannteren virtuellen Maschinen tun. Stattdessen bilden sie einen vom Gastsystem abgetrennten Bereich mit eigenen Ressourcen wie Bibliotheken oder Dateisystemen. Dadurch benötigen sie im Allgemeinen auch weniger Ressourcen und können schneller gestartet und gestoppt werden. Die Unterschiede zwischen virtuellen Maschinen und Containern werden in Abbildung 3.1 und Abbildung 3.2 illustriert.

Erste Ansätze für Container gehen auf den Unix-Befehl *chroot* zurück, der es ab 1979 ermöglichte für einzelne Prozesse ein neues Wurzel-Verzeichnis zu definieren und damit Teile des Dateisystems für diesen unsichtbar machte. Dies wurde in den 2000ern mit *FreeBSD* (Partitionierung in „jails“) und *OpenVZ* (geteilter Linuxkernel für mehrere Betriebssysteme

auf einem Server) ausgebaut. Mit *cgroups* konnten Ressourcen wie CPU und Speicher limitiert und isoliert werden. Ausgehend davon, und den *namespaces* des Linux Kernels, stellt *Docker* Software zum einfachen Umgang mit Containern bereit.

Container bieten also eine gut geeignete Möglichkeit zur Virtualisierung von Arbeitsumgebungen. Sie lassen sich zum einen schnell starten, zum anderen benötigen sie wenige Ressourcen und erlauben dadurch eine einfache Skalierung des Angebots an Arbeitsumgebungen für Studierende.

3.2.1. Kernkonzepte im Umgang mit *Docker*

Wie konkret ein Container aufgebaut ist, wird durch ein sogenanntes *Dockerfile* beschrieben. In ihm wird eine Folge von Instruktionen definiert, mit denen Bibliotheken installiert oder Programme ausgeführt werden.
Beim *Bauen* des *Dockerfiles* entsteht das *Image*. Der *Docker-Container* ist der mit dem *Image* gestartete Prozess. Änderungen, wie das Bearbeiten einer Datei, die in einem *Docker-Container* gemacht werden, gehen nach Beenden des Containers verloren. Um das zu verhindern existieren zwei Möglichkeiten:
Volumes ist der empfohlene Weg *Docker*-Containern persistenten Speicher zur Verfügung zu stellen. Sie sind unabhängig vom Hostsystem und werden von der *Docker*-Engine erstellt und verwaltet. Zusätzlich lassen sich auch Verzeichnisse aus dem Gastsystem in den Container einbinden.
Zuletzt bietet *docker-compose* die Möglichkeit eine Gruppe von Containern für eine Anwendung gemeinsam zu starten und zu koordinieren. Es erstellt ein virtuelles Netzwerk, dass es Containern erlaubt miteinander zu kommunizieren.
Es sei an dieser Stelle noch auf die sogenannten *Secrets* verwiesen. Diese ermöglichen das sichere Verwalten von Daten wie Passwörtern, Schlüsseln oder Zertifikaten. Sie können nur von den Containern gelesen werden, denen explizit Zugang gewährt wurde. [Inca]

3.3. Struktur von *EnJoY*

EnJoY an sich ist aus einer Reihe an Diensten aufgebaut, die in eigenen Containern auf einem Server ausgeführt werden. Ihre Koordination wird mit *docker-compose* durchgeführt und ihre Kommunikation nutzt dessen virtuelles Netzwerk. Abbildung 3.3 zeigt ein Schema der Struktur von *EnJoY*.
Für die Implementierung in den Lehrbetrieb wurde es mit dem bestehenden System des Poolraumes verknüpft. So erfolgt die Authentifizierung auf dem *JupyterHub* über die

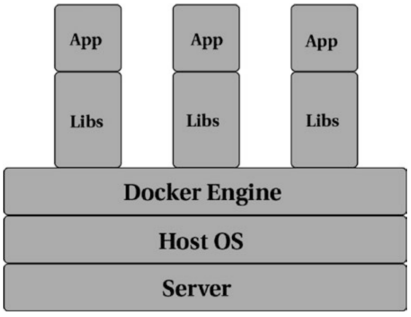


Abbildung 3.2.: [Bha18]: Repräsentation dreier Applikationen in drei verschiedenen Containern. Wieder läuft auf einem Server ein Betriebssystem mit einer weiteren Abstraktionsebene, der *Docker*-Engine. Diese stellt die Container als isolierte Bereiche des Linux-Kernels (*namespaces*) mit den Bibliotheken und der Applikation zur Verfügung. Es ist kein eigenes Betriebssystem für jede Applikation nötig.

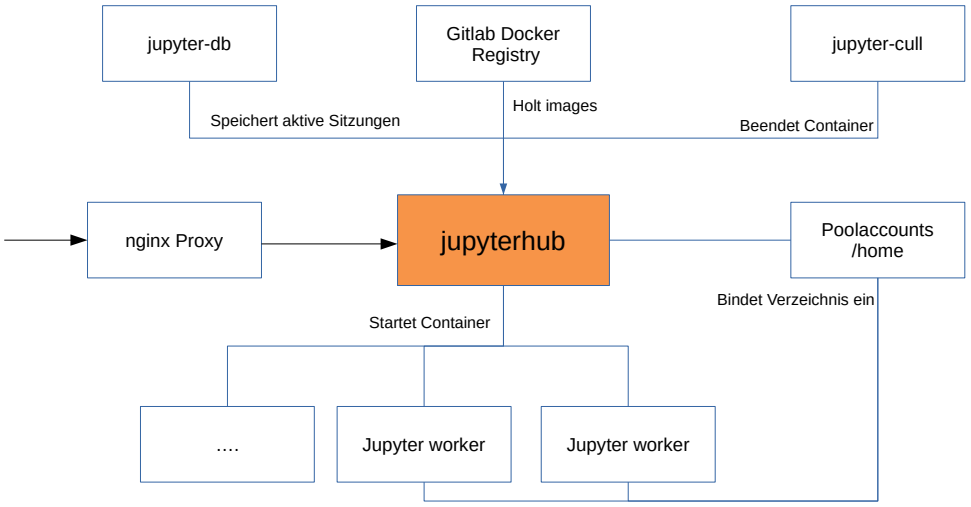


Abbildung 3.3.: Schematische Darstellung der Virtualisierungslösung: Der *JupyterHub* regelt als zentraler Knotenpunkt die Anmeldung der Nutzer und das Bereitstellen der Jupyter-Worker-Container. Der bisherige Poolraumaccount dient als Gastsystem und stellt die Arbeitsverzeichnisse als persistenten Speicher bereit.

Authentifizierungsdaten des Poolraumes. Ebenso wird das Verzeichnis des Nutzers in seinen Container mit eingebunden. Damit fügt sich *EnJoY* in das bestehende Softwareangebot für Studierende ein und erfordert keinen Aufbau eigener Infrastruktur.

Möchte ein Nutzer eine virtuelle Umgebung erhalten, meldet er sich zuerst auf dem *JupyterHub* an. Dieser bietet ihm Images für verschiedene Zwecke an. Deren Inhalte werden in der Regel von den Übungsleitern für die Bearbeitung ihrer Aufgaben erstellt und in einer Registry gespeichert[1]. Aus diesem *Image* erstellt der *JupyterHub* ein Container, den *Jupyter-Worker*, und bindet das `/home`-Verzeichniss des Nutzers ein. Welcher Nutzer zu welchem derzeit laufenden Container gehört, wird in der *Jupyter-DB* gespeichert. Diese Datenbank wird von *JupyterHub* erstellt und in einen eigenen Container verlegt, so dass ein Neustart des *JupyterHubs* an den vorherigen Zustand des Systems anknüpfen kann. Der Dienst *Jupyter-Cull* kontrolliert die laufenden Container und beendet sie nach einer bestimmten Zeit der Inaktivität. Damit wird verhindert, dass der Server mit nicht länger benötigten Containern belastet wird.

3.4. Wartung von *EnJoY* mit CI/CD und Registry

Um den Anforderungen in der Lehre gerecht zu werden, muss *EnJoY* eine dauerhafte Erreichbarkeit gewährleisten. Gleichzeitig besteht die Notwendigkeit Updates für *Images* einzuspielen oder für neue Veranstaltungen neue *Images* einzubringen. Dafür ist es notwendig, Teile von *EnJoY* neu zu starten. Ebenso erleichtert eine Automatisierung der Schritte vom *Dockerfile* zum fertigen *Jupyter-Worker*-Container die Wartung des Systems.

Damit ist eine Lösung notwendig, die die Stabilität von *EnJoY* gewährleistet, während gleichzeitig Updates und Bugfixes eingefügt werden können. Eine solche Softwarelösung nennt sich Continuous Integration/Continuous Deployment, kurz CI/CD [Len19]. In der Entwicklungsplattform *GitLab* [Incc] sind bereits die notwendigen Werkzeuge dafür vorhanden.

[1] Im folgenden Abschnitt 3.4 wird darauf näher eingegangen.

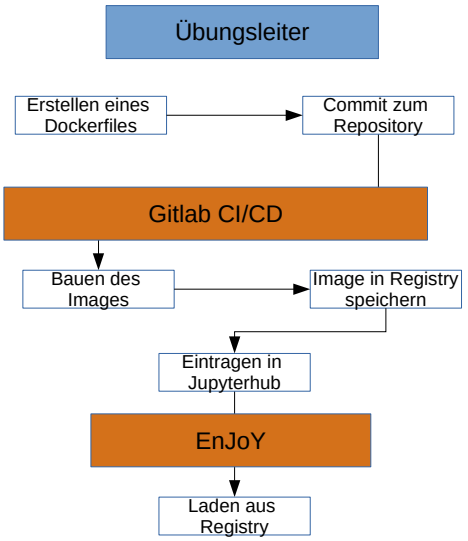


Abbildung 3.4.: Ablauf beim Einfügen neuer Container in *EnJoY*. Ein Übungsleiter erstellt ein *Dockerfile*. Die CI/CD übernimmt dann das Bauen des *Images*, speichert es in der Registry und gibt die Information über das neue *Image* an den *JupyterHub* weiter. *EnJoY* kann diese dann bei Bedarf nutzen.

GitLab ist eine Plattform zur Entwicklung und Betreuung von Software auf Basis des Versions-Kontoll-System *git* [gC]. Zu *gitlabs* zahlreichen Funktionen gehören unter anderem auch eine Docker-Registry [Incb] und eine Implementierung von Continuous Integration/Continuous Deployment [Incc]. Damit lässt sich ein Prozess konstruieren, der aus *Dockerfiles Images* erstellt, diese in der Registry speichert und dann von *EnJoY* wieder heruntergeladen werden können. Abbildung 3.4 zeigt schematisch, wie dies abläuft.

Als Grundlage für ein neues *Dockerfile* dient das *Base-Worker-Image*, dass grundlegende Funktionen wie die Verbindung zum *JupyterHub*, *JupyterLab* und einen *Python*-Kernel installiert. Von diesem ausgehend können neue Kernel für weitere Sprachen und neue Bibliotheken sowie Erweiterungen für *JupyterLab* installiert werden. Aus diesem *Dockerfile* wird dann durch die CI/CD ein *Image* erstellt und in der Registry gespeichert. Dabei wird der Name des *Images* in eine Datei des *JupyterHubs* geschrieben. Im Anschluss muss der *JupyterHub* neu gestartet werden. Durch die *Jupyter-DB* mit persistentem Volume und den Proxy, der die Verbindung der Nutzer zu ihren Containern aufrecht erhält, sind diese dadurch nicht beeinträchtigt. Der gleiche Ablauf kann auch zum Updaten bestehender *Images* genutzt werden, wobei hier das gesamte *Images* neu erstellt wird.

4. Beispielhafte Umsetzung anhand der Übung zur Bestimmung der Topmasse

Wie eine Übung als Notebook in *EnJoY* ablaufen und aussehen kann, wird anhand der Übung zur Bestimmung der Top-Quark-Masse demonstriert.
Die Übung soll den Studierenden die Analyse von Top-Quark-Ereignissen aus Daten des CMS-Detektors näherbringen und gehört zu den Inhalten der Vorlesung Teilchenphysik 1 im Masterstudium. In ihrer derzeitigen Form basiert sie auf der Arbeit von Matthias Schnepf „Erstellung eines Praktikumsversuches zur Untersuchung von Top-Quark-Paar-Ereignissen am LHC mit dem CMS-Experiment“[Sch13], die auch weiterhin die Grundlage der Übung bildet. Größte inhaltliche Änderung ist eine neue Parametrisierung der Top-Quark-Massen-Verteilung durch die Crystal-Ball-Funktion. Ferner wird in den Unsicherheiten nun auch die Unsicherheit der Jet Energy Scale mit einbezogen.

4.1. Physikalischer Hintergrund und verwendete Datensätze

Im Standardmodell der Teilchenphysik stellt das Top-Quark mit $(173{,}1 \pm 0{,}9)\,\mathrm{GeV}$ [THH$^+$18] das schwerste Teilchen dar. Aufgrund seiner sehr kurzen mittleren Lebensdauer von $\tau_{\mathrm{Top}} = 2.0 \cdot 10^{-24}\mathrm{s}$ [Sch13] ist sein Nachweis bisher nur künstlich über Teilchenbeschleuniger möglich.
In der Übung werden deshalb Daten aus den Messungen des CMS-Detektors von 2010 bei einer Schwerpunktsenergie von $\sqrt{s} = 7\,\mathrm{TeV}$ verwendet. Darin wird nach Ereignissen gesucht, bei denen Paare aus Top-Quarks und Anti-Top-Quarks entstehen, denn deren Produktionswirkungsquerschnitt liegt mit $\sigma_{t\bar{t}} = 163^{+7+9}_{-5-9}\mathrm{pb}$ deutlich höher, als der für die Entstehung einzelner Top-Quarks ($\sigma_t + \sigma_{\bar{t}} = 86.06^{+2.83}_{-195}\mathrm{pb}$ [Sch13]).

Top-Quarks zerfallen dann fast immer unter Einfluss der schwachen Wechselwirkung in ein b-Quark und ein W-Boson. Das b-Quark löst mittels Hadronisierung einen Teilchenschauer (Jet) aus, der detektiert wird und mittels eines Algorithmus mit einer gewissen Wahrscheinlichkeit dem b-Quark zugeordnet wird. Dieses Verfahren wird b-Tagging genannt.
Das W-Boson zerfällt entweder in zwei leichtere Quarks (hadronischer Zerfall) oder in ein Lepton und sein Neutrino (leptonischer Zerfall).
Damit sind drei Zerfallsprozesse für den Zerfall eines Top-Anti-Top-Paares möglich: Voll hadronischer, dileptonischer und semileptonischer Zerfall. Die Übung verwendet nur semileptonische Zerfälle mit einem Myon als Lepton.

Um einen besseren Signalanteil in den Daten zu erhalten und deren zu untersuchende Menge zu beschränken wurden Ereignisselektionen durchgeführt. Ein Teil dieser wird mit der Effizienz und Sensitivität der Myonen-Triggern und der Rekonstruktionseffizienz der Jets begründet:

- Es wurde mindestens ein isoliertes Myon registriert im Trigger-Arbeitsbereich von $|\eta| < 2.1$ und $p_t > 45\,\mathrm{GeV}$

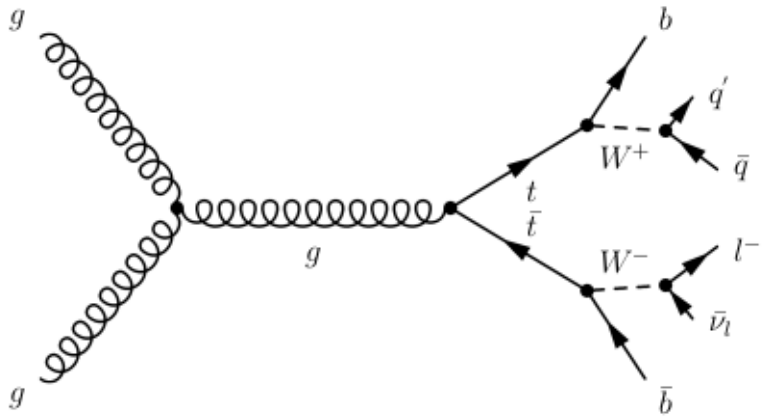


Abbildung 4.1.: [Sch13]: Feynmann-Diagramm des semileptonischen Zerfalls eines $t\bar{t}$-Paares aus Gluonfusion. Ein W-Boson zerfällt in leichtere Quarks (q, $\bar{q}$), das andere in ein Lepton mit seinem Neutrino (l,ν_l).

- Jets liegen im Bereich von $|\eta| < 2.4$ und $p_t > 30\,\text{GeV}$

Die dabei verwendeten Größen Pseudorapidität η und Transversalimpuls p_t sind wie folgt definiert:

$$\eta = \text{arctanh}(\frac{p_z}{|\vec{(p)}|}) \tag{4.1}$$

$$p_t = \sqrt{p_x^2 + p_y^2} \tag{4.2}$$

Zusätzlich wurden alle Ereignisse mit weniger als zwei Jets aussortiert.

Abgeglichen werden die realen Ereignisse mit simulierten Datensätzen. Solche Monte-Carlo-Daten wurden mit *Powheg* [Ole10] für Einzel-Top-Quark-Ereignisse erstellt. Als Hintergrund wurde zur Vereinfachung nur der dominante Prozess der W+Jets betrachtet. Die Simulationen dazu wurden mit *MadGraph* durchgeführt.

Die Studierenden sollen in der Übung einen Blick auf die Kinematik der Top-Quark-Ereignisse werfen. Anschließend wird nach geeigneten Variablen zur Diskriminierung des Untergrundes gesucht. Dann wird mit den simulierten Daten das Verhältnis von Signal zu Untergrund mit und ohne b-Tag bestimmt. Auch eine Rekonstruktion der Top-Quark-Vierervektoren ist Teil der Aufgaben. Die abschließende Aufgabe der Parametrisierung der Massenverteilung der rekonstruierten Top-Quarks und die Bestimmung der Masse werden hier überarbeitet.

4.2. *pandas* und CSV als Datenformate

In der derzeitigen Fassung der Übung werden Daten aus *ROOT*-Dateien [ABB+09] mit dem *KITA*-Framework ausgelesen. Das *KITA*-Framework ist eine Software, die von der Top-Quark-Gruppe des KIT entwickelt wurde. Dieses wird aber nicht mehr verwendet und deshalb sind Kenntnisse über das Framework für Studierende nicht mehr relevant.

An die Stelle von *KITA* und *ROOT*-Dateien, treten mit *CSV* und *pandas* [WM10] weit verbreitete, aktuelle Datenformate. Mit ihnen wird die grundlegende Struktur der Daten als eine Menge von Ereignissen in eine tabellenartige Form gebracht. Sie enthält jede Zeile ein Ereignis mit den zugehörigen Teilcheneigenschaften in den Spalten. Für Jets, Myonen und weitere Teilchen wird mit dem Vierervektor eine sehr grundlegende Beschreibung der Kinematik der Teilchen abgespeichert.

Die in der Auswertung wichtigen Kennzahlen wie Transversalimpuls p_t oder Pseudorapidität η müssen dann von den Studenten selbst berechnet werden.

Mit *pandas* `Dataframe`-Objekt stehen diese Daten im Notebook in einer einfachen, schnellen und ausdrucksstarken Form zur Verfügung, die flexibles Arbeiten an den Daten erlauben [McK19]. Ursprünglich im Hinblick auf Fragestellungen der Finanzanalysen entworfen, ist es nun ein verbreitetes Werkzeug zur Analyse großer und komplexer Daten in *Python*. Verstanden werden können `DataFrames` als Tabellen; Ein Satz Daten ist in einer Zeile eingetragen. Intuitiv wird der Zugriff über das Ansprechen von Zeilen oder Spalten mittels Schlüsselwörtern. So hat das `DataFrame` zum Beispiel eine Spalte mit der Überschrift „Myonen“und und nur durch Angabe dieses Schlüsselwortes kann auf diese zugegriffen werden. Dadurch wird der Code besser lesbar und verständlicher, da im Unterschied zu klassischen Arrays keine Kenntnisse darüber nötig sind in welcher Reihenfolge Daten abgespeichert sind. Auch das Hinzufügen von neuen Spalten erfolgt unter Angabe eines Schlüsselwortes.

In Abbildung 4.2 ist ein Beispiel der Datenstrukturen der Übung in *pandas* gegeben. Es sein angemerkt, dass die Vierervektoren selbst als *numy*-Arrays [vCV11] dargestellt werden. Das trägt insbesondere der variierenden Anzahl der Jets Rechnung.

	Jets	missing E	Muons	Jets bTag
0	[[108.857, -52.5515, 2.04588, 94.0711], [183.4...	[23.0081, 16.7103, 15.8158, 0.0]	[34.6515, -12.3001, -24.3221, 21.3976]	[0.135533, 0.288028]
1	[[46.1862, -38.5572, -14.8651, 19.3557], [93.9...	[49.0855, 42.5448, 24.4812, 0.0]	[65.6198, -33.3948, -4.83266, 56.2795]	[0.109713, 0.144369]
2	[[186.467, -33.7477, -56.6337, -173.84], [48.0...	[50.9159, 45.0782, -23.6725, 0.0]	[79.4006, 17.4953, 47.513, -61.1627]	[0.0178308, 0.114022]
3	[[163.18, 91.6156, 9.55574, 134.192], [109.703...	[24.8113, 24.605, -3.19299, 0.0]	[333.093, -121.942, -57.8864, 304.517]	[0.212426, 0.0306883]
4	[[110.863, -31.5028, 42.8955, -96.8073], [112....	[32.0489, -7.66629, -31.1185, 0.0]	[77.6548, 38.9837, 16.1485, 65.1901]	[0.106201, 0.0992101]

Abbildung 4.2.: Auszug des Notebooks für die Übung zur Bestimmung der Top-Quark-Masse. Dargestellt sind die ersten Einträge eines *pandas*-`DataFrame`. Die dargestellten Ereignisse entstammen der Hintergrundsimulation für W+Jets. Die Myonen werden durch einen *numpy*-Array als Vierervektor beschrieben. Gleiches gilt für die missing Energy. Für die Jets wurde ein mehrdimensionaler *numpy*-Array gewählt. Der Array für die b-Tags der Jets folgt der Reihenfolge der Jets in ihrer Spalte.

4.3. *kafe2* und *matplotlib* zur Anpassung und Visualisierung

Das Notebook der neuen Übung verzichtet auf *ROOT* mit seinen Werkzeugen zur Datenanpassung und -visualisierung. Die Alternativen in *Python* sind die Umfangreiche Bibliothek zur Graphenerstellung *matplotlib* [Hun07] und das Karlsruhe Fit Enviroment 2 (kafe2) [QGCS] zur Funktionsanpassung.

Kafe2 ist der Nachfolger des Werkzeugs *kafe* [Să13]. Es bietet eine einfache Schnittstelle zu Funktionsminimierern wie *IMINUIT* [Teaa], die gezielt auf die Bedürfnisse von Studierenden zugeschnitten ist. Mit der zweiten Version wurde ein modularer Aufbau der Software realisiert, der weitere Arten der Funktionsanpassung wie Histogramm-Fits oder Unbinned-Fits ermöglicht [Ver19].

Sowohl die von *kafe2* generierten, als auch alle weiteren Graphen werden mit *matplotlib* [Hun07] erstellt. Es wurde mit der Philosophie entwickelt einfache Graphen mit nur wenigen Befehlen kreieren zu können, und zeichnet sich folglich durch eine leichte Handhabung und eine weite Verbreitung aus. In Notebooks können die Graphen durch Erweiterungen wie *ipympl* [mT], einer Integration von *matplotlib* in *Jupyter* als interaktive Widgets, leicht eingebunden werden.

4.4. Reparametrisierung der Top-Quark-Massen-Verteilung

Im Laufe der Analyse werden die Vierervektoren der Top-Quarks rekonstruiert und ihre Masse ermittelt. Dazu werden mögliche Kombinationen der Jets durchgegangen und mittels einer $\mathcal{X}^2$-Funktion verglichen. Dabei werden nur Kombinationen untersucht, die mindestens einen Jet mit einem b-Tag größer 0.244 aufweisen. Es wird dann die Masse des Neutrinos mit dem on-shell-Verfahren [Sch13] rekonstruiert und die vektorielle Addition zu den beiden Vierervektoren der Top-Quarks durchgeführt. Das Ergebnis wird durch die Kostenfunktion

$$\chi^2 = \frac{(m_{W_{\text{had}}} - M_{W_{\text{had,exp}}})^2}{\sigma^2_{M_{W_{\text{had,exp}}}}} + \frac{(m_{t_{\text{lep}}} - m_{t_{\text{had}}} - \Delta M)^2}{\sigma^2_{\Delta M_{t_{\text{exp}}}}} \tag{4.3}$$

bewertet und die Kombination mit dem geringsten Wert als wahrscheinlichste Rekonstruktion weiterverwendet. Aus den simulierten Daten wurden die Werte der Massenverteilungen des W-Bosons und der Massendifferenz bereits rekonstruiert und den Studierenden vorgegeben.

Um aus der nun erhalten Massen-Verteilung den Wert der Top-Quark-Masse zu gewinnen muss eine passende Parametrisierung gefunden werden. Dazu wird eine geeignete Funktion an die Monte-Carlo-Daten für den Zerfall der Top-Quarks angepasst. Die gefundenen Form-Parameter werden nun für die Anpassung an die gemessenen Daten fixiert. Lediglich Verschiebe- und Skalierungsparameter werden angepasst. Dies wurde bisher mit einer Landau-Funktion durchgeführt.

$$f_{\text{Landau}}(x; m, b, A) = \frac{A}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-m}{b} + e^{\frac{x-m}{b}}\right)\right). \tag{4.4}$$

Wie Abbildung 4.3 zeigt, liefert diese aber nur eine grobe Näherung an die eigentliche Verteilung der Datenpunkte.

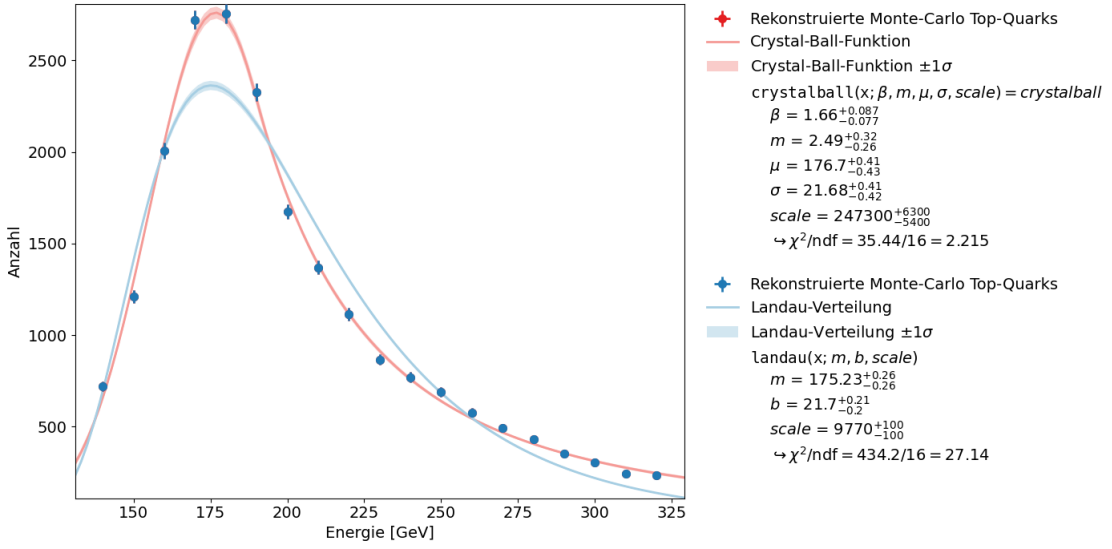


Abbildung 4.3.: Vergleich der Anpassung der Monte-Carlo-Daten an eine Landau-Verteilung und die Crystallball-Funktion.

Eine deutlich bessere Beschreibung lässt sich mit der Crystal-Ball-Funktion [Org80] erreichen.

$$f_{\text{cb}}(x'; \alpha, m, \bar{x}, \sigma) = N \cdot \begin{cases} \exp\left(-\frac{(x'-\bar{x})^2}{2\sigma^2}\right) & \text{, für } \frac{(x'-\bar{x})}{\sigma} > -\alpha \\ A \cdot \left(\frac{m}{|\alpha|} - |\alpha| - \frac{(x'-\bar{x})}{\sigma}\right)^{-m} & \text{, für } \frac{(x'-\bar{x})}{\sigma} \leq -\alpha \end{cases}, \tag{4.5}$$

mit

$$A = \left(\frac{m}{|\alpha|}\right)^m \cdot \exp\left(-\frac{|\alpha|^2}{2}\right),$$

$$N = \frac{1}{\sigma(C+D)},$$

$$C = \frac{m}{|\alpha|(m-1)} \cdot \exp\left(-\frac{|\alpha|^2}{2}\right)$$

und

$$D = \sqrt{\frac{\pi}{2}}\left(1 + \operatorname{erf}\left(\frac{|\alpha|}{\sqrt{2}}\right)\right).$$

Diese abschnittsweise definierte Funktion hat einen gaussförmigen Teil, der mit einem Polynom als asymmetrischen Ausläufer kombiniert ist. Für eine bessere Darstellung der Parameter wurden zusätzlich noch folgende Substitutionen durchgeführt:

- $x' = -x$,
- $\bar{x} = -\mu$,
- $\alpha = \frac{1}{\beta}$.

Mit ihr lässt sich eine deutlich bessere Parametrisierung finden. Wie Abbildung 4.3 zeigt, ergibt sich ein um etwa 92% geringeren $\mathcal{X}^2$-Wert im Vergleich zur Landau-Verteilung. Auch in der Beschreibung der gemessenen Daten zeigt sich dies, wie Abbildung 4.4 veranschaulicht: Rund 57% geringer fällt der $\mathcal{X}^2$-Wert hier aus.

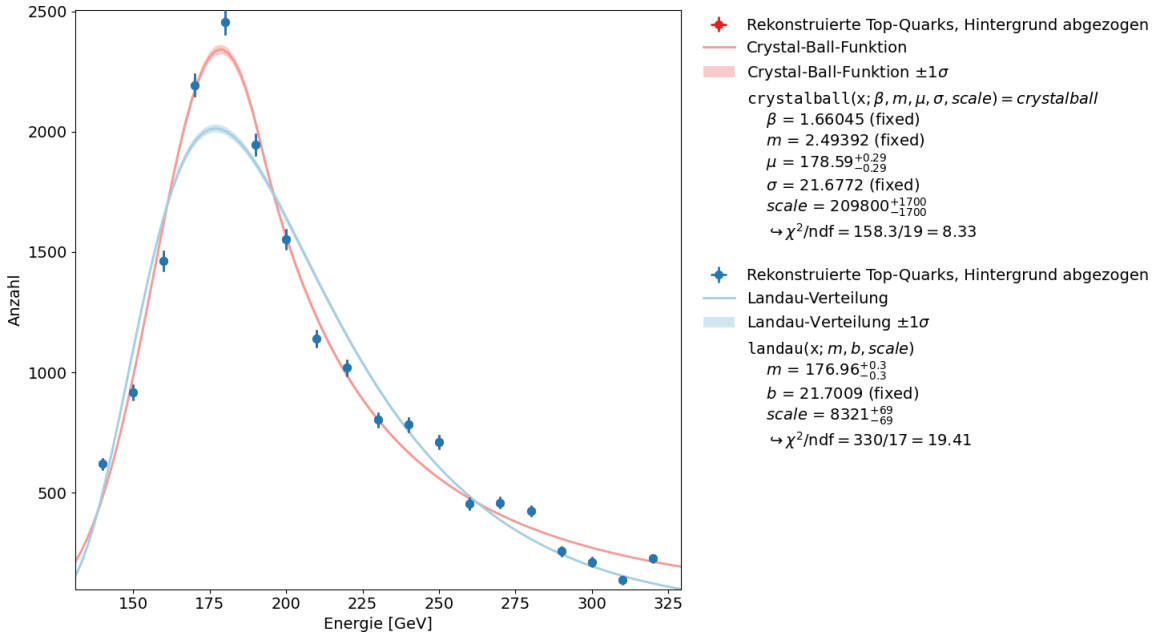


Abbildung 4.4.: Vergleich der Fits der um den Hintergrund bereinigten Daten zwischen Landau-Verteilung und Crystal-Ball-Funktion.

Die Konturen zu den beiden Anpassungen in Abbildung 4.5 und Abbildung 4.6 zeigen, dass die $\mathcal{X}^2$-Funktion für die Anpassungen an die Landau-Verteilung in allen Parametern parabolisch ist. Dies ist bei den Parametern der Crystal-Ball-Funktion nur für σ, bzw. auch für μ näherungsweise gegeben. Bei der Anpassung an die gemessenen Daten tritt dies nicht mehr auf, wie in Abbildung 4.7 veranschaulicht wird. Für den weiteren Verlauf werden deshalb weiterhin nur die symmetrischen Fehler verwendet.

Auf das Ergebnis der Top-Quark-Masse sind die Auswirkung aber eher gering, wie Tabelle 4.1 zeigt.

Tabelle 4.1.: Tabellarischer Vergleich der Top-Quark-Massen aus Crystal-Ball- und Landau-Parametrisierung. Es sind lediglich die statistische Unsicherheiten angegeben.

	Crystal-Ball-Funktion	Landau-Verteilung
Top-Masse	$(174{,}393 \pm 0{,}598)\,\mathrm{GeV}$	$(174{,}229 \pm 0{,}394)\,\mathrm{GeV}$

Tatsächlich liegt der Wert aus der Landau-Verteilung sogar näher an dem Durchschnittswert der PDG von $(173{,}1 \pm 2{,}1)\,\mathrm{GeV}$ [Col11]. Die größere Anzahl an Parametern führt auch zu einer um etwa 52% vergrößerten statistischen Unsicherheit. Diese trägt im Vergleich zur systematischen Unsicherheit der Jet Energy Scale (JES) mit $2{,}4\,\mathrm{GeV}$ [Col11] aber nur wenig zur Gesamtunsicherheit bei, so dass sie eher weniger ins Gewicht fällt.

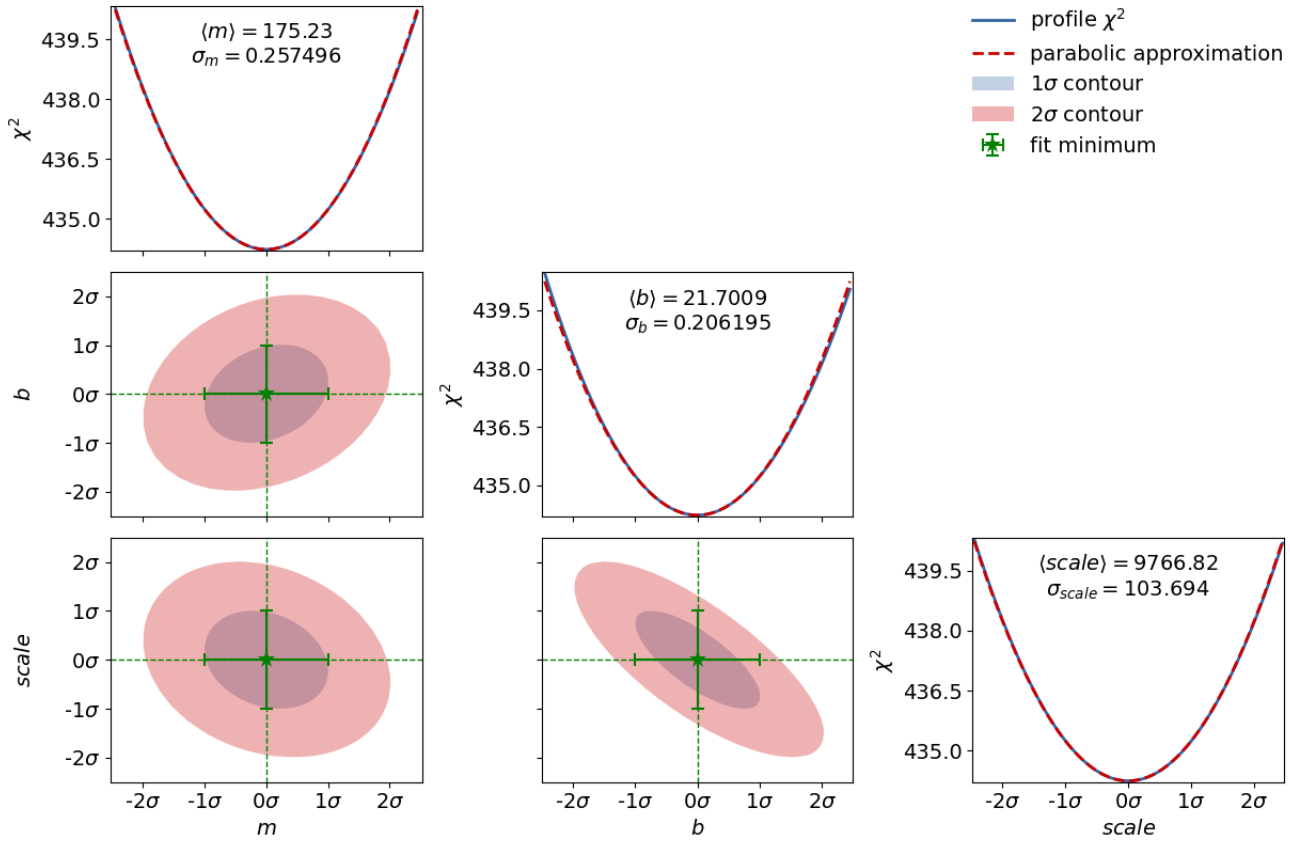


Abbildung 4.5.: Konturen zur Anpassung der Monte-Carlo-Daten an die Landau-Verteilung.

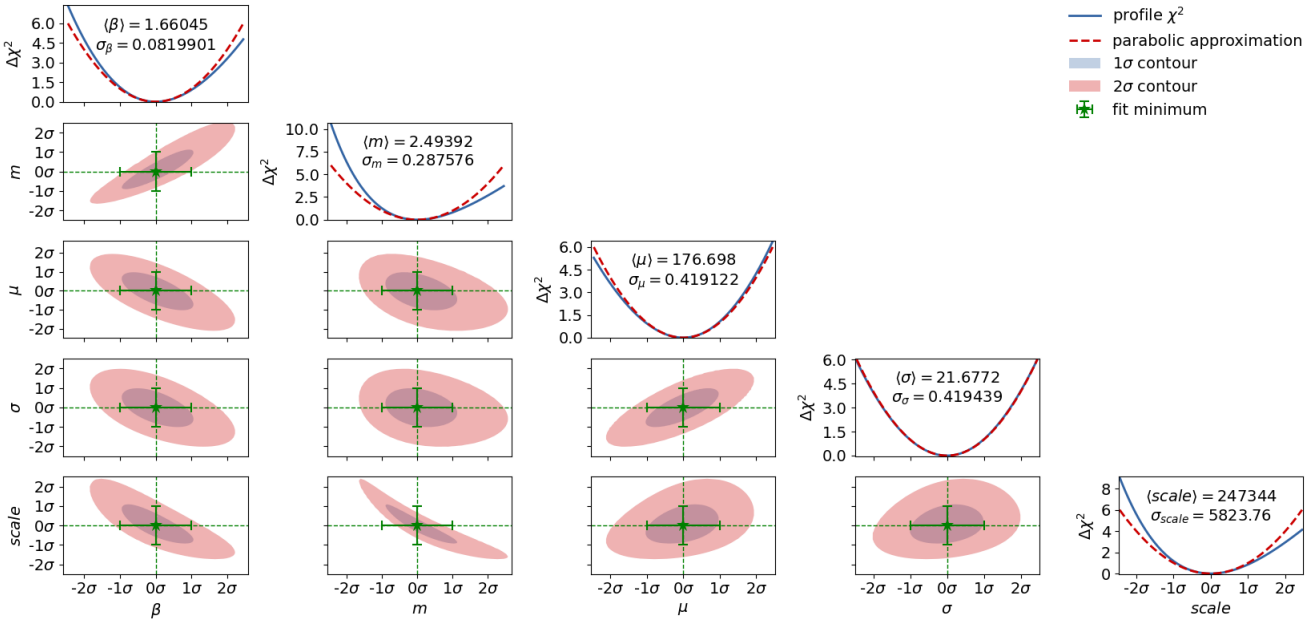


Abbildung 4.6.: Konturen zur Anpassung der Monte-Carlo-Daten an die Crystal-Ball-Funktion.

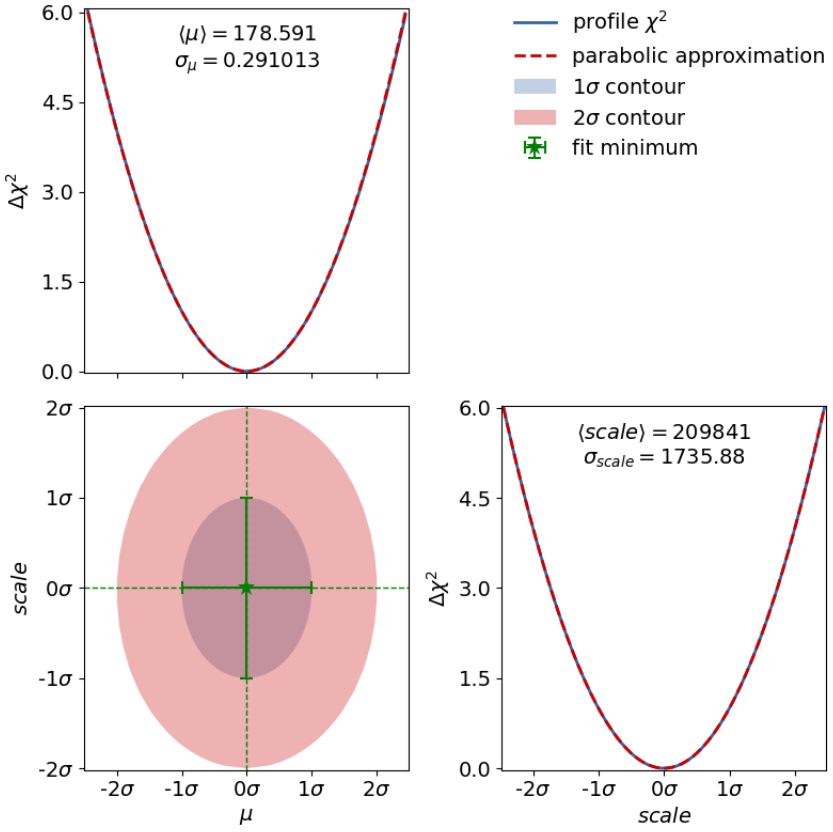


Abbildung 4.7.: Konturen zur Anpassung der gemessenen Daten an die Crystal-Ball-Funktion.

5. Ausblick

5.1. Austeilen, Einsammeln und Teilen von Notebooks mit *jupyterlab-git* und *nbgrader*

Eine bisher nicht diskutierte Frage ist, wie Übungsnotebooks den Studierenden zugänglich gemacht und wie sie nach der Bearbeitung zur Bewertung eingereicht werden können.

Das KIT bietet mit dem Ilias-System [oseLe] zwar eine allgemeine Plattform, diese kann aber nicht alle Vorteile der Notebooks voll ausnutzen. Besonders um die Ergebnisse bearbeiteter Übungen reproduzieren zu können, müssen Notebooks von den Tutoren aus *Ilias* heruntergeladen und in *EnJoY* wieder hochgeladen werden. Auch die Studenten müssen die Notebooks immer zwischenspeichern, um sie dann auf der jeweiligen Plattform zur Bearbeitung oder Abgabe wieder hochzuladen.

Eine einfachere Lösung stellt *jupyterlab-git* dar. Es bietet eine grafische Oberfläche in *JupyterLab* für die Verwaltung von *git*-Verzeichnissen [Jupb]. Der Vorteil dieses Ansatzes liegt in der einfachen Umsetzbarkeit, da von der Universität bereits ein *gitlab*-Server betrieben wird auf den alle Studierenden zugreifen können. Durch einen Link kann ihnen Zugriff auf die entsprechenden Inhalte gewährt werden. Auch kann die Erweiterung *jupyterlab-git* ohne Probleme in den Worker-Container installiert werden.
Dieses System erfordert allerdings den Studierenden die Grundlagen von *git* näherzubringen. Ebenso bietet dieser Ansatz für die Abgabe der bearbeiteten Übungen keine einfache Lösung.

Mit *nbgrader* [JBB+19] existiert bereits Software, die gezielt für den Lehrbetrieb mit Jupyter entwickelt wurde. Sie wurde ursprünglich als Erweiterung für die *Jupyter-Notebook* Software erstellt, mit dem Ziel diese mit zusätzlichen Funktionen für die Lehre auszustatten, ist aber inzwischen auch mit *JupyterHub* und *JupyterLab* nutzbar. Neben einer Infrastruktur zum Verteilen und Einsammeln von Notebooks bietet sie auch Funktionen um automatische Überprüfungen für bearbeitete Zellen durchzuführen und Aufgaben mit Punkten zu bewerten. Die Ergebnisse werden in einer Datenbank gespeichert und können für eine Übersicht der Gesamtpunkte aufgerufen werden. Im Folgenden soll auf Grundlage von Dokumentation zu nbgrader [Teab] ein Umriss der Funktionsweise von *nbgrader* gegeben werden.

Nutzer werden von *nbgrader* in zwei Gruppen unterteilt: *students* und *instructors*. *instructors* können Übungen erstellen und bewerten, *students* erhalten diese, bearbeiten sie und reichen ihre Lösungen ein. Eine solche läuft dann in fünf Schritten ab:

1. *source*
2. *release*
3. *submitted*
4. *autograded*
5. *feedback*

In *source* wird eine Rohfassung des Übungsnotebooks erstellt. Eine besondere Rolle kommen dabei zusätzlichen Klassifikationen von Zellen zu. Über sie wird die Bewertung gesteuert und verhindert, dass bestimmte Inhalte durch die Studierenden geändert werden können. Dabei gibt es folgende Zell-Arten:

- *Read-only*-Zellen können von den Studierenden nicht bearbeitet werden und sind für Erklärungen, Informationen und Aufgabenstellungen gedacht. Auch Code-Zellen mit vorgegebenen Code wie Importen können damit gesperrt werden.
- Mit *manually graded*-Zellen sind Zellen markiert deren Bewertung nicht automatisch erfolgen kann. Für sie muss später manuell eine Punktzahl vergeben werden.
- Mit *autograded answer* werden Code-Zellen markiert, die automatisch bewertet werden sollen. Dazu wird mit *autograded-test* ein Test und erwartete Ausgaben definiert, den der Code der Studierenden durchläuft. So kann z. B. eine Funktion durch die Studierenden geschrieben werden, die ein Argument quadriert zurück gibt. Der Test dazu spielt nun einige Zahlen durch und gleicht die Ausgaben mit den erwarteten Ergebnissen ab.

Bevor die Studierenden die Notebooks erhalten, werden sie im *release*-Schritt in eine Studentenversion umgewandelt. Dadurch werden *Read-only*-Zellen zur Bearbeitung geblockt, die autograded test-Zellen versteckt, so dass sie nicht mehr eingesehen werden können. Diese Version wird dann an die Studierenden verteilt. Nach der Bearbeitung können die Studierenden ihre Ergebnisse im *submit*-Schritt einreichen. Ihre Lösungs-Notebooks werden an einem vom Übungsleiter definierten Ort gespeichert und die automatische Bewertung wird durchgeführt. Zusätzlich müssen die manuellen Bewertungen hinzugefügt werden und es besteht weiter die Möglichkeit die automatischen Bewertungen zu überschreiben, sollten sie der Lösung der Studierenden nicht gerecht werden. Die Punktzahlen werden dann in der Datenbank abgespeichert. Zuletzt kann im *feedback*-Schritt den Studierenden ein Feedback zu ihrer Lösung generiert werden. Dieses enthält die Ergebnisse der automatischen und manuellen Bewertung.

Im Verlauf dieser Arbeit wurde versucht *nbgrader* in *EnJoY* einzubinden. Dies ist aber vorerst an mehreren Faktoren gescheitert. So erschwerte die Kombination von *JupyterHub* mit *Docker*-Containern die Handhabung der einzelne Komponenten von *nbgrader*. Es muss sowohl eine Installation von *nbgrader* auf den *Worker-Containern*, als auch auf dem *JupyterHub* vorhanden sein und diese müssen miteinander kommunizieren können. Dies einzurichten ist bisher nicht gelungen.

Ein zweites Problem stellt die Notwenigkeit eines zentraler Speicherorts für die Notebooks dar. Für diesen müssen alle Nutzer auch über Lese- und Schreibrechte verfügen. Das stellt ein Sicherheitsrisiko dar. Ob eine Umgehung dieser Probleme möglich ist, kann nicht abschließend geklärt werden. Eine Erweiterung von *EnJoY* um *nbgrader* soll aber weiter verfolgt werden.

5.2. Einsatz von Notebooks Abseits von Übungen

Ein offensichtlicher Einsatzbereich für Notebooks im Physikstudium stellen Protokolle der Laborpraktikas dar. In ihnen werden die Ergebnisse der durchgeführten Experimente dargestellt, analysiert und interpretiert, was genau den Stärken der Notebooks entspricht. Mit EnJoY steht den Studierenden nun eine Plattform zur Verfügung auf der diese Protokolle so erstellt werden können, dass sie von den Betreuern leicht nachvollziehbar sind. Kontrollen der verwendeten Methoden sind dann einfach durchzuführen, womit geschönte Ergebnisse erkannt werden können und gezielteres Feedback an die Studierenden möglich ist. Da Protokolle meist zu zweit erstellt werden, liefert *git* eine effektive Möglichkeit gemeinsam an einem Dokument zu arbeiten.

Neben all den Vorteilen sei an dieser Stelle auch auf ein paar Herausforderungen beim Erstellen von Protokollen mit Notebooks hingewiesen. Da sie sich in ihrer Form stark von den bisherigen LaTeX-Dokumenten unterscheiden, fehlen allgemein akzeptierte Konventionen zur Gestaltung und Darstellung. Ein Aspekt ist hier, dass Zahlen als Ergebnisse unter der Zelle entweder in unsauberer Form dargestellt werden, oder sich Textdoppelungen ergeben, wenn sie z. B. mittels `print`-Befehlen formatiert werden. Auch werden die Code-Zellen manche Befehlen enthalten, die nur der Darstellung von z. B. Graphen dienen und für die eigentliche Analyse irrelevant sind. Hier werden sich aber vermutlich Konventionen etablieren, die eine sinnvolle Präsentation von Daten, Analyse und Ergebnissen ermöglichen.

Eher ungeeignet erscheinen Notebooks für das Erlernen von kompilierten Sprachen wie *C++*. Zwar existieren auch hierfür Kernel, sie spiegeln aber nicht die eigentliche Verwendung der Programmiersprachen wieder und sind aus didaktischer Sicht nicht sinnvoll. Gerade Anfängern im Programmieren kann dadurch ein falsches Bild gezeigt werden, wie insbesondere umfangreichere Programme erstellt werden. Da *EnJoY* in den *JupyterLab*-Umgebungen auch einen Terminal bietet, kann trotzdem der Vorteil der virtuellen Computer-Plätzen genutzt werden. In entsprechend ausgestatteten Containern können die Programme geschrieben, im Terminal kompiliert und ausgeführt werden.

6. Fazit

Notebooks können also ein vielseitiges Werkzeug im Physikstudium sein. Wie mit dem Beispiel zur Bestimmung der Top-Quark-Masse gezeigt,können Notebooks gut für Übungen eingesetzt werden. In diesem Zuge wurde mit der Crystal-Ball-Funktion die Analyse deutlich verbessert und gezeigt, dass auch bestehende Übungen leicht als Notebooks umgesetzt werden können.
Durch die weite Verbreitung von *Jupyter* und *Python*, insbesondere im Bereich der Datenanalyse, erlernen Studierende nicht nur wichtige Fähigkeiten, die in der Physik relevant sind, sondern auch Werkzeuge, die in vielen Betätigungsfeldern außerhalb der Physik angewendet werden.

Mit *EnJoY* steht eine entsprechende Infrastruktur zur Verfügung, die Notebooks als Cloudservice bereitstellt und durch virtuelle Umgebungen einheitliche Softwareversionen bietet. Nicht nur stehen deutlich mehr Computer-Plätze als bisher durch den Computer-Poolraum zu Verfügung, die Verwendung ist auch orts- und hardewareunabhängig. Unter anderem bedingt durch die Schutzmaßnahmen im Zuge der Covid-19-Pandemie wird *EnJoY* bereits jetzt für die Vorlesungen „Moderne Methoden der Datenanalyse“und „Computergestütze Datenauswertung“eingesetzt.

Neben der demonstrierten Nutzung für Übungen sind weitere Einsatzfelder möglich. Auch für Protokolle im Rahmen der Laborpraktika stellen die Notebooks eine neue Protokollform dar. Mit der Installation zusätzlicher Erweiterungen und der Nutzung des Terminals in *Jupyterlab* können Container auch für Zwecke jenseits von Notebooks verwendet werden.

Anhang

A. Dokumentation von EnJoY

Übersicht

EnJoY ist eine Implementierung des *JupyterHubs* für die Lehre an der Fakultät für Physik des KITs. Es nutzt *Docker*-Container als virtuelle Umgebungen für *JupyterLab* bereit und Nutzt die bestehnde Infrastuktur des Poolraumes zur Authentifizierung und zum Speichern der Daten der Nutzer. Mittels einer CI/CD-Pipeline können Updates der Container oder neue Container im laufenden Betrieb eingespielt werden.

Jeder Dienst von *EnJoY* ist als *Docker*-Container aufgebaut und über ein *Dockerfile* definiert. Das *Gitlab*-Repository ist unter https://test-gitlab.etp.kit.edu/eppelt/enjoy zu finden.

Installation

Um *EnJoY* auf einem Server zu installieren sind folgende Schritte notwendig:

1. Installieren von *docker* und *gitlab-runner*.
2. Registrieren eines *Docker-Runners* im *gitlab*-Repository mit dem Tag `enjoy_deploy_machine`. Im Anschluss muss noch in der Konfigurationsdatei des Runners die `docker.sock`-Datei eingebunden werden.
3. Wahl eines Passwortes für die Datenbank und Speichern.
4. Klonen des Repositorys (Branch `machine`) und bauen der *Images*. Alternativ können diese auch aus der Registry geladen werden. Lediglich das Image für *jupyter-db* muss lokal gebaut werden.
5. Herunterladen der `docker-compose.yml`
6. Speichern des Zertifikats und Keys im gleichen Ordner unter `host.crt`, bzw. `host.key`
7. Generieren eines API-Schlüssels für den Jupyterhub und speichern als `jupyter_api.token`
8. Starten mit `docker stack deploy -compose-file docker-compose.yml jupyterhub`

Erstellen/Bearbeiten von Workern

1. Erstellen eines neuen Verzeichnisses in `course-container`. Der Name des Verzeichnisses muss mit `jupyter-worker-` beginnen.
2. Erstellen eines Dockerfiles in diesem Ordner. Es muss von abgeleitet sein und als letzte Zeile `USER $NBUSER` eingetragen haben.
3. Um bestehende Worker zu bearbeiten, müssen die Dockerfiles geändert werden.
4. Die CI/CD-Pipeline muss nach dem Commit der Änderungen in beiden Fällen manuell gestartet werden. Dabei muss über die Variable `deploy_tag` angegeben werden, auf welcher Instanz die Änderungen eingespielt werden sollen.

Technische Übersicht

jupyterhub

Um die Laufzeiten der CI/CD-Pipeline zu verkürzen, wird der Container des *JupyterHubs* in zwei Schritten gebaut. Der `jupyter-hub-prebuild` bildet den eigentlichen *JupyterHub*, mit dem zweiten Bauschritt,der im Zuge der CI/CD-Pipeline ausgeführt wird, wird die `whitelist.txt` dmit *Worker-Images* eingefügt. Die `jupyter-hub_config.py` wird bereits im ersten Schritt eingebunden.

Die Konfigurationsdatei nutzt eine eigene *Spawner*-Klasse: `EnjoyDockerSpawner`. Sie erbt von `SystemUserSpawner` und erweitert ihn so, dass die Verzeichnisse der Nutzer mit `root_squash` eingebunden werden.

jupyter-db

Dieser Container enthält eine *PostgresDB*-Datenbank. Sie enthält die ausgelagerte Datenbank des *Jupyterhubs*. Sie ist wird durch ein Passwort geschützt, dass vor dem erstmaligen bauen definiert werden muss. Als persistenter Speicher kommt ein *Volume* zum Einsatz.

jupyter-cull

Der Cull-Service beendet Notebooks, die länger als 600 Sekunden inaktiv waren. Er ist analog zu dem Beispiel des JupyterHub-Git-Repository[1] aufgebaut.

jupyter-worker-base

Das ist das Basis-*Image* für alle *Worker*. Bereits installiert sind *JupyterLab* mit einem *Python*- und *JupyROOT*-Kernel.

nginx-proxy

Der Proxy wird auch als eigener Container realisiert. Dieser basiert auf *CentOS 7.6.1810* und nutzt die *nginx* Software.

CI/CD-Pipeline

Die Pipeline ist in vier Abschnitte (*stages*) untergliedert: der Vorbereitung (`pre`), das Bauen der Images (`build_stack` und `build_container`) und das einführen der Änderungen im laufenden System (`deploy`). Die `build`-Aufgaben sowie die Vorbereitung werden auf einer eigenen *gitlab-runner*-VM des ETPs ausgeführt [2]. Auch wird ein Container erstellt, in dem die weiteren *stages* ausgeführt werden. Dieser ist von `docker:18.09.7-dind` abgeleitet und um die Software *grep* und *bash* erweitert. Der entsprechende *Runner* nimmt nur Jobs mit dem *tag* `enjoy_builder` an.

In der `pre`-*stage* wird die `whitelist.txt` mit den Namen der Container erstellt und als *artifact* für die folgende *stages* gespeichert. Die Container für die Services, sowie für die *Worker* werden in den `build`-*stages* gebaut und in der Registry gespeichert. Während der `deploy`-*stage* werden diese dann auf der laufenden Instanz eingspielt und die aktuell laufenden Container (außer den *Workern* neu gestartet. Über die Variable `CI_COMMIT_BRANCH` wird dabei gesteuert ob dies für die Test-Instanz von *EnJoY* (`test`) oder die eigentliche Instanz (`machine`) durchgeführt wird.

[1] https://github.com/jupyterhub/jupyterhub/tree/master/examples/cull-idle
[2] gitlab-runner.etp.kit.edu

Abbildungsverzeichnis

B. Notebook zur Bestimmung der Top-Quark-Masse

1 Determing the mass of the top quark

by Mathias Schnepf and Jonas Eppelt Version from 3.4.20 ## Introduction In the following exercises, We will study top quarks produced at the LHC. The used data are collected with th CSM detector. Using the now deprecated KITA-Framework, the for this excercise necessary data was extracted from `.root`files into `.csv` files. Top quarks are the heaviest quarks know until now- At the LHC, top quarks are mostly produced in pairs of quark and anti-quarks. These quarks decay under weak interaction almost exclusively into a W-boson and a b-quark. W-bosons decay in about $\frac{2}{3}$ of all cases hadronically in two quarks and in about $\frac{1}{3}$ leptonically in a charged lepton and the corresponding neutrino. Combining these decays, there are 3 possibilities for a $t\bar{t}$ pair to decay:

- allhadronic: both W-bosons decay into quarks
- dileptonic: both W-bosons decay into lepton an d neutrino
- semileptonic: one W-boson decays into quarks and the other decays into lepton and neutrino

In this exercise, we will study the semileptonic channel, more precisely the μ+jets channel. This channel has the advantage of a high branching ratio compared to the dileptonic channel and a small background contribution compared to the allhadronic channel. The files (simulated monte-carlo and real data events) are already preselected by cutting on the transverse momentum (p_T) and the pseudorapidity (η) of jets and the muon. The used cut-values are typical values for the $t\bar{t}$-analyses at CMS. In addition, the events are required to have at least two jets and exactly one muon fulfilling the p_T and η requirements. This analysis will use the `pandas` package, explicitly the `DataFrame` object. For plotting `matplotlib` will be used. The fits will use `kafe2` and `Iminuit`.

1.1 Import and functions

[1]:
```
import pandas as pd
import numpy as np
from iminuit import Minuit
import matplotlib
import matplotlib.pyplot as plt
import scipy as scp
import kafe2
matplotlib.rcParams["patch.linewidth"] = 1
import ast
from itertools import combinations
import warnings
warnings.filterwarnings('ignore')
%matplotlib widget
```

```
Welcome to JupyROOT 6.18/04
```

[2]:
```
# importing MC simulations for ttbar events
conv = {0:lambda x: np.array(ast.literal_eval(x)),
        1:lambda x: np.array(ast.literal_eval(x)),
        2:lambda x: np.array(ast.literal_eval(x)),
```

```
import warnings
warnings.filterwarnings('ignore')
%matplotlib widget
```

```
Welcome to JupyROOT 6.18/04
```

[2]:
```
# importing MC simulations for ttbar events
conv = {0:lambda x: np.array(ast.literal_eval(x)),
        1:lambda x: np.array(ast.literal_eval(x)),
        2:lambda x: np.array(ast.literal_eval(x)),
        3:lambda x: np.array(ast.literal_eval(x)),
        4:lambda x: np.array(ast.literal_eval(x)),
        5:lambda x: np.array(ast.literal_eval(x))}
mc_ttbar = pd.read_csv("csv/MC_TTBar_NJets2.
 ↪csv",sep="\t",index_col=False,converters=conv);

# importing MC simulations for W+Jets events
conv = {0:lambda x: np.array(ast.literal_eval(x)),
        1:lambda x: np.array(ast.literal_eval(x)),
        2:lambda x: np.array(ast.literal_eval(x)),
        3:lambda x: np.array(ast.literal_eval(x))}
mc_wjets = pd.read_csv("csv/MC_WJets_NJets2.
 ↪csv",sep="\t",index_col=False,converters=conv);

# importing the measurment data
conv = {0:lambda x: np.array(ast.literal_eval(x)),
        1:lambda x: np.array(ast.literal_eval(x)),
        2:lambda x: np.array(ast.literal_eval(x)),
        3:lambda x: np.array(ast.literal_eval(x))}
data_ttbar = pd.read_csv("csv/data_NJets4.
 ↪csv",sep="\t",index_col=False,converters=conv);
```

[3]:
```
# function to calculate z-Component of the neutrino momentum
def calcNuPz(lep,E):
    pz1=0
    pz2=0
    mW=80.4
    mu = mW**2/2+lep[1]*E[1]+lep[2]*E[2]
    a = mu*lep[3]/pt(lep)**2
    a2 = a**2
    b = (lep[0]**2*pt(E)**2 - mu**2)/pt(lep)**2
    if a2<b: return a
    else:
        pz1 = a+np.sqrt(a2-b)
        pz2 = a-np.sqrt(a2-b)
        if abs(pz1)<= abs(pz2): return pz1
        else: return pz2
```

[4]:
```
# landau funktion for fitting
latex = "Crystalball"

def crystalball(x, beta, m, mu, sigma, scale):
    return  scp.stats.crystalball.pdf(x = -x,beta = 1/beta ,m = m,loc = -mu,␣
 ↪scale = sigma)*scale
```

1.2 1. Preparation

1.2.1 A look at the datastructure

Have a look at the way the data is stored here. We use pandas DataFrame object to store the data. First look at the Data by using the `df.head` command. For a deeper understanding you can visit the documentation of pandas' DataFrame. In general the date is organized in events. In each event, you can find a number of Jets, a Muon and the missing energy in each component.

[5]:
```
# Monte Carlo for ttbar events
mc_ttbar.head()
```

[5]:
```
                                                Jets  \
0  [[259.87, -168.906, 95.3889, -170.715], [382.1…
1  [[133.5, -2.6198, -88.2144, -97.5182], [68.670…
2  [[388.832, -43.4641, 183.719, -338.966], [84.7…
3  [[245.86, 43.5794, -41.463, -238.077], [55.574…
4  [[203.689, -93.2395, -85.7243, -158.787], [266…

                           missing E                                  Muons  \
0   [52.3892, 39.8444, -34.0154, 0.0]   [73.0453, 44.3549, 5.92641, -57.7332]
1   [20.8802, -1.22342, 20.8443, 0.0]    [151.52, 47.3366, 55.8079, -132.677]
2     [162.236, 126.33, -101.79, 0.0]  [160.232, 39.8741, -89.3442, -126.893]
3    [61.7343, 18.1577, 59.0036, 0.0]   [39.7153, -19.6615, 32.694, -11.0375]
4  [23.1656, -13.3561, -18.9277, 0.0]    [102.962, 46.192, -33.1105, 85.8554]

                                   T_gen  \
0  [913.127, 182.728, -8.88419, -877.793]
1   [470.758, 36.0134, 49.8958, -432.965]
2  [529.629, 130.635, -225.293, -427.565]
3   [204.964, 7.6995, -52.5302, -83.5924]
4   [270.953, -111.918, -105.3, -139.982]

                                anti T_gen  \
0     [456.008, -266.875, 61.67, -314.706]
1  [325.873, -29.1192, -45.2743, -270.622]
2    [654.323, -108.139, 227.838, -577.32]
3     [381.276, 38.5479, 46.5952, -334.59]
4    [259.299, 11.7287, -38.9577, 189.265]
```

```
                                  Jets bTag
0  [0.145328, 0.296399, 0.0325011, 0.14819]
1            [0.278641, 0.810474, 0.177331]
2          [0.0675631, 0.999998, 0.239348]
3            [0.943737, 0.765707, 0.376336]
4   [0.99838, 0.0861655, 0.89793, 0.662582]
```

[6]:
```
# Monte Carlo for W-Jet events
mc_wjets.head()
```

[6]:
```
                                                Jets  \
0  [[108.857, -52.5515, 2.04588, 94.0711], [183.4…
1  [[46.1862, -38.5572, -14.8651, 19.3557], [93.9…
2  [[186.467, -33.7477, -56.6337, -173.84], [48.0…
3  [[163.18, 91.6156, 9.55574, 134.192], [109.703…
4  [[110.863, -31.5028, 42.8955, -96.8073], [112…

                           missing E                                  Muons  \
0    [23.0081, 16.7103, 15.8158, 0.0]  [34.6515, -12.3001, -24.3221, 21.3976]
1    [49.0855, 42.5448, 24.4812, 0.0]  [65.6198, -33.3948, -4.83266, 56.2795]
2   [50.9159, 45.0782, -23.6725, 0.0]    [79.4006, 17.4953, 47.513, -61.1627]
3    [24.8113, 24.605, -3.19299, 0.0]  [333.093, -121.942, -57.8864, 304.517]
4  [32.0489, -7.66629, -31.1185, 0.0]    [77.6548, 38.9837, 16.1485, 65.1901]

                Jets bTag
0   [0.135533, 0.288028]
1   [0.109713, 0.144369]
2  [0.0178308, 0.114022]
3  [0.212426, 0.0306883]
4  [0.106201, 0.0992101]
```

[7]:
```
# data events
data_ttbar.head()
```

[7]:
```
                                                Jets  \
0  [[86.9846, -22.707, 83.1999, -1.97902], [120.9…
1  [[180.578, -176.045, -22.7944, -30.4881], [77…
2  [[105.173, 50.5937, 51.1537, 75.9833], [187.92…
3  [[106.84, 38.2244, 51.1949, -85.1773], [61.526…
4  [[200.9, 72.7218, -19.1113, -185.576], [68.922…

                           missing E                                  Muons  \
0  [25.4749, -14.7394, -20.7779, 0.0]  [61.9502, 48.7334, -22.4358, -30.9759]
1   [178.673, 172.431, -46.8132, 0.0]    [98.3057, 46.3269, 30.2756, 81.2478]
2    [54.3838, 50.2817, 20.7207, 0.0]    [45.7488, -13.451, -43.531, 4.13127]
3   [42.6683, 17.7394, -38.8059, 0.0]  [62.2807, -49.1917, 5.98482, -37.7257]
```

```
4    [68.4526, 19.1516, 65.7189, 0.0]    [63.802, -48.4505, 41.503, -0.85528]

                                    Jets bTag
0     [0.894873, 0.99986, 0.103706, 0.114142]
1  [0.0487902, 0.0724623, 0.977469, 0.248965]
2      [0.982713, 0.37588, 0.125672, 0.11425]
3      [0.109023, 0.4085, 0.929117, 0.195504]
4    [0.993468, 0.588514, 0.518006, 0.135728]
```

As you can see, the structure in all DataFrames is roughly the same. In the columns "missing E" and "Muons" there are numpy arrays stored, representing each a 4-vector $(E, c \cdot \vec{p})$. In "Jets" there is an 2-dimensional numpy array, holding a number of Jets, each characterized by a 4-vector. The "Jets bTag" column holds a numpy array, where each entry corresponds to the B-Tag value for one Jet: the first entry corresponds to the first Jet, and so on.

1.2.2 Usefull commands

Command	description
`df.head()`	shows first entries of a DataFrame `df`
`df["field name"] = df.apply(lambda x: function(x),axis=1)`	adds a new column with values returned by function(x); x is a row
`dataframe = df.loc[compare(df["field name"])]`	selects rows where `compare()` returns `True`
`miu = Minuit(f, par=1)`	initialize the IMinuit minimizer
`miu.migrad()`	run optimizer
`miu.values`	returns list of parameter values
`combinations(x,i)`	generates all combinations of `i` elements from `x`
`np.clip(a,min,max)`	limits values in an iterable: values $>$ `max` are set to `max`, analog for `min`

1.2.3 Helper functions

First start with writing some functions, that will be needed in to following exercises. Implement functions to calculate the following values from 4-vectors: - transverse momentum - invariant mass - pseudorapidity

Note that at CMS the z-axis points in the direction of the ray's axis.

[8]:
```python
# calculate pt
def pt(four_vec):
    return np.sqrt((four_vec[1:3]**2).sum())
```

[9]:
```
# function to calculate pt for a unknown number of jets
def pt_jets(jets):
    res = np.zeros(len(jets))
    for i,j in enumerate(jets):
        res[i]=pt(j)
    return res
```

[10]:
```
# calculate pseudorapidity eta
def p(four_vec):
    return np.sqrt((four_vec[1:4]**2).sum())

def eta(four_vec):
    return np.arctanh(four_vec[3]/p(four_vec))
```

[11]:
```
# calculate invariant mass
def m(four_vec):
    return np.sqrt(four_vec[0]**2-(four_vec[1:4]**2).sum())
```

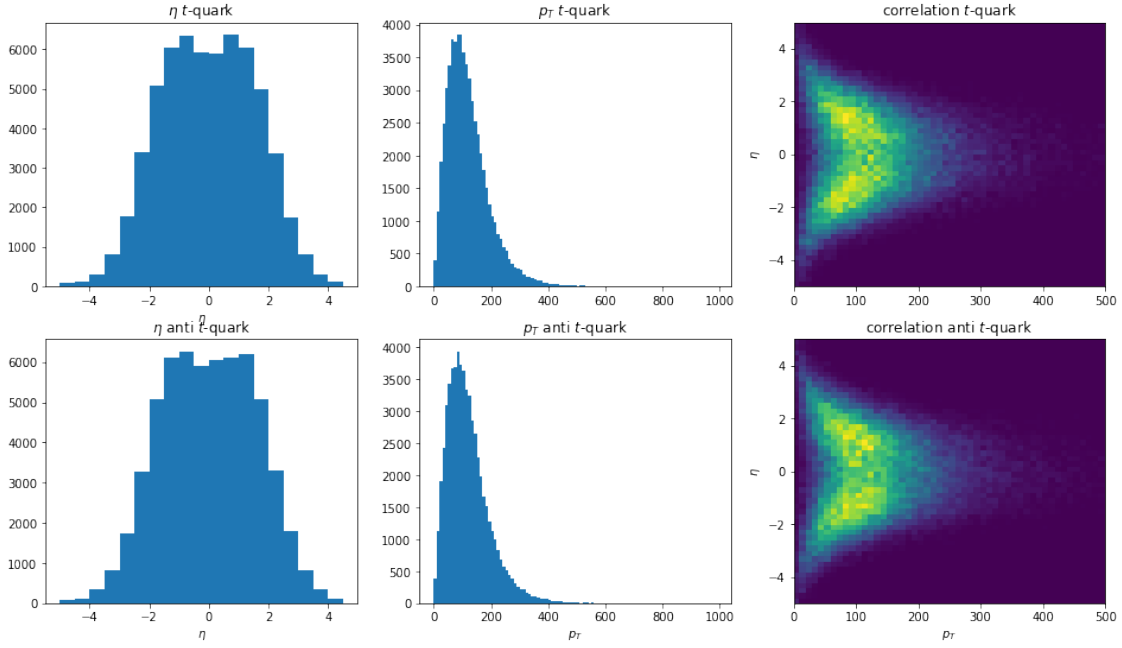
1.3 2. Kinematics of $t\bar{t}$ events

Have a look at simulated $t\bar{t}$ events (`mc_ttbar`). Plot the distributions of p_T and η of top quarks and top anti-quarks. Do you see why $t\bar{t}$ events are called high-p_T events? Also have a look at the correlation of p_T and η.

[12]:
```
# calculate pt and eta
mc_ttbar["Pt Tgen"] = mc_ttbar.apply(lambda x: pt(x["T_gen"]),axis=1)
mc_ttbar["Pt ATgen"] = mc_ttbar.apply(lambda x: pt(x["anti T_gen"]),axis=1)
mc_ttbar["eta Tgen"] = mc_ttbar.apply(lambda x: eta(x["T_gen"]),axis=1)
mc_ttbar["eta ATgen"] = mc_ttbar.apply(lambda x: eta(x["anti T_gen"]),axis=1)
```

[13]:
```
#plot the results
matplotlib.rcParams['figure.figsize'] = [16,9]
fig, axes = plt.subplots(2,3)
axes[0,0].hist(np.clip(mc_ttbar["eta Tgen"],-5,5),bins = np.arange(-5,5,0.5));
axes[0,0].set_xlabel("$\eta$")
axes[0,0].set_title("$\eta$ $t$-quark")

axes[1,0].hist(np.clip(mc_ttbar["eta ATgen"],-5,5),bins = np.arange(-5,5,0.5))
axes[1,0].set_title("$\eta$ anti $t$-quark");
axes[1,0].set_xlabel("$\eta$")

axes[0,1].hist(np.clip(mc_ttbar["Pt Tgen"],0,1000),bins = np.arange(0,1000,10))
axes[0,1].set_title("$p_T$ $t$-quark");

axes[1,1].hist(np.clip(mc_ttbar["Pt ATgen"],0,1000),bins = np.arange(0,1000,10))
axes[1,1].set_title("$p_T$ anti $t$-quark");
```

```
axes[1,1].set_xlabel("$p_T$")

axes[0,2].hist2d(mc_ttbar["Pt Tgen"],mc_ttbar["eta Tgen"],bins =␣
↪50,range=[[0,500],[-5,5]])
axes[0,2].set_title("correlation $t$-quark")
axes[0,2].set_ylabel("$\eta$")

axes[1,2].hist2d(mc_ttbar["Pt ATgen"],mc_ttbar["eta ATgen"],bins =␣
↪50,range=[[0,500],[-5,5]])
axes[1,2].set_title("correlation anti $t$-quark")
axes[1,2].set_xlabel("$p_T$")
axes[1,2].set_ylabel("$\eta$")
fig.show();
```



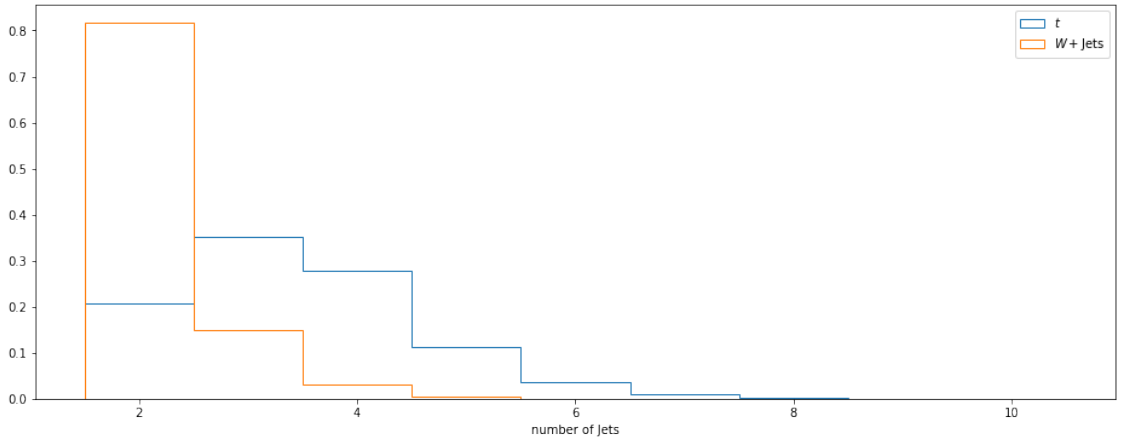
[14]:
```
matplotlib.rcParams['figure.figsize'] = [16,6]
```

1.4 3. Number of Jets

How many jets would you expect in a semileptonically decaying $t\bar{t}$ event? Plot the number of jets for the simulated $t\bar{t}$ events and for the simulated W+jets background. When studying $t\bar{t}$ events there are many more backgrounds to consider. For this exercise we use only the contribution for W+jets, because this is the most important background. Which cut on the number of jets would you use in order to reject W+jets background and keep $t\bar{t}$ events?

[15]:
```
#calculate the number of Jets
mc_wjets['Jets_N'] = mc_wjets.apply(lambda x: len(x['Jets']),axis=1)
mc_ttbar['Jets_N'] = mc_ttbar.apply(lambda x: len(x['Jets']),axis=1)
```

[16]:
```
# plot the number of Jets

plt.hist(mc_ttbar['Jets_N'] ,label = "$t\bar{t}$",density=1, histtype = 'step',␣
 ↪bins=9, range=[1.5,10.5]);
plt.hist(mc_wjets['Jets_N'],label = "$W+$Jets",density=1, histtype = 'step',␣
 ↪bins=9, range=[1.5,10.5]);

plt.legend()
plt.xlabel("number of Jets")
plt.show()
```



[17]:
```
#cut on the number of Jets
cut_ttbar=mc_ttbar.loc[mc_ttbar['Jets'].str.len()>=4]
cut_wjets=mc_wjets.loc[mc_wjets['Jets'].str.len()>=4]
cut_data=data_ttbar.loc[data_ttbar['Jets'].str.len()>=4]
```

1.5 4. Shape variables

For the following exercises we use the samples with 4 or more jets. Study which event shape variable could be used to discriminate background (W+jets) from signal ($t\bar{t}$). Study the missing transverse energy E_T in an event, the transverse momentum (p_T) of the muon and the pseudorapidity ( ) of the muon. Also look at the sum of all transverse momenta (H_T) and the mass of the 3 jets with the highest transverse momentum (M_3). Which of these variables performs best in distinguishing W+jets background from signal?

First create a function to calculate M_3: go through all combinations of 3 Jets and add them. Keep the invariant mass of the resulting vector with the highest transverse momentum.

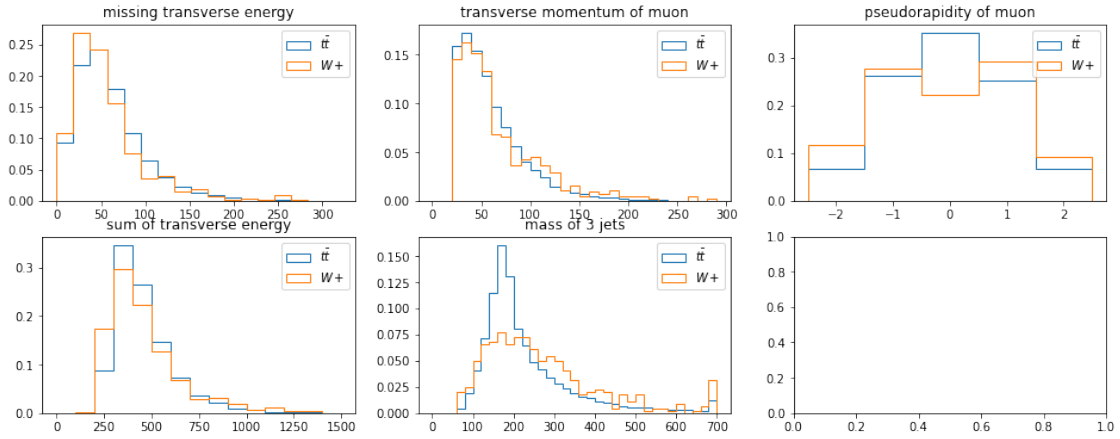
[18]:
```
# m3 function
def m3(x):
    ts = np.zeros(4)
    l = list(combinations(x,3))
    for i in l:
        t = sum(i)
        if pt(t)>pt(ts):
            ts = t
    return m(ts)
```

Now calculate the shape variables (missing E_T, $p_{T,\mu}$, η_μ, H_T and M_3) and plot the results. Make sure your plots are correctly normalized by using the `weights`-keyword, since `density = 1` does not lead to the sum of the bins being equal to 1.

[19]:
```
#calculate shape variables
cut_ttbar['Pt Muon'] = cut_ttbar.apply(lambda x: pt(x['Muons']),axis=1)
cut_wjets['Pt Muon'] = cut_wjets.apply(lambda x: pt(x['Muons']),axis=1)

cut_ttbar['missing Et'] = cut_ttbar.apply(lambda x: pt(x['missing E']),axis=1)
cut_wjets['missing Et'] = cut_wjets.apply(lambda x: pt(x['missing E']),axis=1)

cut_ttbar['eta Muon'] = cut_ttbar.apply(lambda x: eta(x['Muons']),axis=1)
cut_wjets['eta Muon'] = cut_wjets.apply(lambda x: eta(x['Muons']),axis=1)

cut_ttbar['Ht'] = cut_ttbar.apply(lambda x:␣
↪pt(x['Muons'])+sum(pt_jets(x['Jets']))+pt(x['missing E']),axis=1)
cut_wjets['Ht'] = cut_wjets.apply(lambda x:␣
↪pt(x['Muons'])+sum(pt_jets(x['Jets']))+pt(x['missing E']),axis=1)

cut_ttbar['M3'] = cut_ttbar.apply(lambda x: m3(x['Jets']),axis=1);
cut_wjets['M3'] = cut_wjets.apply(lambda x: m3(x['Jets']),axis=1);
```

[20]:
```
#plot results
fig, axes = plt.subplots(2,3)
axes[0,1].set_title("transverse momentum of muon")
axes[0,1].hist(np.clip(cut_ttbar['Pt Muon'],0,300), histtype = 'step', weights␣
↪= np.ones(len(cut_ttbar))/len(cut_ttbar), label = "$t \\bar{t}$", bins=np.
↪arange(0,300,10));
axes[0,1].hist(np.clip(cut_wjets['Pt Muon'],0,300), histtype = 'step', weights␣
↪= np.ones(len(cut_wjets))/len(cut_wjets), label = "$W+$", bins=np.
↪arange(0,300,10));
axes[0,1].legend()

axes[0,2].set_title("pseudorapidity of muon");
axes[0,2].hist(np.clip(cut_ttbar['eta Muon'],-2.5,3.5), histtype = 'step',␣
↪weights = np.ones(len(cut_ttbar))/len(cut_ttbar), label = "$t \\bar{t}$",␣
↪bins=np.arange(-2.5,3.5,1));
```

```
axes[0,2].hist(np.clip(cut_wjets['eta Muon'],-2.5,3.5), histtype = 'step',␣
 ↪weights = np.ones(len(cut_wjets))/len(cut_wjets), label = "$W+$", bins=np.
 ↪arange(-2.5,3.5,1));
axes[0,2].legend()

axes[0,0].set_title("missing transverse energy");
axes[0,0].hist(np.clip(cut_ttbar['missing Et'],0,340), histtype = 'step',␣
 ↪weights = np.ones(len(cut_ttbar))/len(cut_ttbar), label = "$t \\bar{t}$",␣
 ↪bins=np.arange(0,340,340./18));
axes[0,0].hist(np.clip(cut_wjets['missing Et'],0,340), histtype = 'step',␣
 ↪weights = np.ones(len(cut_wjets))/len(cut_wjets), label = "$W+$", bins=np.
 ↪arange(0,340,340./18));
axes[0,0].legend()

axes[1,0].set_title("sum of transverse energy");
axes[1,0].hist(np.clip(cut_ttbar['Ht'],0,1600), histtype = 'step', weights = np.
 ↪ones(len(cut_ttbar))/len(cut_ttbar), label = "$t \\bar{t}$", bins=np.
 ↪arange(0,1600,100));
axes[1,0].hist(np.clip(cut_wjets['Ht'],0,1600), histtype = 'step', weights = np.
 ↪ones(len(cut_wjets))/len(cut_wjets), label = "$W+$",bins=np.
 ↪arange(0,1600,100));
axes[1,0].legend()

axes[1,1].set_title("mass of 3 jets");
axes[1,1].hist(np.clip(cut_ttbar['M3'],0,700), histtype = 'step', weights = np.
 ↪ones(len(cut_ttbar))/len(cut_ttbar), label = "$t \\bar{t}$", bins = np.
 ↪arange(0,720,20));
axes[1,1].hist(np.clip(cut_wjets['M3'],0,700), histtype = 'step', weights = np.
 ↪ones(len(cut_wjets))/len(cut_wjets), label = "$W+$", bins = np.
 ↪arange(0,720,20));
axes[1,1].legend()
fig.show();
```



1.6 5. Signal fraction and S/B

For calculating the $t\bar{t}$ cross-section, you need the number of $t\bar{t}$ events. For that you have to fit the signal fraction (f_{sig}) of the $t\bar{t}$ signal to the data. The signal fraction is the number of $t\bar{t}$ events divided by the total number of observed data events. Use the variable M_3 to determine the signal fraction. Calculate the signal to background ratio (S/B). Check whether a cut on the number of b-tagged jets ($N_{\text{b-tag}} >= 1$) in an event increases the signal fraction and S/B.

First you need the histogram of the M_3 distribution for every dataset.

[21]:
```
data_ttbar["M3"] = data_ttbar.apply(lambda x: m3(x["Jets"]),axis=1)
DataHistM3 = np.histogram(np.clip(data_ttbar["M3"],0,700), bins = np.
 ↪arange(0,720,20), weights = np.ones(len(data_ttbar))/len(data_ttbar));
N_data = DataHistM3[0]

N_signal = np.histogram(np.clip(cut_ttbar["M3"],0,700), bins = np.
 ↪arange(0,720,20), weights = np.ones(len(cut_ttbar))/len(cut_ttbar))[0];

N_background = np.histogram(np.clip(cut_wjets["M3"],0,700), bins = np.
 ↪arange(0,720,20), weights = np.ones(len(cut_wjets))/len(cut_wjets))[0];
```

Now write the χ^2 function:

$$\chi^2 = \sum^{\text{bins}} \frac{(N_{\text{predicted}} - N_{\text{measured}})^2}{N_{\text{measured}}}$$

$$N_{\text{predicted}} = f \cdot N_{\text{signal}} - (1 - f) \cdot N_{\text{background}}$$

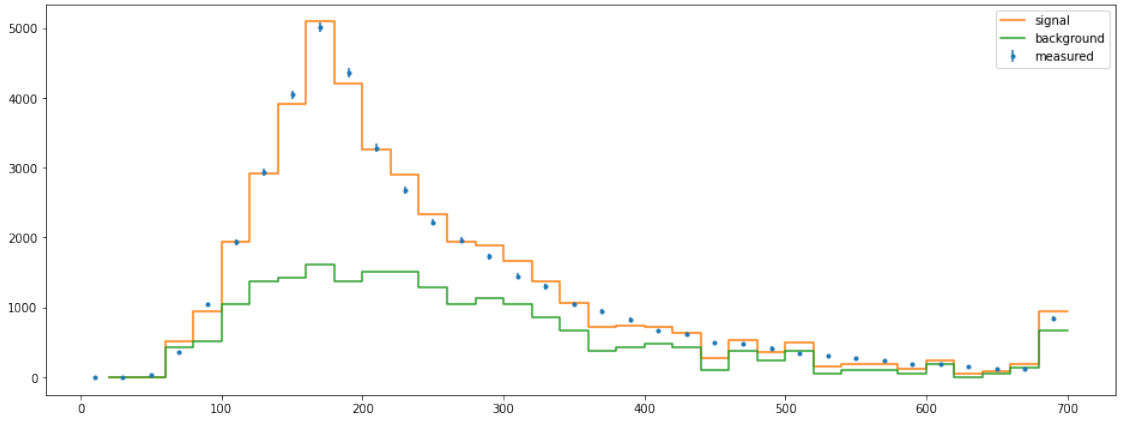
It should only take the signal ratio as an argument. Remember to scale the results correctly back to the sample size of the measured data. Also exclude the first 3 bins, since they mostly do not contain any events.

[22]:
```
# chi^2 funktion
l = len(data_ttbar)
def N_pred(f_sig=0.5):
    return N_signal*f_sig*l-N_background*(1-f_sig)*l

def chisqu(f_sig):
    N_pred=f_sig*N_signal*l - (1-f_sig)*l*N_background
    chi = (N_pred[3:]-N_data[3:]*l)**2/N_pred[3:]
    return sum(chi)
```

Now run the Fit the function and plot the results. You may want to test different starting starting values since the outcome heavily depends on it.

[23]:
```
miu = Minuit(chisqu, f_sig=0.5, error_f_sig=0.01, limit_f_sig=(0.1,1),␣
 ↪errordef=1)
miu.migrad()
f = miu.values["f_sig"]
print(f)
print(f/(1-f))
bs = DataHistM3[1][1:]
plt.errorbar(bs-10,N_data*l,fmt = '.', yerr = np.sqrt(N_data*l), label =␣
 ↪'measured')
plt.plot(bs,N_signal*f*l+N_background*(1-f)*l,drawstyle = "steps", label =␣
 ↪'signal')
plt.plot(bs,N_background*l*(1-f),drawstyle = "steps", label = 'background')
plt.legend();
```

```
0.5081362237225805
1.0330832401774248
```



Select only events with at least one b-tagged Jet with a working medium working point (tagvalue > 0.679) and repeat the fit for these. To repeat the Fit for b-tagged Jets with a medium working point (tagvalue > 0.679) write a function that returns a boolean if an event contains at least one medium working point b-tag.

[24]:
```
# b-tag selection
def btag(x,threshold):
    for i in x:
        if i>threshold:
            return 1
    return 0
```

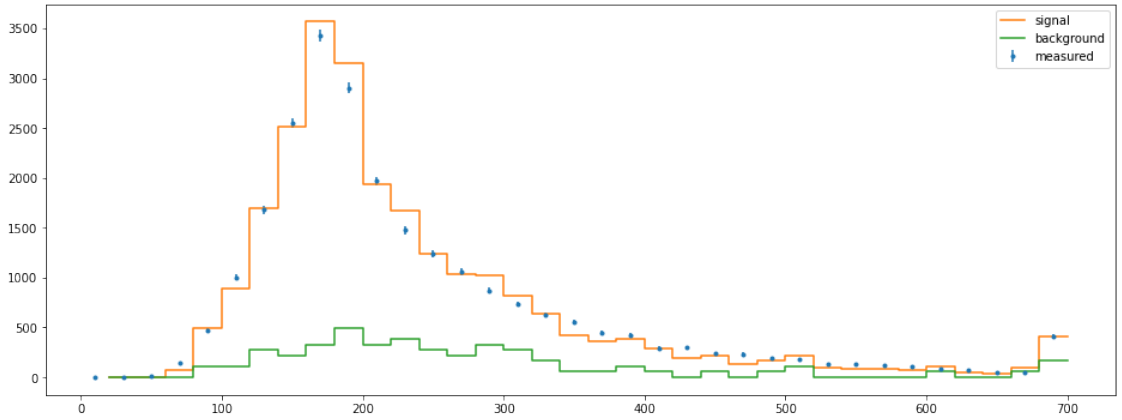
Perform the cut on events containing the necessary b-tag. Then generate the new histograms for the monte carlo and measured data.

[25]:
```
data_ttbar['Btag'] = data_ttbar.apply(lambda x: btag(x['Jets bTag'],0.
↪679),axis=1)
cut_data = data_ttbar.loc[data_ttbar['Btag']==1]
N_data = np.histogram(np.clip(cut_data["M3"],0,700), bins = np.
↪arange(0,720,20), weights = np.ones(len(cut_data))/len(cut_data))[0];

cut_ttbar['Btag'] = cut_ttbar.apply(lambda x: btag(x['Jets bTag'],0.679),axis=1)
cut_ttbar = cut_ttbar.loc[cut_ttbar['Btag']==1]
N_signal = np.histogram(np.clip(cut_ttbar["M3"],0,700), bins = np.
↪arange(0,720,20), weights = np.ones(len(cut_ttbar))/len(cut_ttbar))[0];

cut_wjets['Btag'] = cut_wjets.apply(lambda x: btag(x['Jets bTag'],0.679),axis=1)
cut_wjets = cut_wjets.loc[cut_wjets['Btag']==1]
N_background = np.histogram(np.clip(cut_wjets["M3"],0,700), bins = np.
↪arange(0,720,20), weights = np.ones(len(cut_wjets))/len(cut_wjets))[0];
```

Perform the same fit from above. Watch out for the starting value and the correct length of the measured data to scale the simulation.

[26]:
```
# S/B ratio with b-tag
l= len(cut_data)
miu = Minuit(chisqu, f_sig=0.8, error_f_sig=0.01, limit_f_sig=(0.1,1),␣
↪errordef=1)
miu.migrad()
f = miu.values["f_sig"]
print(f)
print(f/(1-f))
bs = DataHistM3[1][1:]
plt.errorbar(bs-10,N_data*l,fmt = '.', yerr = np.sqrt(N_data*l),label =␣
↪'measured')
plt.plot(bs,N_signal*f*l+N_background*(1-f)*l,drawstyle = "steps",label =␣
↪'signal')
plt.plot(bs,N_background*l*(1-f),drawstyle = "steps", label = 'background')
plt.legend();
```

```
0.8224538453116776
4.632338260186338
```



1.7 6. ttbar - cross-section

Calculate the cross-section for $t\bar{t}$ production at $\sqrt{s} = 5$TeV. The dataset corresponds to an integrated luminosity of $L = 5\text{fb}^{-}1$.

$$\sigma_{t\bar{t}} = \frac{N^{\text{data}} \cdot f_{\text{sig}}}{L \cdot \epsilon_{\text{eff}}}$$

How does your result compare to the predicted theoretical cross-section: $\sigma_{t\bar{t}}^{\text{N N LO approx}}(m = 173\ \text{GeV/c}^2, \sqrt{s} = 7\ \text{TeV}) = 163^{+7+9}_{-5-9}$ pb?

[27]:
```
L = 5
epsilon = 0.0259
sigma = sum(N_data)*l * f / L / epsilon
sigma
```

[27]:
```
153287.61398031437
```

1.8 7. Reconstruction of the top quark 4-vectors

Reconstruct the semileptonically decaying top quark pair from four measured jets, the muon and the missing transverse moment. In some events more than four jets are present and you do not know which jet comes from which quark of the $t\bar{t}$ event. To get the best combination for an event reconstruct all possible jet combinations and select the combination with the smallest χ^2 value. Ensure that at least one jet among the four used jets is b-tagged. Use here the loose working point (b_tag > 0.244) for b-tagging. Reconstruct the mass of the leptonically ($m_{t,\text{lep}}$ and hadronically($m_{t,\text{had}}$) decaying top quarks. Calculate also for each $t\bar{t}$ pair the average top quark mass ($m_{t,\text{av}} = \frac{m_{t,\text{lep}}+m_{t,\text{had}}}{2}$). Fit these three top quark masses and the M_3 distribution from exercise 4. These distributions are not symmetric so you can not use a Gaussian function, but a Landau function. Compare the width of the distribution of M_3 and that of the reconstructed top quark masses. Do you see an improvement from M_3 to the reconstructed top quark masses? The given function `calcNuPz` will be used to calculate the 4-vector of the neutrino.

Start by calculating the 4-vector of the leptonically decaying W-Boson: Reconstruct the Neutrino from the missing momentum in the x- and y-direction. The z-component can be calculated using the `calcNuPz` function. It takes the muon 4-vector and the missing Energy as arguments and returns the z-component of the neutrino momentum. Adding the muon and the neutrino results in the 4-vector of the W-Boson

[28]:
```
def wminus(x):
    arr = np.array([0,x['missing E'][1],x['missing␣
↪E'][2],calcNuPz(x['Muons'],x['missing E'])])
    arr[0] = p(arr)
    arr+=x['Muons']
    return arr
```

[29]:
```
cut_data['Wminus'] = cut_data.apply(lambda x: wminus(x),axis=1)
```

Now reconstruct the hadronic decay and the top quarks: At least one b-quark should be loosely b-tagged. Write loops, that test all allowed combinations of Jets for the b-quark and the quarks resulting from the decay of the hadronic W-Boson. Calculate the χ^2 value :

$$\chi^2 = \frac{(m_{W_{\text{had}}} - M_{W_{\text{had,exp}}})^2}{\sigma^2_{M_{W_{\text{had,exp}}}}} + \frac{(m_{t_{\text{lep}}} - m_{t_{\text{had}}} - \Delta M)^2}{\sigma^2_{\Delta M_{t_{\text{exp}}}}}$$

Keep the combination with the lowest χ^2. One way to achieve this is to create a new column "indices" containing a 4 component array. Each entry stands for the position of one Jet in the dataFrame. The first entry in indices will represent the hadronic b Jet, the next two the hadronic quarks from the W-boson and the last the leptonic b-quark.

$$M_{W_{\text{had,exp}}} = 78.1 GeV/c^2$$

$$\sigma_{M_{W_{\text{had,exp}}}} = 11.2 GeV/c^2$$

$$\Delta M = -7.5 GeV/c^2$$

$$\sigma_{\Delta M_{t_{\text{exp}}}} = 45.2 GeV/c^2$$

The argument of the function `x` will be an event. The entries can be called the same way as in a `DataFrame`: `x['Jets']` returns the array of Jets.

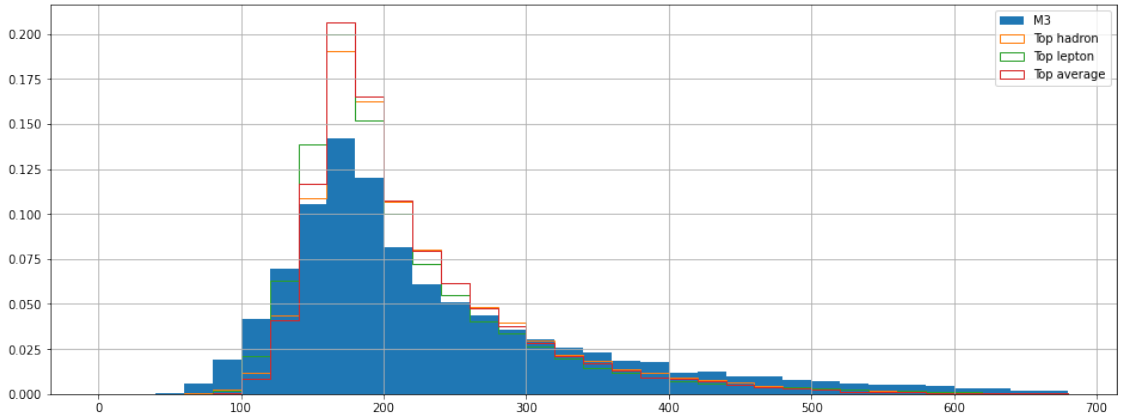
[30]:
```
# reconstruct the top quarks
def rec_jets(x):
    NJets = len(x['Jets'])
    Jets = np.zeros(NJets)-1
    thad = np.zeros(4)
    tlep = np.zeros(4)
    chi2 = 1000
    for bh in range(NJets):
        for bl in range(NJets):
            if bh==bl: continue # not possible combination
            if x['Jets bTag'][bh]>0.244 or x['Jets bTag'][bl]>0.244: # at least␣
↪one b-Jet has to be btaged
```

```
                for q1 in range(NJets):
                    if q1 in ([bh,bl]): continue # not possible combination
                    for q2 in range(NJets):
                        if q2 in ([bh,bl,q1]): continue # not possible␣
↪combination
                        Wplus = x['Jets'][q1]+x['Jets'][q2]
                        thad_helper = Wplus + x['Jets'][bh]
                        tlep_helper = x['Wminus'] + x['Jets'][bl]
                        chi2_helper = (m(Wplus)-78.1)**2/11.2**2 +␣
↪(m(tlep_helper)-m(thad_helper)+7.5)**2/45.2**2
                        if chi2_helper < chi2:
                            chi2 = chi2_helper
                            thad = thad_helper
                            tlep = tlep_helper
    return thad,tlep
```

[31]:
```
Rec = cut_data.apply(lambda x: rec_jets(x),axis=1)
cut_data['Top hadron'] = Rec.apply(lambda x: m(x[0]))
cut_data['Top lepton'] = Rec.apply(lambda x: m(x[1]))
cut_data['Top average'] = cut_data.apply(lambda x: (x['Top hadron'] + x['Top␣
↪lepton'])/2,axis=1)
```

Create the histograms for the hadronic top, the leptonic top and the average between them.

[32]:
```
plt.grid()
plt.hist(cut_data["M3"],histtype = "stepfilled", bins = np.arange(0,700,20),␣
↪label="M3", weights = np.ones(len(cut_data))/len(cut_data));
Top_hadron = plt.hist(cut_data["Top hadron"],histtype = "step", bins = np.
↪arange(0,700,20), label="Top hadron", weights = np.ones(len(cut_data))/
↪len(cut_data));
Top_lepton = plt.hist(cut_data["Top lepton"],histtype = "step", bins = np.
↪arange(0,700,20), label="Top lepton", weights = np.ones(len(cut_data))/
↪len(cut_data));
Top_averag = plt.hist(cut_data["Top average"],histtype = "step", bins = np.
↪arange(0,700,20), label="Top average", weights = np.ones(len(cut_data))/
↪len(cut_data));
plt.legend()
plt.show()
```



1.9 8. Determination of the top quark mass

For comparison start by running the algorithms from above to reconstruct the Quarks also for the Monte-Carlo-Datasets. The monte-carlo sample (`MC_TTBar_NJets2.csv`) was generated with a top quark mass of $m_{t,true} = 172.5 GeV/c^2$. When you fit the reconstructed average top quark mass, it's different to the top quark mass which was used by the generator. This difference between true and reconstructed top quark mass exists also in the observed data. Subtract the background from the data to get the distribution of the top quark mass for a pure $t\bar{t}$ signal. Fit a Landau function to this distribution and add the calculated difference from monte-carlo to the maximum of the fit function. This is the true top quark mass in data. How does your result compare with the PDG mass:$m_t = 173.5 \pm 0.6 \pm 0.8 GeV/c^2$?

1.9.1 8.1 Finding a parametrization

To describe the data it is useful to find a parametrization of the distribution of the top mass. Since the measured data contains background, we will use the monte carlo data to determine the shape parameters. The funktion used here is a modified Crystal Ball function

$$f(x;\beta,m,\mu,\sigma) = N \cdot \begin{cases} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) & \text{, for } \frac{(x-\mu)}{\sigma} > -\beta \\ \left(\frac{m}{|\beta|}\right)^m \cdot \exp\left(-\frac{|\beta|^2}{2}\right) \cdot \left(\frac{m}{|\beta|} - |\beta| - \frac{(x-\mu)}{\sigma}\right)^{-m} & \text{, for } \frac{(x-\mu)}{\sigma} \leq -\beta \end{cases}$$

Where

$$N = \frac{1}{\sigma(C+D)},$$

$$C = \frac{m}{|\beta|(m-1)} \cdot \exp\left(-\frac{|\beta|^2}{2}\right)$$

and

$$D = \sqrt{\frac{\pi}{2}}\left(1 + \operatorname{erf}\left(\frac{|\beta|}{\sqrt{2}}\right)\right).$$

For convenience reasons we will use following reparametrisation:

$$\alpha = \frac{1}{\beta}$$

and

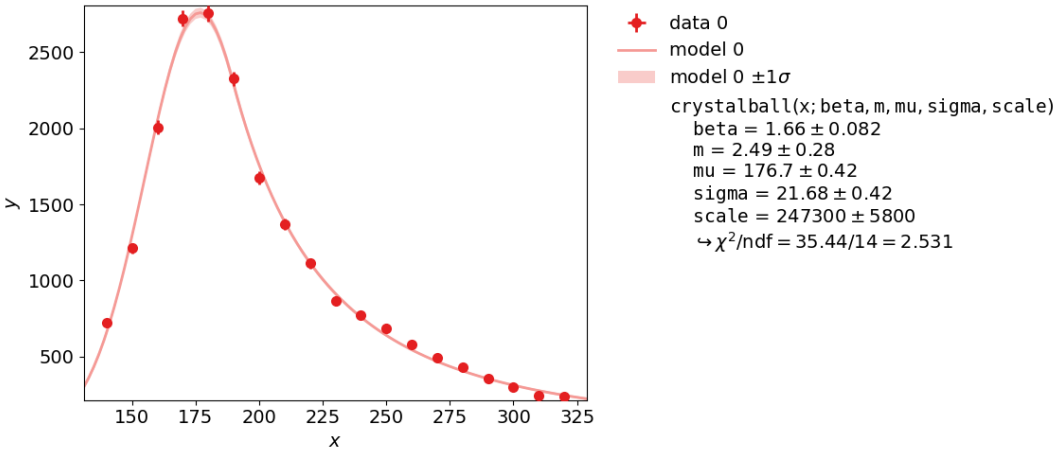
$$\mu = -\mu.$$

Start by reconstructing the tops in the monte carlo data `cut_ttbar`.

[33]:
```
# reconstruction for monte-carlo data
cut_ttbar['Wminus'] = cut_ttbar.apply(lambda x: wminus(x),axis=1)
Rec = cut_ttbar.apply(lambda x: rec_jets(x),axis=1)
cut_ttbar['Top hadron'] = Rec.apply(lambda x: m(x[0]))
cut_ttbar['Top lepton'] = Rec.apply(lambda x: m(x[1]))
cut_ttbar['Top average'] = cut_ttbar.apply(lambda x: (x['Top hadron'] + x['Top␣
↪lepton'])/2,axis=1)
```

Now fit the distribution of the reconstructed top mass to find a parametrization.

[34]:
```
# fit distribution of reconstructed monte-carlo
u,o = 130,330
hist = np.histogram(cut_ttbar['Top average'],bins = np.arange(u,o,10))
datset = kafe2.XYContainer(hist[1][1:],hist[0])
datset.add_error('y',np.sqrt(hist[0]));
datset.add_error('x',hist[1][1:]*0.0)
fit_mc = kafe2.Fit(datset, crystalball)
fit_mc.set_parameter_values(m = 2, beta = 1, mu = 172, sigma=20,scale = 250000)
fit_mc.limit_parameter('beta',(0,5))
fit_mc.limit_parameter('m',(1,3))
fit_mc.limit_parameter('sigma',(0,50))
fit_mc.do_fit()
plot = kafe2.Plot(fit_mc)
plot.plot();
```



1.9.2 8.2 Deviation from the generator mass

There will probably be a deviation of the top mass given by the parametrization function and the top mass used by the monte carlo generator ($172.5 GeV$). We will need this difference later to correct the top mass in the measured data. The parametrization will not have the peak position as an parameter!

[35]:
```
# calculate difference to true top mass
par = fit_mc.poi_values+fit_mc.parameter_errors
mc_top_mass = fit_mc.poi_values[2]
mc_top_mass_err = fit_mc.parameter_errors[2]
diff = mc_top_mass - 172.5
diff_err = mc_top_mass_err
print("Top mass of monte carlo ",mc_top_mass, " +/- ", mc_top_mass_err, "GeV;
 ↪ difference to generator top mass: ", diff, "GeV")
```

```
Top mass of monte carlo  176.697698468523  +/-  0.41916305740883475 GeV;
difference to generator top mass:  4.197698468522987 GeV
```

1.9.3 8.3 Subtracting the background

Now subtract the background from the measured data using the ratio calculated in exercise 5.

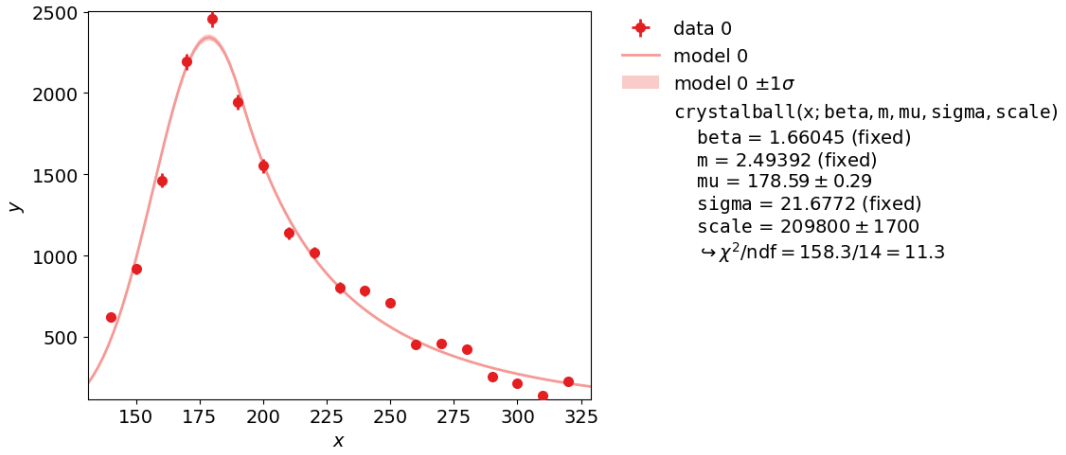
[36]:
```
# subtract the background from measured data
cut_wjets['Wminus'] = cut_wjets.apply(lambda x: wminus(x),axis=1)
Rec = cut_wjets.apply(lambda x: rec_jets(x),axis=1)
cut_wjets['Top hadron'] = Rec.apply(lambda x: m(x[0]))
cut_wjets['Top lepton'] = Rec.apply(lambda x: m(x[1]))
cut_wjets['Top average'] = Rec.apply(lambda x: (m(x[1]) + m(x[0]))/2)
l = len(cut_data)
hist_background = np.histogram(cut_wjets['Top average'], bins = np.
 ↪arange(u,o,10), weights = np.ones(len(cut_wjets))/len(cut_wjets));
Top_averag = np.histogram(cut_data['Top average'],bins = np.arange(u,o,10),
 ↪weights = np.ones(len(cut_data))/len(cut_data))
hist = Top_averag[0]*l-hist_background[0]*(1-f)*l
```

1.9.4 8.4 Determine the top mass

Now a fit of the data can be performed. We have the data striped of the background and a parametrization of the distribution. Use the fit on the monte carlo data as a template for the shape of the fit on the measured data. Take in to account the propagated errors and the deviation fromn the generator mass. The systematic error on the Jet Energy Resolution is $2.3 GeV$.

[37]:
```
# fit without the background
datset = kafe2.XYContainer(hist_background[1][1:],hist)
err = np.sqrt(l*Top_averag[0]+l*hist_background[0]*(1-f)**2)
```

```
datset.add_error(axis='y',err_val=err)
fit = kafe2.Fit(datset,crystalball)
fit.set_parameter_values(beta = fit_mc.poi_values[0], m = fit_mc.poi_values[1],␣
↪mu = fit_mc.poi_values[2],sigma= fit_mc.poi_values[3], scale = fit_mc.
↪poi_values[4])
fit.limit_parameter('mu',(170,180))
fit.fix_parameter('beta',fit_mc.poi_values[0])
fit.fix_parameter('m',fit_mc.poi_values[1])
fit.fix_parameter('sigma',fit_mc.poi_values[3])
fit.do_fit()
plot = kafe2.Plot([fit])
plot.plot();
```



[38]:
```
top_mass = fit.poi_values[2]-diff;
par = fit_mc.poi_values+fit_mc.parameter_errors
top_mass_err = np.sqrt(fit_mc.parameter_errors[2]**2 + diff_err**2 + 2.4**2)
print("The top mass is ",round(top_mass,5)," +/-", round(top_mass_err,5))
```

```
The top mass is  174.3932  +/- 2.47212
```

Literaturverzeichnis

[AAA+16] Abbott, B. P., R. Abbott, T. D. Abbott, M. R. Abernathy, F. Acernese und et al.: *Observation of Gravitational Waves from a Binary Black Hole Merger*. Phys. Rev. Lett., 116:061102, Feb 2016. https://link.aps.org/doi/10.1103/PhysRevLett.116.061102.

[ABB+09] Antcheva, I., M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, Ph. Canal, D. Casadei, O. Couet, V. Fine, L. Franco, G. Ganis, A. Gheata, D. Gonzalez Maline, M. Goto, J. Iwaszkiewicz, A. Kreshuk, D. Marcos Segura, R. Maunder, L. Moneta, A. Naumann, E. Offermann, V. Onuchin, S. Panacek, F. Rademakers, P. Russo und M. Tadel: *ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization*. Computer Physics Communications, 180(12):2499 – 2512, 2009, ISSN 0010-4655. http://www.sciencedirect.com/science/article/pii/S0010465509002550, 40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures.

[Bau11] Baun, Christian: *Cloud Computing : Web-Based Dynamic IT Services*, 2011, ISBN 9783642209178. http://swbplus.bsz-bw.de/bsz348695403cov.htmhttps://doi.org/10.1007/978-3-642-20917-8.

[Bha18] Bhat, Sathyajith: *Practical Docker with Python : Build, Release and Distribute your Python App with Docker*, 2018, ISBN 9781484237847. http://dx.doi.org/10.1007/978-1-4842-3784-7.

[Col11] Collaboration, CMS: *Measurement of the top quark mass in the l+jets channel*. Technischer Bericht CMS-PAS-TOP-10-009, CERN, Geneva, 2011. http://cds.cern.ch/record/1356578.

[Cor] Corporation, OriginLab: *OriginPro*. https://www.originlab.com/, Zugriff am 23.04.2020.

[Cor19] Corlay, Sylvain: *Voilà is now a Jupyter subproject*, Dec 2019. https://blog.jupyter.org/voilà-is-now-an-official-jupyter-subproject-87d659583490.

[gC] Community git: *Webpräsenz zu git*. https://git-scm.com/, Zugriff am 23.04.2020.

[Gif19] Giffels, Manuel: *private Kommunikation*, november 2019.

[Gru] Gruber, John: *Website zu Markdown*. https://daringfireball.net/projects/markdown/, Zugriff am 23.3.2020.

[Hun07] Hunter, J. D.: *Matplotlib: A 2D Graphics Environment*. Computing in Science Engineering, 9(3):90–95, 2007.

[Inca] Inc., Docker: *Manage sensitive data with Docker secrets*. https://docs.docker.com/engine/swarm/secrets/, Zugriff am 4.4.2020.

[Incb] Inc., Gitlab: *Gitlab Container Registry in Gitlabs Documentation.* https://docs.gitlab.com/ee/user/packages/container_registry/, Zugriff am 23.04.2020.

[Incc] Inc., GitLab: *GitLab Documemtation.* https://docs.gitlab.com/, Zugriff am 14.4.2020.

[JBB+19] Jupyter, Project, Douglas Blank, David Bourgin, Alexander Brown, Matthias Bussonnier, Jonathan Frederic, Brian Granger, Thomas Griffiths, Jessica Hamrick, Kyle Kelley, M Pacer, Logan Page, Fernando Pérez, Benjamin Ragan-Kelley, Jordan Suchow und Carol Willing: *nbgrader: A Tool for Creating and Grading Assignments in the Jupyter Notebook.* Journal of Open Source Education, 2(16):32, 2019. https://doi.org/10.21105/jose.00032.

[Jupa] Jupyter, Project: *JupyterLab Documentation - JupyterLab 2.0.1 documentation.* https://jupyterlab.readthedocs.io/en/latest/, Zugriff am 11.3.2020.

[Jupb] Jupyter, Project: *jupyterlab-git Github Repository.* https://github.com/jupyterlab/jupyterlab-git, Zugriff am 18.04.2020.

[Jupc] Jupyter, Project: *Project Jupyter | Home.* https://jupyter.org, Zugriff am 10.3.2020.

[Kit] *Studierendenstatistiken des KIT.* https://www.kit.edu/kit/6407.php, Zugriff am 15.0.4.2020.

[KRKP+16] Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay und et al.: *Jupyter Notebooks - a publishing format for reproducible computational workflows.* Stand Alone, 0(Positioning and Power in Academic Publishing: Players, Agents and Agendas):87–90, 2016, ISSN 0000-0000. http://doi.org/10.3233/978-1-61499-649-1-87.

[LABZ19] Lorena A. Barba, Lecia J. Barker, Douglas S. Blank Jed Brown Allen B. Downey Timothy George Lindsey J. Heagy Kyle T. Mandli Jason K. Moore David Lippert Kyle E. Niemeyer Ryan R. Watkins Richard H. West Elizabeth Wickes Carol Willing und Michael Zingale: *Teaching and Learning with Jupyter*, 2019. https://jupyter4edu.github.io/jupyter-edu-book/.

[Len19] Lenz, Moritz [VerfasserIn]: *Python Continuous Integration and Delivery : A Concise Guide with Examples*, 2019, ISBN 9781484242810. https://doi.org/10.1007/978-1-4842-4281-0;http://dx.doi.org/10.1007/978-1-4842-4281-0.

[Mab20] Mabille, Johan: *Xeus is now a Jupyter subproject*, Feb 2020. https://blog.jupyter.org/xeus-is-now-a-jupyter-subproject-c4ec5a1bf30b.

[McK19] McKinney, Wes [VerfasserIn]: *Datenanalyse mit Python : Auswertung von Daten mit Pandas, NumPy und IPython*, [2019], ISBN 9783960102137; 9783960102144; 9783960102151. http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=1923617.

[mT] Team matplotlib: *ipympl - Matplotlib Jupyter Integration.* https://github.com/matplotlib/ipympl, Zugriff am 20.04.2020.

[Ole10] Oleari, C.: *The POWHEG BOX.* Nuclear Physics B - Proceedings Supplements, 205-206:36–41, August 2010. https://doi.org/10.1016/j.nuclphysbps.2010.08.016.

[Org80] Orgelia, Mark: *A Study of the Reactions* $\Psi' \rightarrow \gamma\gamma\Psi$, december 1980, ISBN SLAC-R-236. https://slac.stanford.edu/pubs/slacreports/reports13/slac-r-236.pdf.

[oseLe] Learning e.V., ILIAS open source e: *ILIAS Webauftritt*. https://www.ilias.de/open-source-lms-ilias/, Zugriff am 18.04.2020.

[PG07] Pérez, Fernando und Brian E. Granger: *IPython: a System for Interactive Scientific Computing*. Computing in Science and Engineering, 9(3):21–29, Mai 2007, ISSN 1521-9615. https://ipython.org.

[QGCS] Quast, Günter, Johannes Gäßler, Verstege Cedric und Daniel Saviou: *kafe2 documentation*. https://kafe2.readthedocs.io/en/latest/, Zugriff am 17.4.2020.

[Sch13] Schnepf, Matthias: *Erstellung eines Pratikumsversuches zur Untersuchung von Top-Quark-Paar-Ereignissen am LHC mit dem CMS-Experiment*, 2013.

[Să13] Săvoiu, Daniel: *kafe – Ein Python-Paket für elementare Datenanalyse im Physikpraktikum*. Dissertation, Karlsruhe Institute of Technology (KIT), 2013.

[Teaa] Team, IMinuit: *IMinuit – A Python interface to Minuit*. https://github.com/scikit-hep/iminuit. Accessed: 17.4.2020.

[Teab] Team, Jupyter Development: *nbgrader Dokumentation*. https://nbgrader.readthedocs.io/en/stable/, Zugriff am 18.04.2020.

[THH+18] Tanabashi, M., K. Hagiwara, K. Hikasa, K. Nakamura, Y. Sumino, F. Takahashi, J. Tanaka, K. Agashe und et al.: *Review of Particle Physics*. Phys. Rev. D, 98:030001, Aug 2018. https://link.aps.org/doi/10.1103/PhysRevD.98.030001.

[vCV11] van der Walt, S., S. C. Colbert und G. Varoquaux: *The NumPy Array: A Structure for Efficient Numerical Computation*. Computing in Science Engineering, 13(2):22–30, 2011.

[Ver19] Verstegen, Cedric: *Likelihood-Anpassungen im physikalischen Fortgeschrittenen-Praktikum*, 2019.

[WM10] McKinney Wes: *Data Structures for Statistical Computing in Python*. In: Walt Stéfan van der und Jarrod Millman (Herausgeber): *Proceedings of the 9th Python in Science Conference*, Seiten 56 – 61, 2010.