ETP-Bachelor-2018-11

Karlsruhe Institute of Technology

FAKULTÄT FÜR PHYSIK
Institut für Experimentelle Teilchenphysik

Subprozessfunktionalität in fastNLO

Bachelorarbeit
von

Jakob Stark

am Institut für Experimentelle Teilchenphysik

Referent: Dr. Klaus Rabbertz
Korreferent: Prof. Dr. Günter Quast

05.06.2018

Erklärung zur Selbstständigkeit

Ich versichere, dass ich diese Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe.

Karlsruhe, den 07.06.2018, _____

Jakob Stark

Inhaltsverzeichnis

1. Einführung

Mit der Entwicklung von immer stärkeren Teilchenbeschleunigern in der zweiten Hälfte des 20. Jahrhunderts konnte immer mehr Energie in die Kollisionen von Teilchen gesteckt werden. Die Systematik der vielen Teilchen, die dadurch auf experimentellem Wege gefunden wurden, wird heutzutage mit dem Standardmodell der Teilchenphysik beschrieben. Die Grundpfeiler des Standardmodells sind durch die Theorie der elektroschwachen Wechselwirkung und die Quantenchromodynamik gegeben. Die Theorien beschreiben den Aufbau aller bis heute bekannten Teilchen und die Wechselwirkungen zwischen ihnen.

Obwohl das Standardmodell viele experimentelle Ergebnisse richtig vorhersagt, lassen sich manche Probleme damit (noch) nicht lösen. Ein Beispiel ist die genaue innere Struktur des Protons: Die Berechnungen sind zwar theoretisch möglich, aber zu komplex um mit der heutigen Rechenleistung durchgeführt zu werden. Die Beschreibung der Struktur von Protonen anhand von Partonverteilungsfunktionen ist zur Zeit nur durch experimentell gewonnene Daten gegeben [1]. Um diese Funktionen zu bestimmen, können zum Beispiel Kollisionen von Protonen mit Elektronen oder mit anderen Protonen beobachtet werden. Bis 2007 wurden am DESY (Deutsches Elektronen-Synchrotron) im HERA Speicherring Protonen mit Elektronen zur Kollision gebracht (Tiefinelastische Streuung). Dabei konnte zum ersten Mal mit großer Genauigkeit andere Konstituenten, als die drei sogenannten Valenzquarks im Proton nachgewiesen werden und Partonverteilungsfunktionen gefittet werden [2, 3].

Das in dieser Arbeit behandelte Softwarepaket *fastNLO* liefert eine Möglichkeit, die zur Bestimmung oder Verbesserung der Partonverteilungsfunktionen nötigen Berechnungen, wie zum Beispiel die Berechnung von Gesamtwirkungsquerschnitten, schneller durchzuführen. Zur Bestimmung der Unsicherheiten der einzelnen Partonverteilungsfunktionen kann es hilfreich sein, nicht nur den gesamten Wirkungsquerschnitt der Kollision zu betrachten, sondern die Beiträge einzelner partonischer Subprozesse dazu. Im Rahmen dieser Arbeit wurde in *fastNLO* neue Funktionalität eingebaut, um auf eine einfache Art und Weise die Subprozessbeiträge zum Gesamtwirkungsquerschnitt auszurechnen. In Kapitel 2 wird eine kurze Einführung in die zugrundeliegende Theorie der Quantenchromodynamik (QCD) gegeben. Das *fastNLO* zugrundeliegende Konzept wird in Kapitel 3 vorgestellt. Die neue Methode zur Berechnung von einzelnen Teilwirkungsquerschnitten wird dann in Kapitel 4 ausführlich erklärt.

2. Theoretische Grundlagen

Der theoretische Rahmen der von *fastNLO* behandelten Kollisionen, bei denen Hadronen beteiligt sind, wird von der Quantenchromodynamik abgedeckt. Diese oft als QCD abgekürzte Quantenfeldtheorie beschreibt die starke Wechselwirkung und stellt damit einen der Grundpfeiler des Standardmodells der Teilchenphysik.

2.1. Einführung in die QCD

Die von der QCD beschriebene starke Wechselwirkung ist eine Eichtheorie deren Eichgruppe die spezielle unitäre Gruppe SU(3) ist. Ein mit der starken Wechselwirkung wechselwirkendes Teilchen hat eine Farbladung, die eine dreizählige Größe ist. In Analogie zur Farbenlehre werden die drei Basiszustände als Rot, Grün und Blau bezeichnet. Ein Teilchen kann einen dieser Zustände oder Linearkombinationen davon annehmen. Die Austauschteilchen der starken Wechselwirkung werden Gluonen genannt. Im Gegensatz zu z.B. dem Austauschteilchen der Quantenelektrodynamik, dem Photon, tragen die Gluonen selbst Farbladungen und wechselwirken damit untereinander. Neben den Gluonen wechselwirken lediglich die Quarks (und Antiquarks) über die starke Wechselwirkung.

Eine wichtige Eigenschaft der QCD ist die laufende starke Kopplungskonstante, das bedeutet die Kopplungskonstante α_s ist abhängig, von der Energieskala des untersuchten Prozesses. Daraus resultieren zwei grundlegende Eigenschaften der QCD: [4]

- Während zum Beispiel das Coulombpotential mit dem Abstand asymptotisch gegen Null geht, nimmt das Potential der starken Kernkraft linear mit dem Abstand zu. Wird der Abstand zweier farbgeladenen Teilchen groß genug, so enthält das Gluonfeld dazwischen irgendwann genug Energie, um ein neues farbgeladenes Teilchen-Antiteilchen-Paar zu bilden. In der Konsequenz können keine freie farbgeladenen Teilchen existieren. Dieses Phänomen wird als *Confinement* bezeichnet.
- Mit zunehmender Energieskala nimmt die starke Wechselwirkung asymptotisch ab, so dass die an einem Prozess beteiligten Teilchen bei großen Energien nahezu als freie Teilchen behandelt werden können. Dieses für die starke Wechselwirkung typische Verhalten wird als *asymptotische Freiheit* bezeichnet.

2.2. Partonverteilungsfunktionen

Bei den von *fastNLO* behandelten hadronischen Kollisionen spielt der innere Aufbau der Hadronen eine entscheidende Rolle. Neben den sogenannten Valenzquarks, die die Quantenzahlen und damit den Typ des Hadrons festlegen, enthält ein Hadron noch Anteile von anderen Partonen nämlich Gluonen und Seequarks, virtuelle Quarks und Antiquarks, die ständig entstehen und verschwinden. Der Anteil der Quarks (sowohl der See- als auch der Valenzquarks) und Gluonen im Hadron wird durch die Partonverteilungsfunktionen (PDFs) beschrieben. Eine PDF ist die Wahrscheinlichkeitsdichte, ein bestimmtes Parton mit einem bestimmten Impulsanteil am Hadron zu finden.

PDFs können nicht durch perturbative Rechnungen bestimmt werden und müssen deshalb experimentell gewonnen werden. Die tiefinelastische Streuung von Elektronen an Protonen liefert Daten, die von den Partonverteilungsfunktionen des Protons abhängig sind. An diese Daten lassen sich die PDFs anpassen. Solche Untersuchungen wurden zum Beispiel am HERA-Experiment durchgeführt [3]. Die aus experimentellen Daten gewonnenen PDFs können stetig verbessert werden, indem sie an weitere Daten angepasst werden. Neben den Daten aus tiefinelastischer Streuung, können auch Proton-Proton-Kollisionen zur Bestimmung oder Verbesserung der PDFs verwendet werden. Die Proton-Proton-Kollisionen sind insofern schwieriger zu behandeln, als das hier die Struktur beider Kollisionsteilchen unbekannt ist.

Sowohl bei der tiefinelastischen Streuung als auch bei Proton-Proton-Kollisionen können die beobachtbaren Daten (Wirkungsquerschnitte) mithilfe der PDFs theoretisch vorhergesagt werden. Für das Anpassen der PDFs, so dass die theoretischen Vorhersagen mit den Daten übereinstimmen (PDF-fitting), ist es elementar, dass die theoretischen Berechnungen vergleichsweise schnell durchgeführt werden können. Die theoretischen Berechnungen durch perturbative QCD sind jedoch sehr aufwendig und benötigen viel Rechenzeit. Allerdings lassen sich die zeitaufwendigen perturbativen Rechnungen unabhängig von den PDFs durchführen. Um sozusagen die Zwischenergebnisse der kompletten Rechnung abzuspeichern und dann mithilfe der zu fittenden PDFs beobachtbare Wirkungsquerschnitte auszurechnen, sind Programme wie *fastNLO*[5, 6, 7] oder *APPLgrid*[8, 9] entwickelt worden.

2.3. Faktorisierungstheorem

Die beobachtbaren Größen bei Streuexperimenten sind die Wirkungsquerschnitte von bestimmten Prozessen. In der QCD können diese Streuwirkungsquerschnitte mithilfe des Faktorisierungstheorems bestimmt werden [4]:

$$\sigma(\mu_r, \mu_f) = \sum_i \int c_i\left(x_a, x_b, \mu_r, \mu_f, \alpha_s(\mu_r)\right) F_i(x_a, x_b, \mu_f) dx_a dx_b \tag{2.1}$$

Summiert wird dabei über alle Partonkombinationen (mit i durchnummeriert), die als Subprozesse zum Gesamtwirkungsquerschnitt beitragen. Integriert wird über die Impulsanteile x_a und x_b, die die verschiedenen Partonen zu den Gesamtimpulsen der beiden Hadronen beitragen, gewichtet mit den beiden PDFs $f_{i,1}$ des ersten und $f_{i,2}$ des zweiten Subprozesspartons.

$$F_i(x_a, x_b, \mu_f) = f_{i,1}(x_a, \mu_f) f_{i,2}(x_b, \mu_f)$$

Dazu kommt der perturbativ berechenbare Teil c_i, die sogenannten Matrixelemente der harten Kollision, die für jeden Subprozess einzeln den dazugehörigen Wirkungsquerschnitt festlegen. Sie werden zur konkreten Rechnung in Potenzen der starken Kopplungskonstante α_s entwickelt, sodass sich ergibt:

$$c_i\left(x_a, x_b, \mu_r, \mu_f, \alpha_s(\mu_r)\right) = \sum_n c_{i,n}\left(x_a, x_b, \mu_r, \mu_f\right) \alpha_s^n(\mu_r)$$

Die niedrigste von Null verschiedene Ordnung α_s^n wird als führende Ordnung (Leading Order oder kurz LO) bezeichnet, die Ordnung α_s^{n+1} entsprechend als Next to Leading Order (kurz NLO) und so weiter. In der Regel werden die Berechnungen der Matrixelemente zu höheren Ordnungen immer komplexer und aufwendiger, da immer mehr Möglichkeiten existieren, auf welche Art und Weise der Prozess ablaufen kann.

3. Der Aufbau von fastNLO

Das Softwarepaket *fastNLO* bietet eine Möglichkeit Wirkungsquerschnitte von QCD Prozessen, in denen Hadronen beteiligt sind, schnell für unterschiedliche Partonverteilungsfunktionen (PDFs) zu berechnen. Ziel davon ist es, eine Möglichkeit zu bieten PDFs an Daten zu fitten (vgl. Abschnitt 2.2)

3.1. Das Prinzip von *fastNLO*

Wirkungsquerschnitte von Prozessen, bei denen Hadronen (etwa Protonen) beteiligt sind, lassen sich mithilfe der Störungsrechnung der QCD berechnen. Dazu wird eine Reihenentwicklung in der Kopplungskonstante α_s der starken Wechselwirkung vorgenommen. Bei einer Hadron-Hadron Kollision müssen die PDFs von beiden involvierten Hadronen berücksichtigt werden. Der Wirkungsquerschnitt wird dann wie folgt berechnet [4]:

$$\sigma(\mu_r, \mu_f) = \sum_{n,i} \int c_{n,i}(x_a, x_b, \mu_r, \mu_f) \alpha_s^n(\mu_r) F_i(x_a, x_b, \mu_f) dx_a dx_b \tag{3.1}$$

Dabei läuft die Summe über i über alle partonischen Subprozesse, die zu dem Wirkungsquerschnitt beitragen. Die PDFs $F_i(x_a, x_b)$ enthalten die Information, bei welchem Impuls welche Partonen mit welchem Anteil im Hadron befinden. Die Impulsbruchteile der beiden Hadronen sind in (3.1) mit x_a und x_b bezeichnet. Während die PDFs nicht perturbativ berechnet werden können, können die Koeffizienten $c_{n,i}$ durch Monte-Carlo-Simulationen beliebig genau bestimmt werden. Dieser Prozess ist allerdings sehr zeitaufwendig. Um PDFs zu fitten, wird eine schnelle Methode benötigt, den Wirkungsquerschnitt (3.1) mit verschiedenen PDFs zu berechnen. In *fastNLO* werden nun Interpolationstabellen für die Koeffizienten $c_{n,i}$ erstellt. Dazu werden die PDFs durch eine Interpolation angenähert [5]:

$$F_i(x_a, x_b) \approx \sum_{k,l} F_i\left(x_a^{(k)}, x_b^{(l)}\right)\ e^{(k)}(x_a)\ e^{(l)}(x_b) \tag{3.2}$$

Wird dieser Ausdruck in Gleichung (3.1) eingesetzt, dann kann die Größe F_i aus dem Integral gezogen werden. Die Skalenabhänigkeit wird von *fastNLO* nach dem gleichen Prinzip interpoliert, wird jedoch in den folgenden Gleichungen weggelassen. Es ergibt sich

$$\sigma = \sum_{n,i,k,l} \alpha_s^n\ F_i\left(x_a^{(k)}, x_b^{(l)}\right) \underbrace{\int c_{n,i}(x_a, x_b)\ e^{(k)}(x_a)\ e^{(l)}(x_b)\ dx_a dx_b}_{=:\tilde{\sigma}_{n,i,k,l}} \tag{3.3}$$

Auf diese Weise wird die zeitaufwendige Berechnung der $\tilde{\sigma}_{n,i,k,l}$ unabhängig von den PDFs ausgeführt. Sind die Matrixelemente $\tilde{\sigma}_{n,i,k,l}$ einmal bekannt, kann der Wirkungsquerschnitt ohne großen Zeitaufwand mittels Gleichung (3.3) zu verschiedenen PDFs berechnet und

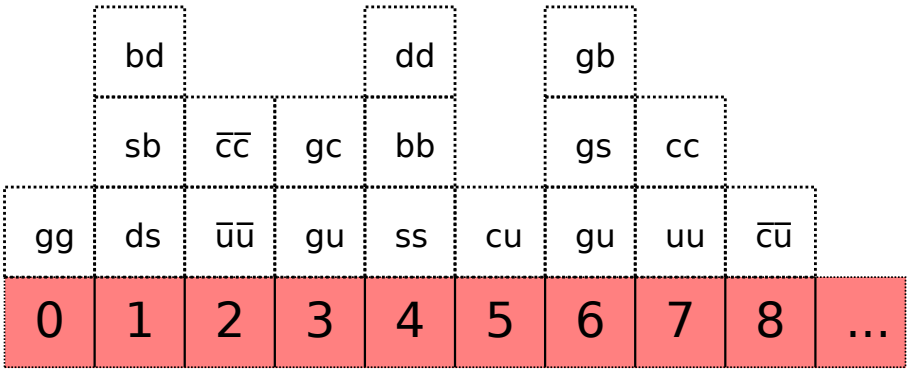


Abbildung 3.1.: Beispiel einer möglichen Zuordnung von Subprozessen zu Unterbeiträgen

damit auch zum Fitten von PDFs verwendet werden. Die Funktionen $e^{(k)}(x)$ sind dabei Interpolationsfunktionen, die in der Summe 1 ergeben. In *fastNLO* werden in der Regel Lagrangefunktionen als Interpolationsfunktionen verwendet.

fastNLO definiert ein eigenes Format, um die Interpolationstabellen (also die Größen $\tilde{\sigma}$) abzuspeichern. Um darauf zuzugreifen stellt *fastNLO* zwei Software Interfaces zur Verfügung. Einerseits implementiert die Klasse `fastNLOCreate` eine Schnittstelle für das Erstellen von *fastNLO*-Tabellen. Die für die Auswertung der Tabellen und das Berechnen von Wirkungsquerschnitten nötige Schnittstelle bieten die Klasse `fastNLOReader` und davon erbende Klassen.

3.2. Der Aufbau einer *fastNLO* Tabelle

Die Interpolationstabellen von *fastNLO* beginnen mit einem Tabellenkopf, der allgemeine Information zu der Tabelle enthält. Darunter fallen unter anderem:

- Information zur Software, die zur Erstellung der Tabelle verwendet wurde.
- Physikalische Parameter der Theorierechnung die zur Erstellung der Tabellen durchgeführt wurde.
- Informationen zur Observablen und deren “Binning” im Phasenraum.

Dem Tabellenkopf folgen ein oder mehrere Datenblöcke die unterschiedlichen Typs sein können. In der Regel ist ein Block mit perturbativen Berechnungen in führender Ordnung (Leading Order oder kurz LO) vorhanden. Zusätzlich können noch weitere additive Beiträge höherer Ordnung (NLO und NNLO) sowie multiplikative Korrekturbeiträge vorhanden sein. Ein Blocktyp für benutzerdefinierte Daten bietet außerdem die Möglichkeit beliebige zusätzliche Information in der Tabelle abzuspeichern. Im Rahmen dieser Arbeit werden hauptsächlich die additiven perturbativen Beiträge behandelt. Die *fastNLO* Tabellen werden in ASCII-Klartext nach einem von *fastNLO* definierten Format gespeichert und können bei Bedarf komprimiert gespeichert werden, um Festplattenspeicher zu sparen.

Ein perturbativer Beitrag kann wiederum aus mehreren Unterbeiträgen bestehen, die von unterschiedlichen Subprozessen einer Proton-Proton-Reaktion stammen. Diese Unterbeiträge werden durch eine Nummer indiziert. Zu jeder Nummer findet sich in der Tabelle eine Liste der dazu beitragenden Subprozesse. Abbildung 3.1 zeigt eine mögliche Zuordnung von Subprozessen zu Unterbeiträgen. Dort wird zum Beispiel der Gluon-Gluon-Subprozess im ersten Unterbeitrag mit Index 0 abgespeichert. Elementar für das Zusammenfassen verschiedener Subprozesse ist, das diese Subprozesse das gleiche Matrixelement besitzen. Welche Subprozesse zusammengefasst werden, wird bei der Erstellung der Tabellen entschieden.

Natürlich kann jeder Subprozess in einem einzelnen Unterbeitrag gespeichert werden. Es ist jedoch aus verschiedenen Gründen vorteilhaft mehrere Subprozesse zusammenzufassen: Der Speicherplatz den die Tabelle benötigt wird deutlich verringert, da zusammengefasste Subprozesse nur minimal mehr Speicherplatz benötigen als ein einzelner Subprozessbeitrag. Dies spart einerseits Festplattenspeicher und beschleunigt andererseits die Auswertung der Tabellen, da die Tabelle schneller geladen (und gegebenenfalls entpackt) werden kann und weniger Beiträge zum Wirkungsquerschnitt berücksichtigt werden müssen.

Eine detaillierte Definition des Tabellenformats von *fastNLO* ist im Anhang A angegeben.

3.3. Die Auswertungsschnittstelle von *fastNLO*

Um Tabellen auszulesen, wird die Funktionalität der Klasse `fastNLOReader` benötigt. Der typische Ablauf einer Tabellenauswertung läuft dabei in folgenden Schritten ab:

1. Das Öffnen der gewünschten Tabelle. Es wird ein `fastNLOReader` Objekt erzeugt, und im Konstruktor der Pfad zur Tabelle übergeben.
2. Die Auswahl der zur Berechnung von Wirkungsquerschnitten nötigen PDF. Dies geschieht in der Regel zusammen mit dem Öffnen der Tabelle durch einen Konstruktoraufruf. Für verschiedene Möglichkeiten PDFs auszuwählen existieren verschiedene von `fastNLOReader` abgeleitete Unterklassen. Ein Beispiel dafür ist die Klasse `fastNLOLHAPDF`, die die benötigten PDFs aus *LHAPDF*-Framework bezieht [10].
3. Auswahl der gewünschten Beiträge und Unterbeiträge durch Aufrufe der Funktionen `SetContributionON()` und `SelectProcesses()` (vgl. Abbildung 3.2).
4. Berechnung der Wirkungsquerschnitte durch einen Aufruf von `CalcCrossSection()`
5. Auslesen der Wirkungsquerschnitte aus dem `fastNLOReader` Objekt durch einen Aufruf von `GetCrossSection()`

Die Klasse `fastNLOReader` besitzt eine Reihe weiterer Methoden um Information aus der Tabelle auszulesen. Dies betrifft in erster Linie Information aus dem Header der Tabelle, aber auch Information über die einzelnen Beiträge. Für Anwender, die lediglich eine Übersicht über eine Tabelle benötigen, stellt das *fastNLO*-Paket das Programm `fnlo-tk-cppread` zur Verfügung, das die wichtigsten Informationen über eine geöffnete Tabelle anzeigt.

Der Quellcode von *fastNLO* ist in einem öffentlichen Git-Repository zugänglich [11]. Eine Anleitung zum Herunterladen und Kompilieren der Software findet sich im Anhang B.

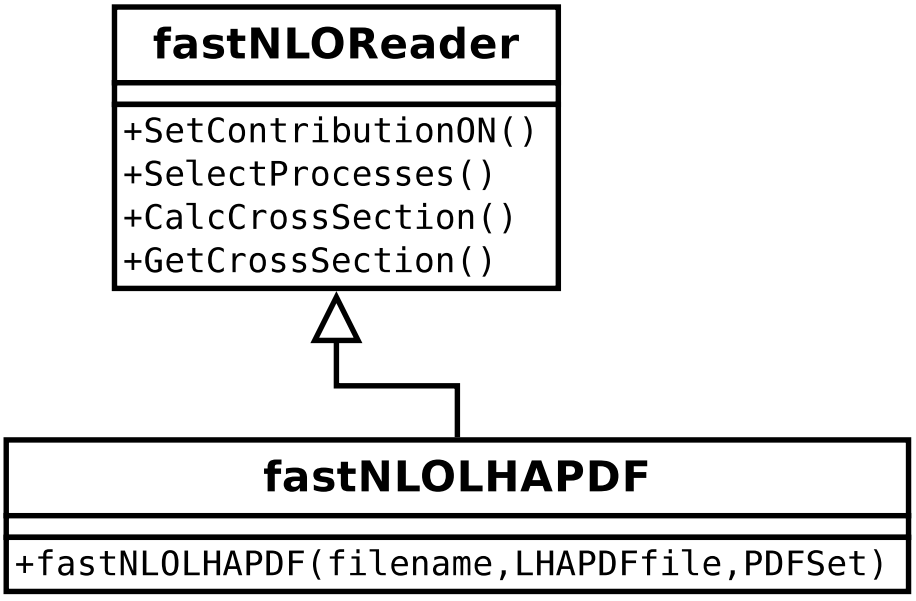


Abbildung 3.2.: Klassendiagramm (unvollständig) von fastNLOReader

4. Neue Funktionalität in fastNLO

Im Rahmen dieser Bachelorarbeit wurde zu der *fastNLO* Software neue Funktionalität zur Auswahl von Subprozessen hinzugefügt. Das Ziel der neuen Funktionen ist es, dem Anwender beim Auslesen der Tabellen die Arbeit abzunehmen, herauszufinden welche Subprozessbeiträge unter welchen Indizes in der Tabelle zusammengefasst und abgespeichert sind (vgl. Kapitel 3).

4.1. Bisherige Unterstützung von Subprozessauswahl

Wie in Kapitel 3 beschrieben, enthalten Blöcke mit additiven, perturbativen Beiträgen in den *fastNLO*-Tabellen eine Liste mit den Unterbeiträgen der einzelnen (oder bei der Erstellung der Tabelle zusammengefassten) Subprozesse. Zusätzlich dazu existiert in jedem Block die Möglichkeit, die Zuordnung der Indizes zu den beteiligten Subprozessen abzuspeichern. Die `fastNLOReader` Klasse, die für das Auslesen der Tabellen zuständig ist, erstellt für jeden Block in der Tabelle ein Objekt der Klasse `fastNLOCoeffBase`. Für die unterschiedlichen Arten von Blöcken, die abgespeichert werden können, sind verschiedene Unterklassen von `fastNLOCoeffBase` vorgesehen. Für die Subprozessauswahl wichtig ist dabei die Klasse `fastNLOCoeffAddBase`, die einen Block mit fixed Order Beiträgen verwaltet. Abbildung 4.1 zeigt das vereinfachte Klassendiagramm mit den Methoden, über die die Information zu den Subprozessen ausgelesen werden kann.

- `int fastNLOCoeffAddBase::GetNSubproc()`
  Diese Funktion gibt die Anzahl der Unterbeiträge in diesem Block an. Sind die Subprozesse einzeln abgespeichert, so entspricht dies gerade der Anzahl der beteiligten Subprozesse.
- `vector<vector<pair<int,int>>> fastNLOCoeffAddBase::GetPDFCoeff()`
  Jeder Subprozess kann durch die beiden beteiligten Partonen identifiziert werden. Das heißt durch die Angabe von zwei PdgIds[1] in einem `pair<int,int>` lässt sich ein Subprozess eindeutig bezeichnen. Diese Funktion gibt eine Liste (vector) zurück, die zu jedem Index eine Liste der enthaltenen Subprozesse liefert.

Die `fastNLOCoeffBase` Objekte, die die einzelnen Beiträge repräsentieren sind in der Auswertungsklasse `fastNLOReader` als Variablen mit Sichtbarkeit **`protected`** deklariert, das heißt der Anwender hat keinen Zugriff auf die beiden oben genannten Funktionen. Ein erstes Ziel, das im Rahmen dieser Arbeit umgesetzt wurde, ist demnach das Implementieren von Wrapper-Funktionen, die die Information über die in einer Tabelle abgespeicherten Subprozesse dem Anwender durch die Klasse `fastNLOReader` zur Verfügung stellen.

Die Berechnung der Gesamtwirkungsquerschnitte wird mit Formel (3.3) durchgeführt. Unter anderem wird über alle beitragenden Subprozesse summiert. *fastNLO* führt alle

[1] Teilchen-Identifikationsnummer der Particle Data Group [12]. In *fastNLO* wird für das Gluon jedoch statt der "offiziellen" ID 23 die 0 verwendet, was die Implementation vereinfacht.

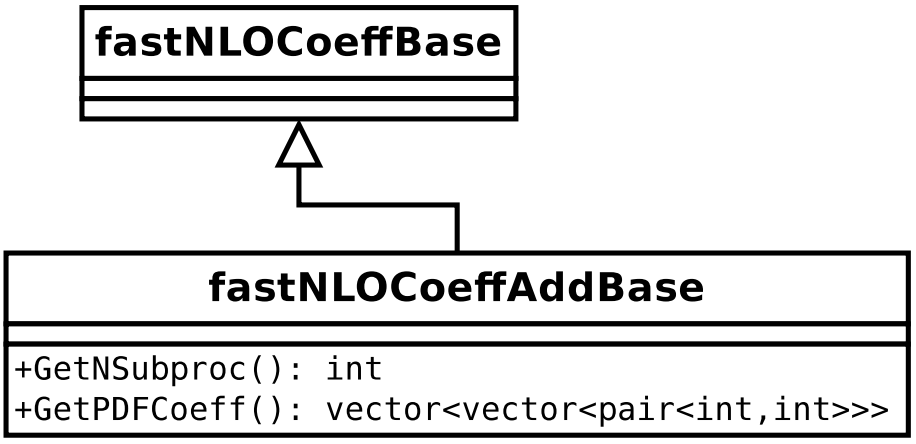


Abbildung 4.1.: (unvollständiges) Klassendiagramm von `fastNLOCoeffAddBase`

diese Summen bei einem Aufruf von `fastNLOReader::CalcCrossSection` aus. Sollen dem Anwender zusätzlich zum Gesamtwirkungsquerschnitt auch die Beiträge der einzelnen Subprozesse zum gesamten Wirkungsquerschnitt zur Verfügung gestellt werden, so muss ein Mechanismus eingeführt werden, der es ermöglicht bei der Summenbildung nur die vom Anwender gewünschten Subprozessbeiträge miteinzubeziehen. Ein erster Ansatz dazu war in Form der Funktion `SetCalculateSingleSubprocessOnly()` der Klasse `fastNLOReader` vorhanden. Mithilfe dieser Funktion lässt sich ein einzelner Unterbeitrag auswählen und bei den anschließenden Berechnungen fließt nur noch dieser eine Unterbeitrag mit ein. Wichtig ist hierbei, das die Auswahl der gewünschten Subprozessbeiträge dabei über den Index des Unterbeitrags in der Tabelle geschieht. Das bedeutet der Anwender muss folgende Arbeitsschritte durchführen, um zum Beispiel den Beitrag aller Gluon-Quark Prozesse zum gesamten Wirkungsquerschnitt auszulesen:

1. Herausfinden, welche Subprozesse unter welchen Indizes in der Tabelle abgespeichert sind. Diese Information kann entweder direkt durch den Ersteller der Tabellen übermittelt werden, oder aus der Tabelle ausgelesen werden. Im zweiten Fall muss die oben genannte Funktion `fastNLOCoeffAddBase::GetPDFCoeff()` bzw. eine Wrapper-Funktion in `fastNLOReader` aufgerufen werden.

2. Ermitteln, unter welchen Indizes Gluon-Quark Prozesse abgespeichert sind. Sind unter einem Index nicht ausschließlich Gluon-Quark Prozesse zusammen abgespeichert, dann kann der gewünschte Teilwirkungsquerschnitt nicht bestimmt werden. Dieses Problem wird später noch weitergehend behandelt werden.

3. Der Beitrag der Gluon-Quark Prozesse zum gesamten Wirkungsquerschnitt lässt sich dann wie folgt bestimmen:

```
for each index do
begin
    SetCalculateSingleSubprocessOnly(index);
    CalcCrossSection();
    result += GetCrossSection();
end
```

Die Funktionalität der Funktion `SetCalculateSingleSubprocessOnly` bietet nur in eingeschränktem Maße die Möglichkeit Teilwirkungsquerschnitte aus der Tabelle zu berechnen.

- Der oben beschriebene Arbeitsaufwand für den Anwender ist nicht unerheblich. Außerdem muss sich der Anwender mit dem Aufbau der Tabellen, speziell mit der

Indizierung beschäftigen. Die implementationsspezifische Problematik, zu welchen Indizes welche Subprozesse gehören, sollte im besten Fall in der Software-Bibliothek gelöst werden und nicht im Anwender-Code. Auf diese Art und Weise wird das Tabellenformat für den Anwender transparenter.

- Mit der Funktion `SetCalculateSingleSubprocessOnly` lässt sich immer nur ein Unterbeitrag (durch einen Index festgelegt) für die Berechnung einstellen. Sollen mehrere Prozesse zusammengefasst werden, wie im obigen Beispiel, dann muss die Addition der Teilwirkungsquerschnitte im Anwender-Code stattfinden. Dies ist ein Mehraufwand für den Anwender und kann in der Software-Bibliothek einfacher und schneller stattfinden. Es wird dann vom Anwender nur noch je ein Aufruf von `CalcCrossSection` und `GetCrossSection` benötigt.
- Die Implementation von `SetCalculateSingleSubprocessOnly` legt den verwendeten Unterbeitrag global für das `fastNLOReader`-Objekt fest. Nicht möglich ist zum Beispiel das Verwenden des Indexes 0 für den Leading Order Beitrag und gleichzeitig des Indexes 1 für den Next to Leading Order Beitrag. Das bedeutet für den Anwender, dass auch für die verschiedenen Fixed Order Beiträge die Wirkungsquerschnitte einzeln berechnet werden müssen und dann im Anwender-Code zusammenaddiert werden müssen. Dieses Problem tritt dann auf, wenn die Zuordnung Subprozesse $\leftrightarrow$ Unterbeitragindex für die verschiedenen Fixed Order Beiträge unterschiedlich ist. Eine solche Praxis beim Erstellen der Tabellen kann durchaus sinnvoll sein, da zum Beispiel einige Subprozesse gar nicht in den LO Beitrag eingehen (etwa weil die betrachteten Subprozesse in LO gar nicht existieren). Dann können alle diese nicht benötigten Subprozesse in LO unter einem Index zusammengefasst werden oder komplett weggelassen werden. In NLO dagegen werden mehrere Indizes verwendet, da die unterschiedlichen Subprozesse hier nicht mehr zusammengefasst werden können.
- `SetCalculateSingleSubprocessOnly` besitzt nur eine rudimentäre Fehlerbehandlung. So ist es zum Beispiel ohne Probleme möglich einen Index auszuwählen, der bei LO und NLO unterschiedlichen Subprozessen entspricht, oder erst gar nicht existiert. Dies kann zu, für den Anwender undurchschaubaren, falschen Ergebnissen führen.

Ein erster Ansatz während dieser Arbeit dazu war die Funktion `SetCalculateProcesses`, die eine Verallgemeinerung von `SetCalculateSingleSubprocessOnly` darstellt. Dabei können statt nur einem Index mehrere Indizes aktiviert werden. Auf diese Art kann die Addition der gewünschten Teilwirkungsquerschnitte innerhalb von *fastNLO* ausgeführt werden (2. Punkt oben). Die restlichen Probleme, die mit der rudimentären Implementation von `SetCalculateSingleSubprocessOnly` einhergehen, blieben jedoch bestehen.

Im Rahmen dieser Arbeit wird versucht diese Mängel zu beheben und dem Anwender eine transparente und möglichst einfache Methode bereitzustellen, die Beträge gewünschter Subprozesse zum Gesamtwirkungsquerschnitt aus der Tabelle auszulesen.

4.2. Das neue Anwenderinterface zur Subprozessauswahl

Um das Auslesen der Subprozessinformationen und der Teilwirkungsquerschnitte möglichst leicht und weniger fehleranfällig zu machen, wird in der `fastNLOReader` Klasse eine neue Funktion `SelectProcesses()` eingeführt. Im Idealfall wählt der Anwender dann mithilfe dieser neuen Funktion einen oder mehrere Subprozesse aus, die bei der Berechnung der Wirkungsquerschnitte miteinbezogen werden sollen. Die Funktion `SelectProcesses` sucht dann bei den aktivierten Beiträgen nach den zugehörigen Indizes und schränkt die Berechnung auf diese ein. Können die gewünschten Subprozesse nicht isoliert aktiviert werden (zum Beispiel weil sie mit anderen, nicht ausgewählten Subprozessen in der Tabelle zusammengefasst sind), dann gibt die Funktion eine Warnung zurück. Die beiden veralteten

Funktionen `SetCalculateSingleSubprocessOnly` und `SetCalculateProcesses` wurden deaktiviert und geben nun nur noch eine Warnung aus.

Werden ein oder mehrere Subprozesse ausgewählt (aktiviert), so wird diese Auswahl im `fastNLOReader` Objekt gespeichert. Auf diese Weise kann, wenn zum Beispiel mit der Funktion `SetContributionOn` ein weiterer Beitrag aktiviert wird, die Subprozessauswahl auch auf diesen Beitrag angewandt werden. Ist ein neu aktivierter Beitrag inkompatibel zu den ausgewählten Subprozessen, so schlägt die Funktion `SetContributionON` mit einer Warnung fehl.

Von der `SelectProcesses` Funktion existieren zwei überladene Varianten, von welchen je nach Anwendungsfall die eine oder die andere sinnvoller ist:

- `fastNLOReader::SelectProcesses(vector<pair<int,int>> proclist)`
- `fastNLOReader::SelectProcesses(string processes, bool symmetric = true)`

Wird eine Liste (vector) an Paaren (pair) mit PdgIds als Argument übergeben, so wird die erste Variante aufgerufen. So kann zum Beispiel mit dem Aufruf

```
Reader.SelectProcesses( { {1,1}, {-1,-1} } )
```

der Beitrag der Prozesse Down-Down und Antidown-Antidown ausgewählt werden. Wird im Anschluss daran `CalcCrossSection` aufgerufen, so werden bei der erneuten Berechnung des Wirkungsquerschnitts nur diese beiden Prozesse berücksichtigt.

Die zweite Version von `SelectProcesses` dürfte für den Standardanwender interessanter sein. Sie nimmt einen String und optional einen Bool als Argumente an. Der String beschreibt die Subprozesse, die ausgewählt werden sollen nach einem festgelegten Muster. Er besteht aus einer Reihe von, durch Leerzeichen getrennten, Substrings, von denen jeder für sich einen oder mehrere Subprozesse beschreibt. Die Gesamtauswahl wird dann durch die Vereinigung aller durch die Substrings beschriebenen Prozesse gebildet. Ein Substring hat folgende Form:

```
(none|all|[a](u|d|c|s|b|q)|g)([a](u|d|c|s|b|[!|=]q)|g)
```

Die Bedeutung dieser Buchstaben ist in Tabelle 4.1 zusammengefasst. Durch verschiedene Buchstabenkombinationen lassen sich die wichtigsten Subprozessklassen einfach für die Wirkungsquerschnitt-Berechnung auswählen. Der Wahrheitswert `symmetric` gibt an, ob die Prozesse symmetrisiert werden sollen. Ist er wahr, dann wird zu jedem ausgewählten Prozess auch der Prozess mit getauschten Partonen ausgewählt. Bei der im Rahmen dieser Arbeit untersuchten Proton-Proton Streuung ist eine Symmetrisierung der Prozessauswahl in der Regel erwünscht, da ein Prozess und der Prozess mit getauschten Partonen dasselbe Matrixelement besitzen. Es werden nun einige eventuell für den Anwender interessante Beispiele aufgeführt, an denen die Funktionsweise der Auswahlstrings in `SelectSubprocesses` noch einmal deutlich wird.

- `processes="ud dd gd"`, `symmetric=false`
  Die Prozesse Up-Down, Down-Down und Gluon-Down werden ausgewählt.
- `processes="bb asd"`, `symmetric=true`
  Diese Kombination an Argumenten führt dazu das die Prozesse Bottom-Bottom, Antistrange-Down und wegen der Symmetrisierung auch Down-Antistrange ausgewählt werden.
- `processes="qg"`, `symmetric=true`
  Das Wildcard `q` wählt alle Quarks aus, das heißt die hier ausgewählten Prozesse sind

Erster Teil des Substrings	
`none`	kein Subprozess wird ausgewählt, der zweite Teil des Substrings wird ignoriert.
`all`	alle Subprozesse werden ausgewählt, der zweite Teil des Substrings wird ignoriert.
`[a](u|d|c|s|b)`	Das durch das Kürzel spezifizierte Quark wird als erstes Parton im gewünschten Prozess ausgewählt. Der Modifizierer `a` wählt das entsprechende Antiquark aus. Zum Beispiel wird durch `d` ein Down-Quark und durch `as` ein Anti-Strange-Quark ausgewählt.
`g`	Ein Gluon wird als erstes Parton ausgewählt.
`[a]q`	Der Buchstabe `q` agiert als Wildcard für ein beliebiges Quark (bzw. Antiquark, wenn der Modifizierer `a` vorhanden ist), das heißt es werden alle Prozesse ausgewählt, deren erstes Parton ein beliebiges Quark (Antiquark) ist.
Zweiter Teil des Substrings	
`[a](u|d|c|s|b)`	Einzelnes (Anti)-Quark wird als zweites am Prozess beteiligten Parton ausgewählt.
`g`	Ein Gluon wird als zweites Prozessparton ausgewählt.
`[!|=][a]q`	Das zweite am Prozess beteiligte Parton ist beliebiges Quark bzw. Antiquark (wenn `a` vorhanden). Im Gegensatz zu dem `q` im ersten Teil des Substrings kann das zweite `q` durch zusätzliche Präfixe modifiziert werden. Ein `!` (Ungleich, an die C-Syntax angelehnt) wählt für das zweite Parton nur Quarks mit anderem Flavour, als das Erste aus, das Gleichheitszeichen `=` entsprechend nur Quarks mit gleichem Flavour.

Tabelle 4.1.: Auswahlstring bei `SelectSubprocesses`

alle Quark-Gluon nicht jedoch Antiquark-Gluon Prozesse. Wegen der Symmetrisierung werden auch alle Gluon-Quark Prozesse ausgewählt. Das gleiche Ergebnis ist mit `processes="qg gq", symmetric=false` zu erwarten.

- `processes="qq aqaq"`
  Durch das erste `q` werden alle Quarks ausgewählt und durch das zweite `q` mit wiederum allen Quarks gepaart. Der Substring `qq` wählt also alle Quark-Quark Prozesse aus. Auf die gleiche Art werden durch `aqaq` alle Antiquark-Antiquark Prozesse ausgewählt. Im Endeffekt liefert der String also alle Quark-Quark und alle Antiquark-Antiquark Kombinationen. Die Auswahl ist sowieso schon symmetrisch, das heißt das Symmetrisierungsflag hat keine Auswirkung.

- `processes="q=aq"`, `symmetric=true`
  Dieser String liefert alle Quark-Antiquark und (wegen `symmetric=true`) Antiquark-Quark Kombinationen mit gleichem Flavour, also Up-Antiup, Antiup-Up, Antidown-Down, Down-Antidown, Anticharm-Charm, usw.

- `processes="q!q"`
  Der Modifizierer `!` bewirkt, das als zweites Parton nur Quarks mit anderem Flavour als das erste ausgewählt werden. Das heißt die Auswahl besteht aus Kombinationen wie Down-Up oder Strange-Down, nicht aber Kombinationen gleichen Falvours wie z.B. Charm-Charm. Die Auswahl ist an sich schon symmetrisch, das heißt das Symmetrisierungsflag hat keine Auswirkung auf die Auswahl.

Auf diese Art und Weise kann der Anwender Subprozessbeiträge flexibel und mit relativ kleinem Aufwand auslesen. Um beispielsweise den Beitrag aller Quark-Gluon und Antiquark-Gluon Prozesse auszulesen ist folgender Code notwendig:

```
SelectProcesses("gq gaq", true);
CalcCrossSection();
result = GetCrossSection();
```

Ist der eingegebene String fehlerhaft und entspricht nicht genau dem oben angegebenen Muster, wird der Aufruf der Funktion `SelectProcesses` mit einer Warnung ignoriert. Ebenso wird der Aufruf bei einer Tabelle abgebrochen, die die ausgewählten Subprozesse nicht unterstützt, weil entweder einer der angeforderten Prozesse nicht abgespeichert ist, oder die Prozesse in der Tabelle so zusammengefasst sind, dass sie bei der Berechnung nicht wie angefordert isoliert berücksichtigt werden können.

4.3. Implementation des neuen Interfaces

Die Implementation des oben beschriebenen, neuen Anwenderinterfaces zur Subprozessauswahl beinhaltet neben der neuen Funktion `SelectSubprocess` noch einige weitere Neuerungen und Änderungen im Quellcode von *fastNLO*.

4.3.1. Die Funktion `UpdateProcesses`

Die zentrale Funktion für die neue Funktionalität ist `UpdateProcesses`. Sie ist eine `protected` Memberfunktion der `fastNLOReader` Klasse, also vom Anwender nicht direkt aufrufbar. Sie wird jedoch intern immer dann aufgerufen, wenn sich am Auswahlstatus für die Subprozesse oder Fixed Order Beiträge etwas ändert. Wie in Abbildung 4.2 skizziert wird `UpdateProcesses` von den beiden `public` also dem Anwender zur Verfügung stehenden Funktionen `SelectProcesses` für die Subprozessauswahl und `SetContributionON` für die Beitragsauswahl aufgerufen. In der Implementation wird dann von jedem aktivierten Fixed Order Beitrag die Funktion `SubSelect` mit den durch den letzten Aufruf von

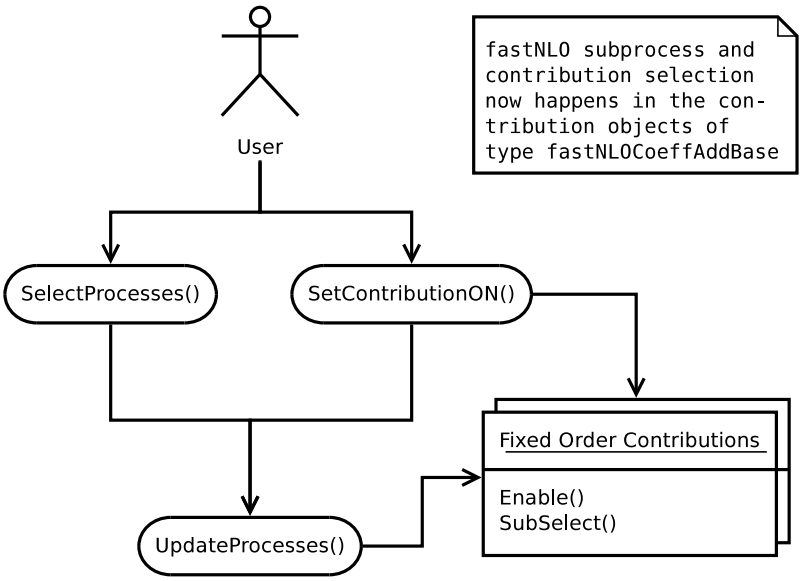


Abbildung 4.2.: Flussdiagramm zur Subprozess- und Beitragsauswahl in *fastNLO*

`SelectProcesses` ausgewählten Subprozessen aufgerufen. Die Fixed Order Beiträge, die in `fastNLOReader` durch Objekte vom Typ `fastNLOCoeffAddBase` repräsentiert werden, speichern selbstständig, ob sie aktiviert sind, und wenn ja, welche Subprozesse zur Berechnung des Wirkungsquerschnitts verwendet werden sollen.

Der Umweg über `UpdateProcesses` ist notwendig, da ein Speichern der ausgewählten Subprozesse in den Fixed Order Beitragsobjekten alleine nicht ausreicht. In den `fastNLOCoeffAddBase` Objekten wird nämlich nicht abgespeichert, welche Prozesse aktiviert wurden, sondern nur, welche Tabellenunterbeiträge (Indices) aktiviert sind (vgl. Abschnitt 3.2). Eine Rekonstruktion der ausgewählten Subprozesse ist zwar prinzipiell möglich, würde jedoch den Code unübersichtlicher und schwerer nachvollziehbar machen. Außerdem ist es möglich über `SetContributionON` alle Fixed Order Beiträge zu deaktivieren und anschließend die Subprozessauswahl zu ändern. Da die Änderung nur in aktivierte Beiträge geschrieben wird, würden die Änderungen nirgends abgespeichert und gingen verloren. Aus diesen Gründen wird die Subprozessauswahl zusätzlich in der Variable `fastNLOReader::fselected_processes` vom Typ `vector<pair<int,int>>` gespeichert. Bei jedem Aufruf von `UpdateProcesses` wird überprüft, ob die gespeicherte Subprozessauswahl für alle aktivierten Beiträge übernommen werden kann. Ist dies der Fall, wird die Subprozessauswahl über die `SubSelect` Funktion an die Fixed Order Beiträge übergeben und `UpdateProcesses` gibt den Erfolgswert `true` zurück. Kann die Subprozessauswahl nicht für alle aktivierten Beiträge übernommen werden, gibt `UpdateProcesses` `false` zurück und die aufrufende Funktion muss alle Änderungen wieder Rückgängig machen und dem Anwender eine Warnung zukommen lassen. Dieser funktionale Ablauf garantiert, dass die Subprozessauswahl in den einzelnen `fastNLOCoeffAddBase` Objekten immer synchron zu der durch den Anwender eingestellten Auswahl bleibt.

Offensichtlich muss die Funktion `UpdateProcesses` nicht nur aufgerufen werden, wenn sich die Subprozessauswahl ändert, sondern auch dann, wenn durch einen Aufruf von `SetContributionON` neue Beiträge aktiviert werden. `UpdateProcesses` sorgt dann dafür, dass auch für die neu aktivierten Fixed Order Beiträge die Unterbeiträge mit den richtigen Indices aktiviert werden. Kann dies nicht geschehen, etwa weil der neu aktivierte Beitrag die Subprozesse gröber zusammenfasst, als durch die Subprozessauswahl angefordert, schlägt auch `SetContributionON` mit einer Warnung fehl. Der Anwender muss dann zuerst die Subprozessauswahl so anpassen, dass sie mit dem neuen Beitrag kompatibel ist und dann

erneut `SetContributionON` aufrufen.

4.3.2. Die Klasse `fastNLOCoeffAddBase`

Wie in Abschnitt 3.2 beschrieben, enthalten *fastNLO*-Tabellen verschiedene Beiträge. Die Fixed Order Beiträge werden dabei in der `fastNLOReader` Klasse durch Objekte vom Typ `fastNLOCoeffAddBase` repräsentiert. Die Grundaufgabe dieser Klasse besteht darin, einen Fixed Order Beitrag aus der Tabelle auszulesen, die dazugehörigen Daten abzuspeichern und dem Reader Objekt zur Verfügung zu stellen. Zu diesen Daten gehört neben den Interpolationswerten auch die Information, welche Subprozessbeiträge unter welchen Indices abgespeichert sind.

Im Rahmen dieser Arbeit wurde der Klasse `fastNLOCoeffAddBase` weitere Funktionalität hinzugefügt. Ziel davon ist es, Code, der für jeden Fixed Order Beitrag ausgeführt werden muss, aus der `fastNLOReader`-Klasse in die `fastNLOCoeffAddBase`-Klasse auszulagern. Konkret wurden folgende neue Funktionen und Variablen eingeführt:

- `vector<bool> sub_enabled`
  Diese Variable speichert für jeden Unterbeitrag (durch seinen Index identifiziert), ob er gerade aktiv ist oder nicht. Beim Auslesen der Tabelleninformation werden alle Unterbeiträge aktiviert.
- `bool SubIsEnabled(int index)` und `SubEnable(int index, bool on = true)`
  Die obige Variable ist **`protected`**. Um auf ihren Wert auszulesen und zu verändern werden diese beiden Funktionen verwendet.
- `SubEnableAll(bool on = true)`
  Eine Funktion um alle Unterbeiträge auf einmal zu aktivieren oder zu deaktivieren.
- `bool SubSelect(vector<pair<int,int>> processes, bool on = true)`
  In dieser Funktion wird die Übersetzung Subprozesse → Unterbeiträge (Indices) vorgenommen. Die Funktion nimmt eine Liste an Subprozessen entgegen und aktiviert (bzw. deaktiviert) die entsprechenden Unterbeiträge. Der dazu notwendige Algorithmus wird im Folgenden beschrieben. Kann die angeforderte Subprozessauswahl nicht übernommen werden, wird **`false`** zurückgegeben. Die aufrufende Funktion (in der Regel `UpdateProcesses` von `fastNLOReader`) erfährt auf diesem Weg von dem Problem und kann entsprechend reagieren.

Neben diesen Änderungen in der Klasse `fastNLOCoeffAddBase` wurde in deren Basisklasse `fastNLOCoeffBase` noch eine weitere boolesche Variable eingeführt, um einen kompletten Beitrag zu aktivieren bzw. zu deaktivieren. Die hierzu notwendigen Funktionen sind:

- `bool IsEnabled()`
  Gibt zurück ob dieser Beitrag aktiviert ist. Diese Funktion wird bei der Berechnung des Wirkungsquerschnitts aufgerufen und gibt dann an, ob dieser Beitrag berücksichtigt werden soll.
- `Enable(bool on = true)`
  Aktiviert bzw. deaktiviert den Beitrag. Diese Funktion wird von `fastNLOReader::SetContributionON` aufgerufen, wenn Beiträge für die Berechnung der Wirkungsquerschnitte aktiviert oder deaktiviert werden sollen.

Diese beiden Funktionen wurden in der Basisklasse eingeführt, da somit auch nicht additive Beiträge gesondert aktiviert und deaktiviert werden können. Diese Funktionalität ist nicht neu in *fastNLO* wurde aber im Zuge der neuen Subprozessfunktionalität aus der Reader Klasse in die `fastNLOCoeffBase` Klasse ausgelagert um, dem objektorientierten Denken entsprechend, den Zustand (aktiviert oder deaktiviert) eines Objektes direkt im betreffenden Objekt zu speichern.

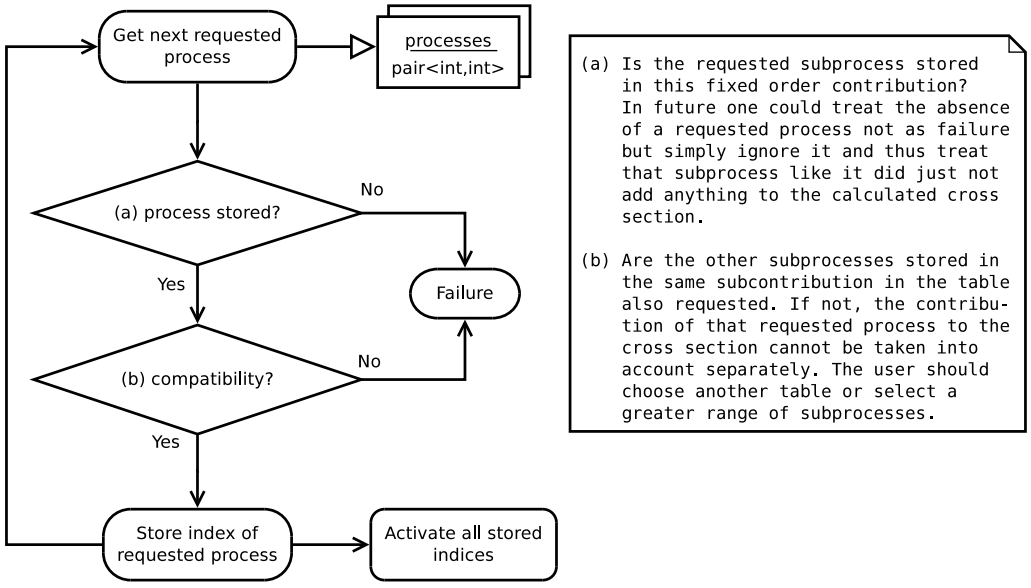


Abbildung 4.3.: Algorithmus von `SubSelect`

Implementation von `SubSelect`

Das Problem, das in der Funktion `fastNLOCoeffAddBase::SubSelect` gelöst wird, ist die Zuordnung von Subprozessen zu den jeweiligen Unterbeiträgen. In jedem Unterbeitrag können mehrere Subprozessbeiträge gespeichert sein. `SubSelect` nimmt zwei Parameter entgegen:

- `vector<pair<int,int>> processes`
- `bool on`

Der erste Parameter gibt die gewünschte Prozessauswahl an und besteht aus einer Liste an Integer-Paaren, die über PdgIds jeweils einen Subprozess identifizieren. Der zweite Parameter gibt an, ob die Prozessauswahl aktiviert oder deaktiviert werden soll. Jedes Objekt vom Typ `fastNLOCoeffAddBase` hat außerdem über die Membervariable `fPDFCoeff` Zugriff auf die Information unter welchem Index welche Subprozesse gespeichert sind. `fPDFCoeff` ist eine Liste mit so vielen Elementen wie Unterbeiträge in dem Beitrag existieren. Jedes Listenelement wiederum ist eine Liste von Integerpaaren, die angeben welche Subprozesse unter diesem Unterbeitrag (Index) enthalten sind.

Der Algorithmus, der von `SubSelect` verwendet wird ist in Abbildung 4.3 skizziert. Ein gewünschter Prozess wird aus der übergebenen Liste `processes` entnommen. Anschließend wird dieser Prozess in `fPDFCoeff` gesucht (Schritt (a) in Abbildung 4.3). Ist er nicht vorhanden, dann ist der Subprozessbeitrag nicht in diesem Fixed Order Beitrag vorhanden. In der aktuellen Version wird das als Fehler interpretiert. In Zukunft könnte ein fehlender Prozess auch einfach ignoriert werden, sodass *fastNLO*-Tabellen erstellt werden können, die nur diejenigen Subprozessbeiträge enthalten, die ungleich Null sind. Das Ignorieren eines fehlenden Prozesses führt zum gleichen Ergebnis, wie ein Subprozessbeitrag, der nur Nullen enthält. Wird der Prozess in `fPDFCoeff` gefunden, dann kann es sein, das unter dem selben Index noch andere Subprozessbeiträge gespeichert sind. Der in Abbildung 4.3 als (b) bezeichnete Schritt untersucht, ob alle diese Subprozessbeiträge in der Liste `processes` mit der gewünschten Subprozessauswahl enthalten sind. Ist dies der Fall, dann kann der Index aktiviert werden, ansonsten ist die Tabelle (bzw. der aktuelle Beitrag) zu grob für die gewünschte Subprozessauswahl. Ein Fehler wird zurückgegeben und der Anwender muss entweder eine Tabelle mit einer feineren Subprozessunterteilung verwenden,

oder seine Subprozessauswahl so vergrößern, dass immer alle gemeinsam unter einem Index abgespeicherten Subprozesse enthalten sind.

Die Aufgabe, die `SubSelect` übernimmt, ist die zentrale Problemstellung, die dem Anwender in der neuen Funktionalität abgenommen wird. Auf diese Art und Weise wird das interne Speichersystem von *fastNLO* so gut wie möglich vor dem Anwender verborgen. Trotzdem ist es notwendig, dass der Anwender das Prinzip der zu Unterbeiträgen zusammengefassten Subprozessen versteht, um die richtige Tabelle für seine Anwendung auszusuchen.

5. Anwendungsbeispiel

Zur Demonstration der neu implementierten Funktionalität soll an dieser Stelle noch ein Anwendungsbeispiel angeführt werden. Betrachtet wird dabei der Wirkungsquerschnitt des Proton-Proton zu Z+Jet Prozesses bei einer Schwerpunktsenergie von 13 TeV.

5.1. Der betrachtete Prozess

Stoßen zwei Protonen aufeinander, so können eine Vielzahl an neuen Teilchen produziert werden. In diesem Beispiel wird der Prozess betrachtet, bei dem zwei Partonen der Protonen zu einem Z-Boson und einem Jet reagieren. Der Wirkungsquerschnitt eines solchen Prozesses kann (wie alle Hadronischen Prozesse) über das Faktorisierungstheorem (2.1) berechnet werden. Der Gesamtprozess setzt sich also additiv aus den einzelnen Parton-Parton zu Jet+Z Prozessen mit allen möglichen Parton-Parton Kombinationen zusammen. Die Beiträge zum Gesamtwirkungsquerschnitt sind jedoch je nach Richtung und Impuls der entstehenden Teilchen unterschiedlich.

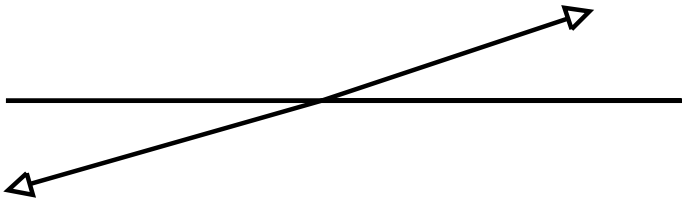
Ein Teilchen, das als Produkt aus der Kollision kommt, besitzt einen Viererimpuls, aus dem sich verschiedene andere, leichter zu messende Größen berechnen lassen. Dabei wird die z-Achse des Koordinatensystems in die Strahlachse der kollidierenden Protonenstrahlen gelegt. Der Impuls eines Teilchens lässt sich in eine Komponente p_z parallel zur Strahlachse und eine Transversalkomponente p_T senkrecht zur Strahlachse aufteilen. Der azimutale Winkel ϕ des Impulses wird nicht weiter betrachtet, da die Anordnung rotationssymmetrisch ist. Mittels p_z wird schließlich die kinematische Größe der Rapidität y definiert.

$$y = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right) \tag{5.1}$$

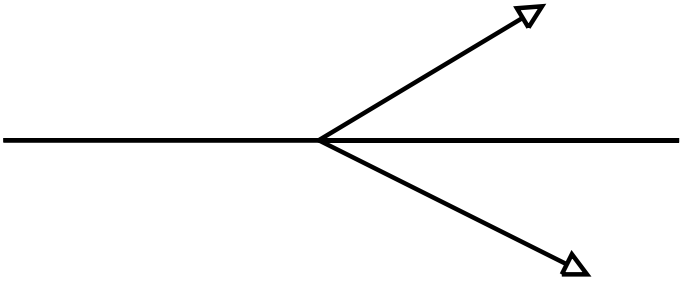
Eine Rapidität von $y = 0$ entspricht einem Wert von $p_z = 0$ und damit einer Flugbahn senkrecht zur Strahlachse. Je stärker die Rapidität von Null abweicht, desto kleiner ist der zwischen der Bahn des Teilchens und der Strahlachse eingeschlossene Polarwinkel θ. Tatsächlich gilt für Teilchen mit $E \gg m$ ein Zusammenhang zu diesem Winkel:

$$y \approx -\ln \left(\tan \frac{\theta}{2} \right) \tag{5.2}$$

Im hier betrachteten Prozess entstehen ein Z-Boson und ein Jet. Diese beiden Objekte tragen jeweils einen Transversalimpuls, der aufgrund der Impulserhaltung in transversaler Richtung betragsmäßig gleich groß sein muss. Außerdem lassen sich die beiden Rapiditäten y_Z und y_{jet} angeben. Es stellt sich jedoch heraus, dass es zweckmäßiger ist, die Summe und die Differenz der beiden Rapiditäten zu betrachten.



(a) Stoßsituation bei kleinem $|y_b|$ und großem $|y^*|$



(b) Stoßsituation bei großem $|y_b|$ und kleinem $|y^*|$

Abbildung 5.1.: Räumliche Bedeutung der Größen y_b und y^*

$$y_b := \frac{1}{2}\left(y_Z + y_{\text{jet}}\right) \tag{5.3}$$

$$y^* := \frac{1}{2}\left(y_Z - y_{\text{jet}}\right) \tag{5.4}$$

Die Größe y^* ist invariant unter Lorentzboosts in die z-Richtung. Im Schwerpunktsystem der beiden entstehenden Teilchen unterscheiden sich die Rapiditäten nur um ein Vorzeichen. Das bedeutet, y^* hängt über Gleichung (5.2) mit dem Winkel der Teilchenbahnen zur Strahlachse im Schwerpunktsystem zusammen.Die Größe y_b dagegen hängt mit der Geschwindigkeit des Schwerpunktsystems zusammen und damit auch mit dem Winkel zwischen den beiden Teilchenbahnen im Detektorsystem. In diesem Beispiel werden zwei Phasenraumbereiche in y_b und y^* betrachtet.

- $|y_b| < 0.5\ , \quad 1.5 \leq |y^*| < 2.0$
  In diesem Phasenraumbereich finden Ereignisse statt, bei denen sowohl das Z-Boson, als auch der Jet hohe Rapiditäten in unterschiedliche Richtung haben. Abbildung 5.1a zeigt eine mögliche Anordnung eines solchen Ereignisses.
- $1.5 \leq |y_b| < 2.0\ , \quad |y^*| < 0.5$
  In diesem Phasenraumbereich finden Ereignisse statt, in denen sowohl das Z-Boson, als auch der Jet hohe Rapiditäten haben, aber die Rapiditäten das gleiche Vorzeichen haben. Das Schwerpunktsystem ist also stark geboostet und im Detektorsystem sieht das Ereignis wie zum Beispiel in Abbildung 5.1b dargestellt aus.

Zusätzlich zu der Phasenraumeinschränkung auf die beiden obigen Bereiche in $|y_b|$ und $|y^*|$ wird der Wirkungsquerschnitt differentiell im Transversalimpuls $p_{T,Z}$ des Z-Bosons (und damit auch des Jets) betrachtet.

5.2. Auswertung der Tabellen

Die für diese Beispielanwendung verwendeten *fastNLO*-Tabellen wurden mithilfe des *MCgrid*-Frameworks [13, 14] erstellt. Dieses Softwarepaket ermöglicht die Berechnung von

fastNLO-Tabellen in NLO. *MCgrid* verwendet den Monte-Carlo-Ereignisgenerator *Sherpa*[15, 16] sowie das Analyseprogramm *Rivet*[17, 18]. Zur Untersuchung der zum Wirkungsquerschnitt beitragenden Subprozesse wird die *Python*-Schnittstelle von *fastNLO* benutzt, um die (Teil-)Wirkungsquerschnitte aus den Tabellen zu berechnen.

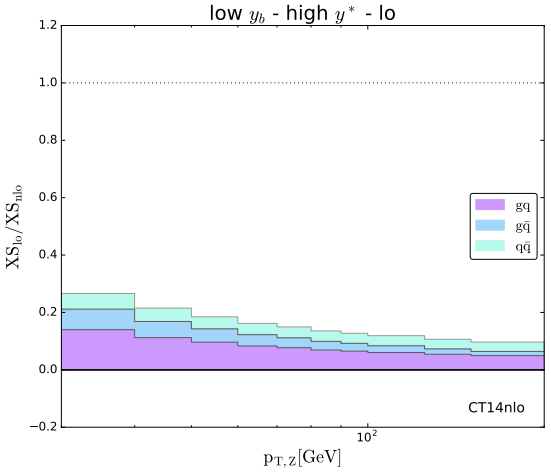
Es werden jeweils nur Anteile der Teilwirkungsquerschnitte am NLO-Gesamtwirkungsquerschnitt betrachtet, um die Zusammensetzung des Prozesses zu untersuchen. Neben den Anteilen der NLO-Wirkungsquerschnitte der einzelnen Subprozesse werden auch die Anteile der LO-Wirkungsquerschnitte am NLO-Gesamtwirkungsquerschnitt ausgerechnet. Dadurch kann Information darüber gewonnen werden, wie gut die Näherung in führender Ordnung im Vergleich zur nächsten Ordnung in α_s ist. Quantifiziert wird diese Größe durch den sogenannten k-Faktor, das Verhältnis von NLO- zu LO-Wirkungsquerschnitt.

Abbildung 5.2 zeigt die daraus resultierenden Grafiken. Oben (Abbildungen 5.2a und 5.2b) sind für beide Phasenraumbereiche die LO-Wirkungsquerschnitte anteilig am NLO-Gesamtwirkungsquerschnitt eingezeichnet. Die weißen Flächen unterhalb der 1 entsprechen gerade dem k-Faktor. Zu sehen sind die Beiträge der einzelnen Subprozesse, wobei berücksichtigt werden muss, das in LO nur Gluon-Quark, Gluon-Antiquark und Quark-Antiquark Subprozesse beitragen können, da nur zu diesen Subprozessen Feynman-Graphen in LO existieren. Offensichtlich ist der k-Faktor im Falle eines niedrigen y_b und eines hohen y^* deutlich größer als im anderen Fall.

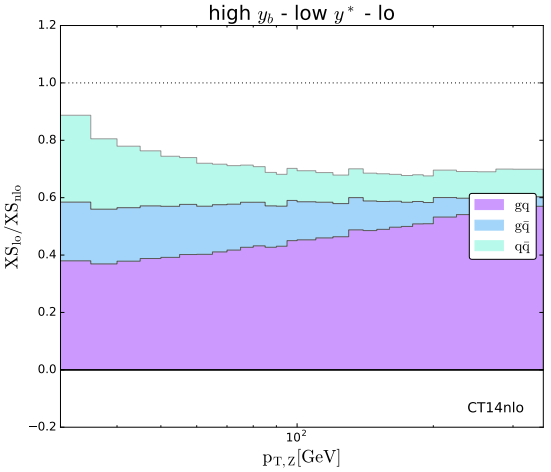
Die Abbildungen 5.2c und 5.2d zeigen die NLO-Wirkungsquerschnitte der einzelnen Subprozesse anteilig am Gesamtwirkungsquerschnitt. Aus diesen Bildern lässt sich also die Zusammensetzung des Prozesses ablesen. Auffällig ist die Dominanz des Gluon-Quark- über den Gluon-Antiquark-Subprozess in beiden Phasenraumregionen, vor allem aber bei hohem y_b und hohem $p_{T,Z}$. Dieses Phänomen lässt sich den Valenzquarks zuschreiben, die mit größerer Wahrscheinlichkeit mit hohen Impulsanteilen im Proton anzutreffen sind, als andere Quarks und Antiquarks.

Die Valenzquarks können auch den nur bei niedrigem y_b und hohem y^* nennenswert auftretenden Beitrag des Quark-Quark-Prozesses (orange in Abbildung 5.2c) erklären. Um ein niedriges y_b zu erhalten, müssen zwei Partonen mit ähnlichem Impuls zusammenstoßen. Da gleichzeitig y^* groß ist, müssen beide Partonen großen Impuls tragen. Die Wahrscheinlichkeit zwei Partonen mit ähnlichem aber großen Impulsanteil in den Protonen zu finden, ist für zwei Valenzquarks größer, als für andere Partonpaare. Mit zunehmenden $p_{T,Z}$ verstärkt sich dieser Effekt noch, da noch größere Impulse in den Partonen benötigt werden. Dieser Zusammenhang ist an der anteiligen Zunahme des Quark-Quark-Prozesses mit zunehmendem $p_{T,Z}$ in Abbildung 5.2c gut zu erkennen. Der Quark-Quark-Prozess existiert in LO nicht, und liefert damit einen nicht unbedeutenden Beitrag zu dem großen k-Faktor, angedeutet durch die große weiße Fläche in Abbildung 5.2a, in diesem Phasenraumbereich.

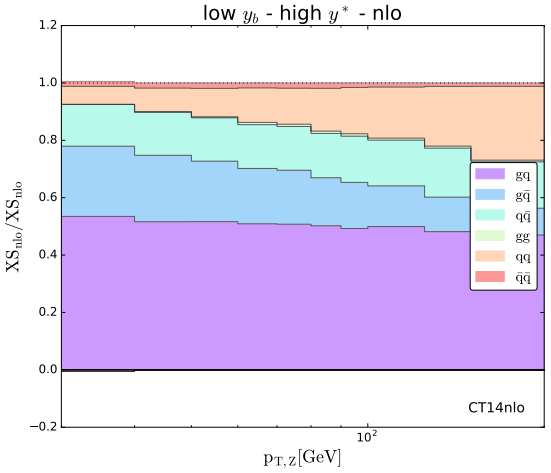
Im Falle eines hohen y_b und niedrigen y^* (Abbildung 5.2d) werden zwei Partonen mit unterschiedlichem Impuls benötigt. Dies ist sehr viel wahrscheinlicher, als das Aufeinandertreffen zweier Partonen mit ähnlichem Impuls, was sich in einem um etwa einen Faktor 20 größeren absoluten Wirkungsquerschnitt bemerkbar macht (vgl. Abbildungen 5.2e und 5.2f). Mit steigendem $p_{T,Z}$ werden für den Prozess Partonen mit immer größerem Impuls benötigt. Auch hier zeigen sich die Valenzquarks als Lieferanten hoher Impulswerte: Der Gluon-Quark-Prozess nimmt anteilsmäßig zu, hauptsächlich zulasten des Gluon-Antiquark-Prozesses.



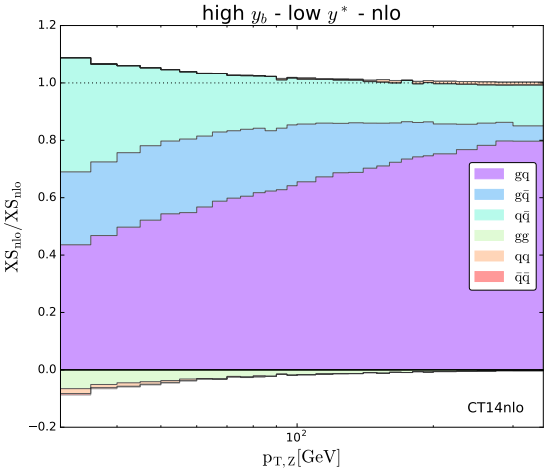
(a) Anteilige LO Wirkungsquerschnitte am NLO-Gesamtwirkungsquerschnitt bei niedrigem y_b und hohem y^*



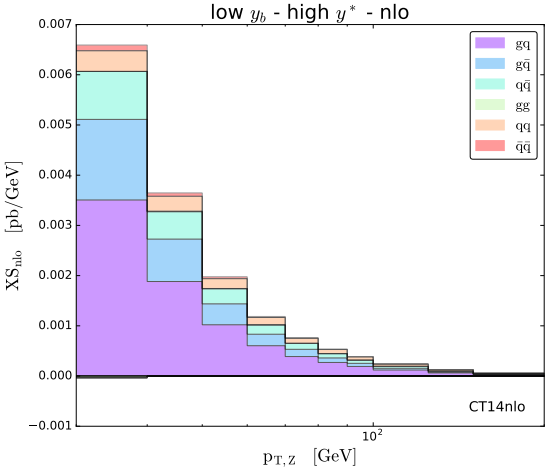
(b) Anteilige LO Wirkungsquerschnitte am NLO-Gesamtwirkungsquerschnitt bei hohem y_b und niedrigem y^*



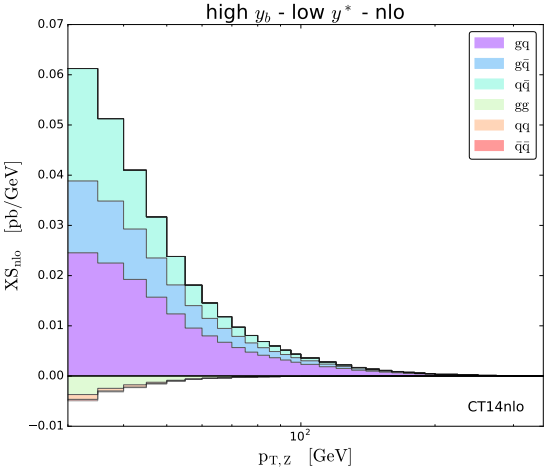
(c) Anteilige NLO Wirkungsquerschnitte am NLO-Gesamtwirkungsquerschnitt bei niedrigem y_b und hohen y^*



(d) Anteilige NLO Wirkungsquerschnitte am NLO-Gesamtwirkungsquerschnitt bei hohem y_b und niedrigem y^*



(e) NLO-Gesamtwirkungsquerschnitt bei niedrigem y_b und hohen y^*



(f) NLO-Gesamtwirkungsquerschnitt bei hohem y_b und niedrigem y^*

Abbildung 5.2.: Zusammensetzung des Z+Jet Prozesses

6. Zusammenfassung

Die neue Funktionalität in *fastNLO* wurde in Abschnitt 5 an NLO-Tabellen zu Z+Jet Analysen getestet. Die Subprozessbeiträge zum Gesamtwirkungsquerschnitt lassen sich einfacher und transparenter als zuvor aus den *fastNLO* Tabellen ausrechnen. Informationen über die Zusammensetzung von Gesamtwirkungsquerschnitten aus den einzelnen Subprozessen können ohne Wissen über die innere Struktur und das Speicherformat der Tabellen berechnet werden.

Es können Situationen auftreten, in denen bestimmte Unterbeiträge Null sind. Dies kommt zum Beispiel dann vor, wenn in einer bestimmten Störungsordnung ein Subprozess nicht stattfinden kann (etwa Quark-Quark in LO). In solchen Fällen können die nicht beitragenden Subprozesse auf unterschiedliche Art und Weise in der Tabelle abgespeichert sein:

1. Einzeln, aber Null enthaltend. Die Beiträge können dann addiert werden, ohne dass sich am Ergebnis etwas ändert. Diese Lösung ist insofern unbefriedigend, als das auch für nicht zum Ergebnis beitragende Subprozesse Speicherplatz in der Tabelle benötigt wird.
2. Zusammengefasst, aber Null enthaltend. Auf diese Art wird deutlich weniger Speicherplatz verwendet, als wenn die Subprozesse einzeln abgespeichert wird.
3. Gar nicht. Da die Subprozesse nichts zum Gesamtwirkungsquerschnitt beitragen können sie komplett aus dem Tabellenbeitrag weggelassen werden. Diese Methode benötigt am wenigsten Speicher und stellt somit die beste Speicherkonvention dar. Allerdings wird diese Speichermethode bisher kaum verwendet.

Die Implementation des neuen Subprozessinterfaces ist auf vollständige Tabellen hin programmiert, das heißt, wenn ein angeforderter Subprozess nicht vorhanden oder mit nicht ausgewählten Subprozessen zusammen abgespeichert ist, wird ein Fehler ausgegeben und die Auswahl abgebrochen. Dieses Verhalten ist ungünstig, falls die nicht beitragenden Subprozesse nach den obigen Konventionen 2 oder 3 abgespeichert sind, da dann unter Umständen die Subprozessauswahl wegen der nicht beitragenden Subprozesse fehlschlägt. Die Konvention 1 dagegen ist sehr speicheraufwändig und sollte langfristig durch die beiden anderen Speichermethoden ersetzt werden. Dazu ist dann eine Anpassung der Implementation der Subprozessauswahl erforderlich, die Nullbeiträge und nicht vorhandene Beiträge einfach ignoriert. Eine solche Anpassung darf aber auf der anderen Seite nicht dazu führen, dass ein Anwender erwartet einen bestimmten ausgewählten Subprozessbeitrag zu sehen, obwohl dieser Beitrag in der Tabelle nicht vorhanden ist. Das heißt in den Fällen, in denen ein nicht vorhandener Subprozessbeitrag ignoriert wird, sollte *fastNLO* zumindest eine Warnung ausgeben, um den Anwender zu warnen. Eine andere Möglichkeit wäre eine weitere Option in den Subprozessauswahlroutinen, um das Ignorieren von nicht vorhandenen Beiträgen manuell an- bzw. auszuschalten.

In dieser Arbeit wurde die Subprozessauswahl nur für Tabellen implementiert, die die Subprozessinformation auf eine bestimmte Art und Weise abspeichern (IPDFdef2=0 in Tabelle A.7). Die Definition des Tabellenformats lässt jedoch auch Tabellen zu, die die Subprozessinformationen anders oder gar nicht abspeichern (vgl. Tabellenformat im Anhang A). Im Rahmen dieser Arbeit wurden solche Tabellen nicht weiter untersucht und können mit der aktuellen Version der Subprozessauswahl nicht verwendet werden. In Zukunft kann versucht werden, die Implementation des Subprozessinterfaces auf diese anderen Formate der Subprozessinformationen zu verallgemeinern.

Anhang

A. Tabellenformat von *fastNLO*

Das Tabellenformat, in dem *fastNLO* abgespeichert wird, ist ein Klartextformat, das heißt alle Daten lassen sich im Prinzip auch mit einem normalen Texteditor ansehen und bearbeiten. Der grundsätzliche Aufbau ist in Tabelle A.1 beschrieben.

Tabelle A.1.: Aufbau einer *fastNLO* Tabelle

Header Teil 1 (Tabelle A.2)	Der erste Teil des Tabellenheaders gibt Information über die verwendete Tabellenversion und legt die Anzahl der Beiträge (siehe unten) fest.
Header Teil 2 (Tabelle A.3)	Der zweite Teil des Headers beschreibt das Szenario und das Binning der Observablen.
Datenblöcke (Tabelle A.4)	Diese Blöcke, deren Anzahl und Typ im Header Teil 1 festgelegt wurde, enthalten die tatsächlichen Daten.

Tabelle A.2.: Header Teil 1 - *fastNLO* Version

MagicNumber	string	immer “1234567890”
Itabversion	int	Tabellenversion mit 10000 multipliziert (z.B. 23000 für Version 2.3)
ScenName	string	Name des Szenarios
Ncontrib	int	Anzahl der Beiträge (sowohl multiplikative als auch additive)
Nmult	int	Anzahl der multiplikativen Beiträge
Ndata	int	Anzahl der Datenblöcke in der Tabelle
NUserBlocks	int	Anzahl der Benutzerblöcke
UserBlocks	UserBlock[NUserBlocks]	Benutzerblöcke (siehe Nutzerblockdefinition in Tabelle A.7)

Tabelle A.3.: Header Teil 2 - Beschreibung des Szenarios

MagicNumber	string	immer “1234567890”
Ipublunits	int	negative Zehnerpotenz der Einheit der Wirkungsquerschnitte
NscDescript	int	Zeilenzahl der folgenden Szenariobeschreibung
scDescript	string[NscDescript]	Szenariobeschreibung
Ecms	double	Schwerpunktsenergie der Kollisionen
ILOord	int	Ordnung in α_s des LO Beitrages
NObsBin	int	Gesamtanzahl der Observablen Bins
NDim	int	Anzahl der Dimensionen des Binnings
DimLabel	string[NDim]	Namen der einzelnen Dimensionen
IDiffBin	int[NDim]	Ein Flag, das das Format der folgenden Binning-Information festlegt
BinCenter	double[NObsBin][NDim]	Die jeweiligen Binpositionen im Phasenraum (falls IDiffBin=1)
BinBound	double[2][NObsBin][NDim]	Die jeweiligen Bingrenzen im Phasenraum (falls IDiffBin=2)
BinSize	double[NObsBin]	Bingröße des jeweiligen Bins
INormFlag	int	Normalisierungsflag = 0 (zur Zeit nicht verwendet)

Tabelle A.4.: Beitragsblock - Beschreibung eines Beitrags

MagicNumber	string	immer “1234567890”
IXsectUnits	int	negative Potenz der Einheit der Wirkungsquerschnitte dieses Beitrags
IDataFlag	int	0: Theoriebeitrag 1: Datenblock
IAddMultFlag	int	0: additiver Beitrag 1: multiplikativer Beitrag
IContrFlag1	int	1: perturbativer Beitrag 2: Threshold Korrektur 3: elektroschwache Korrektur 4: nicht-perturbativer Korrekturfaktor
IContrFlag2	int	1: LO-Beitrag 2: NLO-Beitrag 3: NNLO-Beitrag
NscaleDep	int	0: Beitrag ohne Skalenabhängigkeit 3: SigmaIndep, SigmaMuf und SigmaMur sind vorhanden 4: Nur SigmaMuIndep ist gespeichert 5: SigmaMuIndep, SigmaMur und SigmaMuf vorhanden 6: Zusätzlich ist noch SigmaMuRF vorhanden
NContrDescr	int	Zeilenzahl der folgenden Beitragsbeschreibung
CtrbDescript	string[NContrDescr]	Beitragsbeschreibung
NCodeDescr	int	Zeilenanzahl der folgenden Codebeschreibungen
CodeDescript	string[NCodeDescr]	Beschreibung des Codes, der für die Berechnungen verwendet wurde
Payload	Blockdaten	Die tatsächlichen Daten dieses Beitrags. Der Typ wird in den obigen Flags festgelegt. Die Formate der einzelnen Daten sind in den Tabellen A.5, A.6 und A.7 festgelegt.
NUserBlocks	int	Anzahl der Benutzerblöcke
UserBlocks	UserBlock[NUserBlocks]	Benutzerblöcke (siehe Tabelle A.8)

Tabelle A.5.: Blockdaten - multiplikativer Beitrag

Nuncorrel			int	Anzahl an unkorrelierten Unsicherheitsquellen
UncDescr			string[Nuncorrel]	Beschreibungen der Unsicherheiten
Ncorrel			int	Anzahl an korrelierten Unsicherheiten
CorDescr			string[Ncorrel]	Beschreibung der Unsicherheiten
NObsBin	fact		double	multiplikativer Faktor
	Nuncorrel	UnCorLo	double	untere Unsicherheitsgrenze
		UnCorHi	double	obere Unsicherheitsgrenze
	Ncorrel	CorrLo	double	untere Unsicherheitsgrenze
		CorrHi	double	obere Unsicherheitsgrenze

Tabelle A.6.: Blockdaten - experimentelle Daten

Nuncorrel			int	Anzahl der unkorrelierten Unsicherheiten
UncDescr			string[Nuncorrel]	Beschreibungen der Unsicherheiten
Ncorrel			int	Anzahl der korrelierten Unsicherheiten
CorDescr			string[Ncorrel]	Beschreibungen der Unsicherheiten
NObsBin	Xcenter		double	Bin Zentrum in der letzten Dimension
	value		double	Wirkungsquerschnitt
	Nuncorrel	UnCorLo	double	untere Unsicherheitsgrenze
		UnCorHi	double	obere Unsicherheitsgrenze
	Ncorrel	CorrLo	double	untere Unsicherheitsgrenze
		CorrHi	double	obere Unsicherheitsgrenze
NerrMatrix			int	Anzahl der vorhandenen Fehlermatrizen
EMatrix			double[NerrMatrix] [NObsBin^2]	Elemente der Fehlermatrix

Tabelle A.7.: Blockdaten - additiver Beitrag

IRef	int	0: Standard Tabellenbeitrag 1: Referenzeintrag
IScaleDep	int	Flag für die Skalenabhängigkeit
Nevt	int	Anzahl an Events
Npow	int	absolute Ordnung in α_s
NPDF	int	Anzahl der beim Prozess beteiligten PDFs
NFragFunc	int	Anzahl der beteiligten Fragmentierungsfunktionen (FF)
NFFPDG	int[NFragFunc]	PDF Codes der FFs
NFFDim	int	Flag, das angibt, wie die FFs gespeichert sind
NSubproc	int	Anzahl der partonischen Subprozesse
IPDFdef1	int	1. Flag zur Definition des Subprozesslayouts: 0: nicht weiter spezifiziert 1: e^+e^- 2: inclusive DIS 3: hh / $h\bar{h}$ 4: verschiedene Hadronen
IPDFdef2	int	2. Flag zur Definition des Subprozesslayouts: 0: einzelne Partonkombinationen 1: NC DIS (falls IPDFdef1=2), $h\bar{h}$-jets (3), resolved gamma-p (4) 2: direct gamma-p (2), $t\bar{t}$ (3) 121: 11x11 Standard-Kombinationen 169: 13x13 Standard-Kombinationen

IPDFdef3				int	3. Flag zur Definition des Subprozesslayouts: falls IPDFdef1=2 und IPDFdef2=1 - 1: 1 Subprozess (incl DIS LO: Delta) - 2: 2 Subprozesse (DIS jets LO: Delta, Gluon) - 3: 3 Subprozesse (DIS jets NLO: Delta, Gluon, Sigma) falls IPDFdef1=3 und IPDFdef2=1 - 1: 6 Subprozesse - 2: 7 Subprozesse - 3: 10 Subprozesse falls IPDFdef1=3 und IPDFdef2=2 - 0: 2 Subprozesse (gg, qq) falls IPDFdef2=0 =NSubproc
IPDFdef2=0	IPDFCoeffFormat			int	=0
	NSubproc	NpartonPairs		int	Anzahl der Partonpaare in diesem Unterbeitrag
		Nparton Pairs	PDF1Flavour	int	PDGId des ersten Partons
			PDF2Flavour	int	PDGId des zweiten Partons
NEvtBinProc				int[NObsBin] [NSubproc]	Anzahl an Events in diesem Bin und Subprozessbeitrag
NObsBin	Nxtot1			int	Anzahl der PDF Stützstellen
	XNode1			int[Nxtot1]	x-Werte der Stützstellen
NObsBin	Nxtot2			int	Anzahl der Stützstellen der zweiten PDF (nur falls NPDFdim=2)
	XNode2			int[Nxtot2]	Stützstellen der zweiten PDF
NObsBin	Nztot			int	Anzahl der Knoten in der FF-Matrix
	Znode			int[Nztot]	Stützstellen der FFs
NScales				int	Anzahl der Skalen
NScaleDim				int	Anzahl der Dimensionen, in denen Skalen gespeichert sind
Iscale				int[NScales]	Dimensionen, in denen die Skalen gespeichert sind
Nscale Dim	NscaleDescript			int	Zeilenzahl der Skalenbeschreibung
	ScaleDescript			string[NscaleDescript]	Beschreibungen der Dimensionen
nur wenn NscaleDep=0	Nscale Dim	Nscalevar		int	Anzahl der Skalenvariationen
		Nscalenode		int	Anzahl der Skaleninterpolationspunkte
	ScaleFac			double[Nscaledim] [Nscalevar]	Skalenvariationsfaktor
	ScaleNode			double[NObsBin] [Nscaledim] [Nscalevar] [Nscalenode]	Werte der Stützstellen der Skalenvariation

	SigmaTilde		double[NObsBin] [Nscaledim] [Nscalevar] [Nscalenode] [nmax] [Nsubproc]	Werte der Interpolationspunkte
NscaleDep>=3	Nscalenode1		int	Anzahl der Stützstellen für Skala 1
	ScaleNode1		double[NObsBin] [Nscalenode1]	Stützstellen für Skala 1
	Nscalenode2		int	Anzahl der Stützstellen für Skala 2
	ScaleNode2		double[NObsBin] [Nscalenode2]	Stützstellen für Skala 2
	SigmaMuIndep		double[NObsBin] [nxmax] [Nscalenode1] [Nscalenode2] [Nsubproc]	Skalenunabhängige Beiträge
	>=5	SigmaMuFDep	double[...]	$\log(\mu_f)$-abhängige Beiträge
		SigmaMuRDep	double[...]	$\log(\mu_r)$-abhängige Beiträge
	>=6	SigmaMuRRDep	double[...]	$\log^2(\mu_r)$-abhängige Beiträge
	>=7	SigmaMuFFDep	double[...]	$\log^2(\mu_f)$-abhängige Beiträge
		SigmaMuRFDep	double[...]	$\log(\mu_r)\log(\mu_f)$-abhängige Beiträge

Tabelle A.8.: Nutzerblock - Enthält Benutzerdefinierte Daten

IUserFlag	int	Ein Flag, das die angegeben Daten charakterisiert 0: undefiniert 1: Theoretische Unsicherheit pro Bin 2: Theoretische Unsicherheit pro Bin und Subprozess 3: Zusätzliche Blockbeschreibung
NUserBlockDescr	int	Zeilenzahl an Beschreibungen dieses Blocks
UserBlockDescr	string[NUserBlockDescr]	Beschreibung
IUserLines	int	Anzahl an Zeilen in diesem Block
Content	string[IUserLines]	Benutzerdefinierter Inhalt

B. Anleitung zum Kompilieren des *fastNLO*-Codes

Der Quellcode von *fastNLO* ist öffentlich in einem Git-Repository zugänglich [11]. Die folgenden Anleitungsschritte sollen einen kurzen Überblick geben, wie der Code heruntergeladen, kompiliert und installiert werden kann. Damit der Code ohne Fehlermeldungen kompiliert, sollten folgende Abhängigkeiten schon vorhanden/installiert sein:

- Ein C++11 kompatibler Compiler (z.B. GCC ab Version 4.8.1)
- Swig[1] falls das Python-Interface aktiviert werden soll.
- LHAPDF v6 [19, 10]
- fastjet [20, 21]

Die folgenden Shellbefehle legen ein Quellcodeverzeichnis an und klonen das Git-Repository dort hinein.

```
> mkdir fnlo_src
> cd fnlo_src
> git clone https://gitlab.ekp.kit.edu/qcd-public/fastNLO.git
```

Das *fastNLO*-Repository enthält neben dem Quellcode von *fastNLO* noch einige Skripte, etwa zum Plotten von Tabellendaten oder zum Überprüfen von Tabellen (Verzeichnis `tools`). Das Haupt-Quellcode-Verzeichnis ist `v2.0/toolkit`:

```
> cd v2.0/toolkit
```

fastNLO verwendet *Automake*[2] als Buildsystem. Der Aufruf von `autoreconf` erzeugt ein Configure-Skript, dass dann verwendet wird, um die Einstellungen für die gewünschte Kompilation vorzunehmen. Eine Übersicht über alle Einstellungen gibt der Aufruf von `./configure -help`. In dieser Anleitung beschränken wir uns auf die Aktivierung des Python Interfaces von *fastNLO* durch die Option `-enable-pyext`.

```
> autoreconf -vi
> ./configure --prefix=my_installation_path --enable-pyext
```

Das Configure-Skript überprüft die Abhängigkeiten und erstellt Makefiles, die dann zum Kompilieren und Installieren verwendet werden.

```
> make -j 4
> make install
```

[1] Webpage unter http://www.swig.org/ (letzter Zugriff am 24.04.2018)
[2] Webpage unter https://www.gnu.org/software/automake (letzter Zugriff am 24.04.2018)

Literaturverzeichnis

[1] J. C. Collins, D. E. Soper, and G. Sterman, "FACTORIZATION OF HARD PROCESSES IN QCD", in *Perturbative QCD*, pp. 1–91. WORLD SCIENTIFIC, jul, 1989. doi:10.1142/9789814503266_0001.

[2] D. E.-S. DESY, "HERA-Speicherring". http://www.desy.de/forschung/anlagen__projekte/hera/index_ger.html. [Online; Zugriff 23.04.2018].

[3] F. D. Aaron et al., "Combined measurement and QCD analysis of the inclusive e ± p scattering cross sections at HERA", *Journal of High Energy Physics* **2010** (jan, 2010) doi:10.1007/jhep01(2010)109.

[4] P. Skands, "Introduction to QCD", in *Searching for New Physics at Small and Large Scales*. WORLD SCIENTIFIC, sep, 2013. doi:10.1142/9789814525220_0008.

[5] T. Kluge, K. Rabbertz, and M. Wobisch, "fastNLO: Fast pQCD calculations for PDF fits", in *14th International Workshop on Deep Inelastic Scattering (DIS 2006)*, p. 483. Tsukuba, Japan, April 20-24, 2006. arXiv:hep-ph/0609285. doi:10.1142/9789812706706_0110.

[6] D. Britzger, K. Rabbertz, F. Stober, and M. Wobisch, "New features in version 2 of the fastNLO project", in *Proceedings, XX. International Workshop on Deep-Inelastic Scattering and Related Subjects (DIS 2012)*, p. 217. Bonn, Germany, March 26-30, 2012. arXiv:1208.3641. doi:10.3204/DESY-PROC-2012-02/165.

[7] "fastNLO - fast pQCD calculations for hadron-induced processes". https://fastnlo.hepforge.org/. Online, Zugriff 19.09.2018.

[8] T. Carli et al., "A posteriori inclusion of parton density functions in NLO QCD final-state calculations at hadron colliders: The APPLGRID Project", *Eur. Phys. J. C* **66** (2010) 503, doi:10.1140/epjc/s10052-010-1255-0, arXiv:0911.2985.

[9] "APPLgrid Projekt". https://applgrid.hepforge.org/. Online, Zugriff 06.05.2018.

[10] M. R. Whalley, D. Bourilkov, and R. C. Group, "LHAPDF - the Les Houches Accord PDF Interface". http://lhapdf.hepforge.org/, 2005. [Online; Zugriff 23.04.2018].

[11] "fastNLO, Fast pQCD calculations for hadron-induced processes". https://gitlab.etp.kit.edu/qcd-public/fastNLO.git. [Online, Zugriff 23.04.2018].

[12] L. Garren et al., "Monte Carlo Particle Numbering Scheme", technical report, Particle Data Group, 2017. [Online Zugriff 23.04.2018].

[13] L. Del Debbio, N. P. Hartland, and S. Schumann, "MCgrid: projecting cross section calculations on grids", *Comput. Phys. Commun.* **185** (2014) 2115, doi:10.1016/j.cpc.2014.03.023, arXiv:1312.4460.

[14] "MCgrid Projekt". http://mcgrid.hepforge.org/. Online, Zugriff 06.05.2018.

[15] T. Gleisberg et al., "Event generation with SHERPA 1.1", *JHEP* **02** (2009) 007, doi:10.1088/1126-6708/2009/02/007, arXiv:0811.4622.

[16] "Sherpa Projekt". https://sherpa.hepforge.org/. Online, Zugriff 06.05.2018.

[17] A. Buckley et al., "Rivet user manual", *Computer Physics Communications* **184** (dec, 2013) 2803–2819, doi:10.1016/j.cpc.2013.05.021.

[18] "Rivet". https://rivet.hepforge.org/. Online, Zugriff 06.05.2018.

[19] A. Buckley et al., "LHAPDF6: parton density access in the LHC precision era", *Eur. Phys. J.* **C75** (2015) 132, doi:10.1140/epjc/s10052-015-3318-8, arXiv:1412.7420.

[20] M. Cacciari, G. P. Salam, and G. Soyez, "FastJet User Manual", *Eur. Phys. J. C* **72** (2012) 1896, doi:10.1140/epjc/s10052-012-1896-2, arXiv:1111.6097.

[21] "FastJet Projekt". http://fastjet.fr/. Online, Zugriff 06.05.2018.