Automated Matrix-Element Re-Weighting in Effective Field Theories

Master's Thesis

Christopher Robert Jacobsen
Niels Bohr Institute
November 2015

Advisor
Jørgen Beck Nielsen
Niels Bohr Institute

Abstract

The paper describes a project to develop an automated software tool that enables the use of matrix-element reweighting, using Sherpa to calculate matrix-elements for events, and to generate reweight-coefficients and embed them with the event.

Reweighting was further tested with an *effective field theory* for dimension six-operators. The fitting performance of reweighting an EFT model-dataset to a SM pseudo-dataset for parton-level WZ-production, with phase-space as well as optimal observables was examined.

Table of Contents

1 Introduction

In the search for physics beyond the standard model it is necessary to compare the predictions of a candidate theoretical model to experimental data, in order to ascertain the likelihood of the candidate's veracity. Most models have parameters that influence or determine the behavior of the model, and often the exact values of these parameters are not known in advance. It is common to fit the model to data, to find the set of parameter values that give the best match to experimental data, and in conjunction with error estimates determine confidence limits for the model parameters.

Monte-Carlo event-generators and detector simulators are used to generate model events for a specific set of model parameters, which can then be compared to experimental data for various observables. To test another set of model parameters, new events must typically be generated, and this cycle of changing parameters and re-generating events can be highly time consuming.

For example, to generate one-million events with the Sherpa event generator takes about 10 minutes on a fast desktop computer. Thereafter, the events need to be run through a detector simulator, which will take significantly longer. For ATLAS detector simulations, one million $t\bar{t}$ events will take the fastest simulator (ATLFASTIIF) about 14 days, and the slowest (GEANT4) about four years [1].

Reweighting is a technique that can be used to very quickly recalculate event-distributions for different model parameter values from an existing event dataset. Event generation and detector simulation do not need to be repeated for each model parameter configuration. One can create new event-distributions in milliseconds instead of days or weeks. Clearly reweighting is a

Reweighting works by determining, with the help of an event generator, the functional dependency of the squared matrix-element for each event on the model parameters. By definition, this functional dependency is a quadratic function of the model parameters and can be stored for later retrieval as a set of quadratic coefficients for each event.

As the probability of an event is proportional to its squared matrix element, we can calculate the relative change in frequency for each generated event to a new set of model parameters, by comparing the squared matrix-elements of the new and old model parameters. Reweighting then uses this relative change factor as an event-weight when constructing event-distributions for our model. In this manner, we can construct equivalent event-distributions for any set of model parameters.

The primary goal of this project is to develop a generic software tool that enables the use of matrix-element reweighting. Specifically, to use the *Sherpa* Monte-Carlo event generator [2] to determine the matrix-element dependencies on model parameters, and embed these in generated event files, so that they can be used later for reweighting of observable event-distributions.

The secondary goal of this project is to test the reweighting workflow with *effective field theory* and dimension six-operators, and to test the fitting characteristics of reweighted events to standard-model (SM) pseudo-data. This will involve creating libraries of methods and classes for performing reweighting and fitting.

The tertiary goal of this project is to examine the fitting performance of different observables, specifically testing optimal observables. Additionally, to test the fit characteristics of binned and unbinned fits.

In the remainder of this section we examine reweighting in more detail, and introduce the different third-party software that went into this project.

In section 2, we introduce and describe *SherpaWeight*, the application that was developed to meet our primary goal.

In section 3, we introduce effective field theory (EFT) for dimension-six operators, and describe and test our implementation of the model.

In section 4, we describe and test our reweighting implementation.

In section 5, we introduce and describe optimal observables.

In section 6, we describe our choice of observables, binning, and our attempts to optimize binning.

In section 7, we fit the EFT model to SM pseudo-data for 21 different observables, including optimal observables, using both binned and unbinned fits.

Finally, our conclusions can be found in section 8.

1.1 Reweighting details

We will now look a bit deeper into the details of reweighting. According to Fermi's golden rule, the probability of a scattering process is proportional to its squared matrix-element (aka amplitude):

$$\mathrm{d}\sigma \propto |\langle f|H|i\rangle|^2 = |ME|^2 \tag{1.1}$$

Where $\mathrm{d}\sigma$ is the differential cross-section, essentially the probability of the event; *i*, and *f* are the initial and final states; and *H* is the Hamiltonian of the system. The probability of a scattering process will also include a final-state density term. For proton-proton collisions, the probability of extracting the incoming partons in the initial state from colliding protons must also be included. For convenience, we will often refer to the squared matrix-element as simply the matrix-element or ME.

Theoretical model parameters are generally added linearly to a model's Lagrangian, such that Lagrangian model terms are scaled by the value of the model parameter. Changes to model parameters will only affect the Lagrangian and thus the Hamiltonian. All states and other factors affecting the probability of the event are unchanged.

For model parameters linear in the Lagrangian, the squared matrix-element will by definition be quadratic in terms of these parameters. In the case of a single linear model parameter ϕ the squared matrix-element can be written as:

$$|ME|^2 = a + b\phi + c\phi^2 \tag{1.2}$$

Where A, B, and C are quadratic coefficients, which we will call reweight coefficients. If ϕ is zero in the standard-model (SM), then the A coefficient becomes equivalent to the SM squared matrix-element.

For two or more model parameters cross-terms are added:

$$|ME|^2 = a + \sum_i b_i \phi_i + \sum_i c_i \phi_i^2 + \sum_i \sum_{j>i} d_{i,j} \phi_i \phi_j \tag{1.3}$$

We can write this in matrix-form as:

$$|ME|^2 = \Phi^T \, \boldsymbol{F} \, \Phi = \begin{bmatrix} 1 & \phi_1 & \phi_2 & \cdots \end{bmatrix} \begin{bmatrix} F_{0,0} & F_{0,1} & F_{0,2} & \cdots \\ 0 & F_{1,1} & F_{1,2} & \cdots \\ 0 & 0 & F_{2,2} & \cdots \\ 0 & 0 & 0 & \ddots \end{bmatrix} \begin{bmatrix} 1 \\ \phi_1 \\ \phi_2 \\ \vdots \end{bmatrix} \tag{1.4}$$

Where Φ is a vector of model parameters prefixed with value of 1 in the first index, and ***F*** is an upper triangular matrix of reweight coefficients, corresponding to the a, b, c and d coefficients in (1.3). $F_{0,0}$ is a zero-order coefficient, $F_{0,i}$ are the first-order coefficients, $F_{i,i}$ are the second-order coefficients, and $F_{i,j\ (i \neq j)}$ are the cross-term coefficients.

The number of reweight-coefficients depends upon the number of model parameters, that we wish to be able to reweight events with, and is given by:

$$N_F = \frac{1}{2}\,(N_P + 1)\,(N_P + 2) \tag{1.5}$$

Where *N_F* is the number of reweight-coefficients and *N_P* is the number of reweight-parameters.

1.1.1 Calculating the reweight-coefficients

If we can determine the reweight-coefficient matrix F for each event, then we can recalculate the matrix-element for that event for any model parameter values. Fortunately, Monte-Carlo generators like Sherpa are capable of calculating the matrix-element value for a given event and set of model parameters. We can use linear algebra to solve for the N_F unknown reweight-coefficients by by calculating matrix-elements for each event for N_F independent evaluations of model parameters.

This is achieved by placing the N_F elements of the triangular matrix ***F*** into a vector $\bar{f}$, in row-column order:

$$\bar{f} = [F_{0,0} \quad F_{0,1} \quad F_{0,2} \quad \cdots \quad F_{1,1} \quad F_{1,2} \quad \cdots \quad F_{2,2} \quad F_{2,3} \quad \cdots \quad F_{N_P,N_P}]^T \tag{1.6}$$

Similarly, we can construct an evaluation vector $\bar{a}$ with matching model parameters corresponding to the terms in $\bar{f}$:

$$\bar{a} = [1 \quad \phi_1 \quad \phi_2 \quad \cdots \quad \phi_1^2 \quad \phi_1\phi_2 \quad \cdots \quad \phi_2^2 \quad \phi_2\phi_3 \quad \cdots \quad \phi_{N_P}^2] \tag{1.7}$$

The matrix-element equation for a single event now becomes a simple linear equation:

$$|ME|^2 = \bar{a}\bar{f} \tag{1.8}$$

We can write the N_F independent evaluations and calculated matrix-elements in matrix form, giving us an equation we can solve by linear algebra:

$$\begin{bmatrix} (|ME|^2)_1 \\ (|ME|^2)_2 \\ \vdots \\ (|ME|^2)_{N_F} \end{bmatrix} = \begin{bmatrix} \bar{a}_1 \\ \bar{a}_2 \\ \vdots \\ \bar{a}_{N_F} \end{bmatrix} \begin{bmatrix} F_{0,0} \\ F_{0,1} \\ \vdots \\ F_{N_P,N_P} \end{bmatrix} \quad \equiv \quad \bar{m} = \boldsymbol{A}\bar{f} \tag{1.9}$$

Defining $\bar{m}$ as the vector of calculated ME-values, and $\boldsymbol{A}$ as the matrix of model parameter evaluations, we can acquire $\bar{f}$ our unknown reweight coefficients by:

$$\bar{f} = \boldsymbol{A}^{-1}\bar{m} \tag{1.10}$$

This only requires that ***A*** is constructed in such a manner that it can be inverted, i.e. that the N_F model parameter evaluations are independent.

With the aid of a Monte-Carlo generator, one can thus calculate matrix-elements for different reweight-parameter evaluations and solve for the reweight-coefficients of any generated event. The calculated reweight-coefficients can be stored separately or embedded within the generated event file.

1.1.2 Using reweight-coefficients to reweight event-distributions

With the reweight-coefficients for each event one can calculate the matrix-element for that event using (1.8) for any set of reweight-parameter values. This does not give the new probability or differential cross-section for the event, but one can use it to compare the relative probabilities of the event at two different sets of reweight-parameter values:

$$\frac{P_1}{P_0} = \frac{(|ME|^2)_1}{(|ME|^2)_0} \tag{1.11}$$

If one also knows the the reweight-parameter values used to generate the events, we can calculate the relative probability of an event for any set of reweight-parameter values to its probability when generated.

Normally all generated events in a dataset have equal probability, and are placed in event-distributions with equal weights of one. To reweight an event-distribution, one can simply use the relative probability, compared to the probability when generated, to weight the event. A relative increase in probability for an event will result in it being given a proportionally higher weight in the distribution, and correspondingly for a relative decrease.

Given the reweight-coefficients for each event, and the values of the reweight-parameters for the generated events, one can reweight all event-distributions for the dataset to a new set of reweight model parameter values.

A word of caution: one cannot reweight events if those events are not present in the dataset. One can reweight an event to zero probability, but one cannot give a probability to an event that is not present. So it is essential when reweighting that the dataset not be generated with a set of model parameters that effectively zero terms of the Lagrangian that will later be reweighted.

1.2 Software tools

This project has the primary focus of developing a software tool that automates the calculation of reweight-coefficients using the Monte-Carlo event-generator *Sherpa*. In this section, we briefly introduce *Sherpa* as well as the other major software tools that were used in this project.

1.2.1 Sherpa

Sherpa is a Monte-Carlo event generator for the ***S**imulation of **H**igh-**E**nergy **R**eactions of P**A**rticles*. It is an open-source project developed by the *Sherpa Team*, a global group of physics researchers and students. Information, documentation, and download links for Sherpa are available on Sherpa's web-site [2].

Sherpa can be downloaded from its web-site and built locally on Linux and Mac OS operating systems. For Mac OS, we recommend using the software package tool *Homebrew* [3] and the HEP software repository for *Homebrew* [4], to download and install the latest version of Sherpa easily.

There are various build-options that can be used when building Sherpa. For reference purposes, we used the following build-options for Sherpa for this project:

`--enable-pyext`	Enables Python support. Required for UFO model support.
`--enable-root`	Enables writing and reading of ROOT event files.
`--enable-hepmc2`	Enables writing and reading of HepMC event files.
`--enable-gzip`	Enables support for gzipped event files.
`--enable-lhapdf`	Enables support for PDFs from the LHAPDF library.
`--enable-mpi`	Enables support for OpenMPI multi-processor computing.

Sherpa can be used both as a command line tool and as a C++ library. In both instances, Sherpa uses a text file it calls a run card to setup and configure event generation. The run card file can contain many different parameters, and we refer the reader to the documentation for *Sherpa* for details. Figure 1.1 shows an example run card for *Sherpa*.

The command line for Sherpa is generally fairly simple:

```
Sherpa -f run_card_filename
```

Without the `-f` option Sherpa will look for a run-card called *Run.dat*. There are many other options described in the documentation.

Sherpa also provides a set of C++ classes that can be dynamically linked to by an application. These classes are not documented, however, and it is up to the user to read the header files and source code for Sherpa to determine how they may be used.

Beware: this is not a task for the light-hearted. In this project, much time was used reading and understanding *Sherpa* source code. It is not well commented, and is written in a highly compact and difficult to read form. Further, there is no guarantee of backwards compatibility of the C++ code with future versions of Sherpa.

```
!------ Run parameters (Sherpa 2.2.0 manual 5.1) ------
(run){
    EVENTS        10000
    EVENT_OUTPUT  HepMC_GenEvent[events]
    OUTPUT        2        ! output info and error messages but not events
    PRETTY_PRINT  Off      ! disable color messages
}(run)

!------ Beam parameters (Sherpa 2.2.0 manual 5.2) ------
(beam){
    BEAM_1  2212;  BEAM_ENERGY_1  6500
    BEAM_2  2212;  BEAM_ENERGY_2  6500
  ! BEAM_REMNANTS 0    ! disable beam remnants - only hard process
}(beam)

!------ ISR parameters (Sherpa 2.2.0 manual 5.3) ------
(isr){
    PDF_LIBRARY  CT10Sherpa
    PDF_SET      ct10
}(isr)

!------ Models (Sherpa 2.2.0 manual 5.4) ------
(model){
    MODEL    SM
    CKMORDER 1    ! enable quark-mixing using CABIBBO parameter (default = 0.2272)
}(model)

!------ Matrix elements (Sherpa 2.2.0 manual 5.5) ------
(me){
    ME_SIGNAL_GENERATOR Comix
  ! SCALES VAR{Abs2(p[0]+p[1])}    ! required if SHOWER_GENERATOR is None
  ! ME_QED Off                     ! disable FSR - only hard-process
}(me)

!------ Processes (Sherpa 2.2.0 manual 5.6) ------
(processes){
    Process 93 93 -> 24 23
    End process
}(processes)

!------ Selectors (Sherpa 2.2.0 manual 5.7) ------
(selector){
    PT  24 10.0 E_CMS
    PT  23 10.0 E_CMS
}(selector)

!------ Parton showers (Sherpa 2.2.0 manual 5.10) ------
(shower){
  ! SHOWER_GENERATOR None    ! disable parton showers - only hard-process
}(shower)

!------ Multiple interactions (Sherpa 2.2.0 manual 5.11) ------
(mi){
    MI_HANDLER None          ! disable multiple interactions
}(mi)

!------ Hadronization (Sherpa 2.2.0 manual 5.12) ------
(fragmentation){
  ! FRAGMENTATION Off        ! disable hadronization - only hard-process
}(fragmentation)
```

Figure 1.1: Example Sherpa run card file.

This file instructs Sherpa to generate 10000 events from proton-proton collisions at $\sqrt{s}$ = 13 TeV, with the process $q\bar{q} \rightarrow W^{+}Z$. Run parameters are grouped into different sections, and are either listed on a line by themselves, or may be separated by semi-colons on the same line. Section-names, parameters and parameter values are case-sensitive. Comments begin with either an exclamation mark (!) or a hash (#). Multiple whitespace characters (spaces and tabs) are treated as one.

As this project took place over the span of a year, work began with Sherpa version 2.1.1 and continued with Sherpa version 2.2.0, upon its release in July 2015. These two versions are significantly different in many aspects, and much of the programming work done with 2.1.1 had to re-done with 2.2.0. The biggest change that affected this project was the change of supported models.

Sherpa 2.1.1 supports among others, the standard model, an anomalous gauge coupling model, and user defined models via a specially defined FeynRules export format. Sherpa 2.2.0 only supports the standard model and user-defined FeynRules UFO models, a completely different FeynRules model format than used in the previous version.

1.2.2 FeynRules and Mathematica

FeynRules is a software package that allows the calculation of Feynman rules in momentum space for *any* QFT physics model. It is used together with *Mathematica*, a commercially available multi-purpose mathematics software system. *FeynRules* is an open-source project maintained by a global group of physics researchers and students. Information, documentation, and download links for *FeynRules* are available on *FeynRules* web-site [5]. Some additional instructions on using *FeynRules*, which is well worth reading, is given in [6] and [7].

FeynRules allows the user to define a QFT physics model using FeynRules methods and definitions. The model can be tested for Hermiticity along with many other tests. Models can be exported in several different formats for use by event-generators.

We used *FeynRules* to define a working Effective Field Theory model for three dimension-six operators, and exported this model to *Sherpa* using two different model formats: a special FeynRules-Sherpa model format, and the FeynRules UFO model format [8]. The first is a custom format only useable by Sherpa version 2.1.1 and earlier. The second is an open format supported by many event-generators, and supported by Sherpa version 2.2.0.

We recommend using the UFO model format together with Sherpa 2.2.0, as we encountered many problems using the special FeynRules-Sherpa model with Sherpa 2.1.1. Specifically, FeynRules had difficulty compiling dimension-six operators into the specialized Sherpa export format.

We discuss using FeynRules to implement our EFT model further in section 3.1.

For this project we used FeynRules version 2.0.32 through 2.3.7, and recommend using version 2.3.7 which was the final version we used and it worked without issue. We did encounter some issues with earlier versions.

1.2.3 HepMC

HepMC is a C++ library for reading and writing object-orientated event records for use by high energy physics Monte-Carlo event-generators. HepMC is an open-source project and the HepMC file format is supported by many tools and event-generators. The library and its documentation can be downloaded from its web-site [9].

We used HepMC for storing this project’s generated events, as it is a open format, is supported by Sherpa, compresses well using gzip, and supports the addition of reweight-coefficients to the event records.

We used version 2.06.08 of HepMC.

1.2.4 ROOT

ROOT is a large library of C++ classes and methods that provides data analysis, statistics, mathematics, and physics functionality. ROOT is developed and maintained by CERN and the source code is freely available for download. Both the documentation and installation links are available on ROOT's web-site [10].

For Mac OS users, we again recommend using Homebrew to install ROOT quickly and easily.

We used ROOT for event and data storage, constructing and storing histograms, performing binned and unbinned fits, and drawing figures. We also used ROOT for four-momentum vector calculations on event data.

ROOT is a large and comprehensive library, that is easy to get started with, but hard to master. The documentation is pretty good for most downloadable software projects, using automated tools to create the documentation from the source code. However, the documentation can be quite thin is many areas, and the user is left reading header files and source code to understand the underlying functionality. In this project, much time was used in understanding and reading the source code for histograms (*TH1D*), profiles (*TProfile*), and all the many layers of fitting and function minimization classes and helper methods.

We used version 5.34.32, the last available version of ROOT when the project started. Since then, ROOT version 6 has been released. As version 6 had no new functionality that we required, and we were familiar with version 5, we continued using the version we began the project with.

More details are given on how we implanted key functionality in this project using ROOT in various sections of this report.

2 SherpaWeight

SherpaWeight is the name of the software tool developed in this project for calculating reweight-coefficients for events, using Sherpa to calculate the matrix-elements, and to embed the coefficients with the events in an output event file.

It is the intention from the outset of this project that this tool be made as generic as possible and to be shared upon completion with the high energy physics community for use in the search for physics beyond the standard model. SherpaWeight was also required to be compatible with future-extensions, such as new file or model support. Thus the code of *SherpaWeight* must be written to a high standard.

SherpaWeight is a command-line application, designed to implement the following functionality:

1. Read events from a number of supported input file formats, that is easily extendable. Currently three event file formats are supported: HepMC, gzipped HepMC, and Sherpa-ROOT.

   Note that the events do not need to be generated by Sherpa.
2. Initialize the *Sherpa* framework using a user-provided run-card file that details the model, processes, and other criteria that Sherpa requires to generate matrix-elements for events.
3. From the same run-card used to initialize the *Sherpa* framework, read from a *SherpaWeight* specified section, the names of the model-parameters for which reweight-coefficients should be calculated for, as well as their evaluation center- and delta-values.
4. Calculate the number of reweight-coefficients N_F, and construct N_F independent sets of parameter evaluation values, using the evaluation range for each parameter. Using the N_F sets of evaluation values, construct the $N_F \times N_F$ $\boldsymbol{A}^{-1}$ matrix from (1.10).
5. For each of the N_F sets of parameter evaluation values, set the corresponding model parameters in the *Sherpa* framework, and then iterate through all events acquiring the matrix-element from the Sherpa framework and adding it to a vector $\bar{m}$ for each event.
6. iterate through all events the vector $\bar{m}$ for that event along with the $\boldsymbol{A}^{-1}$ matrix and (1.10) to acquire the vector of reweight-coefficients $\bar{f}$ for each event.
7. Generate a new event file of the same format as the input event file, copying all the event data and embedding the calculated reweight-coefficients for each event in its event record.

2.1 Developing SherpaWeight

SherpaWeight was developed using C++11 and a selection of classes from the Sherpa, HepMC and ROOT libraries. It was developed on Mac OS X, using the clang compiler and XCode for editing, debugging and build configuration. It is easily portable to Linux, and can be built with a simple makefile.

In developing *SherpaWeight* it was necessary to gain an intimate knowledge of how some of the key classes in Sherpa works, particularly how its framework is initialized and prepared for generating events, as well as how it calculates matrix-elements. As mentioned earlier, the source code for Sherpa is not documented, and is written in a terse manner making it difficult to read. Much time was spent digging through the layers of Sherpa to understand why something did not work as expected. Further, as we began using Sherpa fully, we found that it contained many bugs or undocumented setup requirements, which if you were lucky would result in an exception being thrown, but more often than not would result in a direct crash or hang.

The first issue to resolve was how to get the matrix-element for an event from the Sherpa framework. Fortunately, Sherpa includes with its source code an example C++ class, called *MEProcess*, with the required functionality, and an example application showing how to use it. Unfortunately, the *MEProcess* class did not work properly for processes that include a decay specification. Eventually, we figured out how to modify the *MEProcess* to support all possible processes, and created our own class based upon *MEProcess* to do the matrix-element calculations.

The second issue we had to overcome was the issue of setting the reweight model-parameters for each evaluation in step 5 above. We discovered to our dismay, that the Sherpa framework only allows the setting of model-parameters during the frameworks initialization, and that the framework can only initialized once per execution of an application.

This resulted in us dividing the functionality of *SherpaWeight* into two applications: *SherpaWeight* and *SherpaME*. *SherpaME* is a standalone application that simply calculates the matrix-elements for the events in a given input event file and stores the results in a ROOT data file. *SherpaWeight* sets up the running environment for *SherpaME*, including the values of the reweight model-parameters, and calls *SherpaME* to calculate the matrix-elements for that specific configuration. *SherpaWeight* calls *SherpaME* N_F times, once for each set of parameter evaluation values, reading the matrix-elements from the ROOT data file generated by *SherpaME*.

2.1.1 Calculation of independent evaluations

For the calculation of the independent sets of parameter evaluation values, we were inspired by the code used by *AfterBurner*, an application developed at NBI by Jørgen Beck Hansen and some of his students.

It constructs a $N_F \times N_P$ matrix, where each row is a set of evaluation values for the N_P reweight model-parameters. It fills the first row with all zeroes, then next N_P rows with a diagonal $N_P \times N_P$ matrix of +1, and the following next N_P rows with a diagonal diagonal $N_P \times N_P$ matrix of −1, and the remaining rows with the vector $[1 \quad -1 \quad 0 \quad \cdots \quad 0]$, rotated by 1 in each subsequent row.

For N_P = 1 we get 3 independent rows:

$$\begin{bmatrix} 0 \\ +1 \\ -1 \end{bmatrix} \tag{2.1}$$

For N_P = 2 we get 6 independent rows:

$$\begin{bmatrix} 0 & 0 \\ +1 & 0 \\ 0 & +1 \\ -1 & 0 \\ 0 & -1 \\ +1 & -1 \\ -1 & +1 \end{bmatrix} \tag{2.2}$$

For N_P = 3 we get 10 independent rows:

$$\begin{bmatrix} 0 & 0 & 0 \\ +1 & 0 & 0 \\ 0 & +1 & 0 \\ 0 & 0 & +1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ +1 & -1 & 0 \\ 0 & +1 & -1 \\ -1 & 0 & +1 \end{bmatrix} \tag{2.3}$$

And so on.

To support different parameter ranges, SherpaWeight enables uses to define a center-value and delta-value for each reweight-parameter. SherpaWeight multiplies each column of the simple ±1 matrixes derived above by the delta-value for the corresponding parameter, and adds the center-value. In this way the parameter values are transformed in each column from ±1 to $c \pm \delta$ for the parameter.

The resulting parameter evaluation matrix is power and cross-term expanded into the $\boldsymbol{A}$ evaluation matrix, which is inverted to get the $\boldsymbol{A}^{-1}$ matrix.

2.2 Using SherpaME

The command line for SherpaME is:

```
SherpaME inputFile outputFile <Sherpa command line options>
```

The *inputFile* argument is the file path to an event file, in either HepMC, gzipped HepMC, or Sherpa-ROOT format. The *outputFile* argument is the file path to a ROOT data file in which will be written a table of matrix-element values and event ids for each event. If the output file already exists, it will be overwritten.

Any valid *Sherpa* command line option can be appended to the command line following the first two arguments, and will be passed unmodified to the Sherpa framework.

2.3 Using SherpaWeight

The command line for *SherpaWeight* is:

```
SherpaWeight inputFile outputFile <Sherpa command line options>
```

The *inputFile* argument is the file path to an event file, in either HepMC, gzipped HepMC, or Sherpa-ROOT format. The *outputFile* argument is the file path to store a copy of the input events with the reweight-coefficients embedded in each event record. If the output file already exists, it will be overwritten. The output file will be created in the same format as the input file. The output file path may not be the same as the input file path.

Any valid *Sherpa* command line option can be appended to the command line following the first two arguments, and will be passed unmodified to the Sherpa framework. Typically, this might be used to specify the run-card for *SherpaWeight* to use, otherwise it will look for a Run.dat file in the same folder where *SherpaWeight* is run from.

In addition, SherpaWeight requires a `(SherpaWeight)` section to be present in the run card file that Sherpa will use for calculating matrix elements, which is Run.dat by default, unless specified using a Sherpa command line option. The `(SherpaWeight)` section follows the same rules for all sections in a Sherpa run-card file. Everything following a comment is ignored and blank lines are ignored. Each entry in the `(SherpaWeight)` section corresponds to a desired reweight parameter, and has the the following format:

```
<parameter-name> <center-value> <delta-value>
```

`<parameter-name>` is the name of a model-parameter to be used for reweighting. `<center-value>` and `<delta-value>` are floating-point numbers that specify the center-and delta-value used for evaluating this parameter to determine the reweight-coefficients for each event.

If the model is a UFO model, where parameters are stored in blocks in a `(ufo)` section, the parameter-name is the block-name for the parameter, followed by the parameter-number in square brackets. For example, `eftcoef[0]` is the parameter-name for the zeroth parameter in the `eftcoef` block. Alternatively, a user-friendly name can be placed in a comment following the parameter values in a section, and in this case, the name in the comment can be used directly. Note that the first word in the comment will be treated as the name.

SherpaWeight creates a temporary folder called *SherpaWeight.tmp* in the running folder, in which it places log files and other temporary files, such as the output ROOT data file from *SherpaME*. This folder is not removed when *SherpaWeight* exits, so its contents can be used to help debug any issues that may have occurred. Users are free to delete this folder manually.

3 Effective Field Theory

To test re-weighting we have chosen to apply it to effective field theory (EFT), specifically three dimension-six operators. Effective field theory extends the standard model while respecting gauge symmetries. It can be used to search for physics beyond the standard model by searching for new interactions of known particles. The EFT framework is considered cleaner and simpler to that of anomalous couplings, as it does not use an arbitrarily chosen form factor, but takes a model-independent approach to the physics of non-standard interactions. In this section, we will primarily follow the theoretical arguments laid forth by [11] and [12].

In the standard model (SM), all operators are of dimension four, except the quadratic term in the Higgs potential. The principle idea of EFT is to simply add operators of higher dimension with unknown coefficients that define the strength of the operators and its corresponding interactions. By dimensional analysis, the operator coefficients will be of inverse powers of mass, and a mass scale Λ can be defined to help characterize the coefficients. The following is the EFT extension to the SM for dimension-six and dimension-eight operators:

$$\mathcal{L} = \mathcal{L}_{SM} + \sum_i \frac{c_{6,i}}{\Lambda^2} \mathcal{O}_{6,i} + \sum_j \frac{c_{8,j}}{\Lambda^4} \mathcal{O}_{8,j} + \cdots \tag{3.1}$$

Where $c_{6,i}$ and $c_{8,j}$ are dimensionless coefficients to i dimension-six operators $\mathcal{O}_{6,i}$ and j dimension-eight operators respectively.

EFT assumes that Λ is large compared to experimental energies, and the higher dimensional operators are thus suppressed at the lower experimental energies. An effective field theory is essentially a low-energy approximation to new physics, below the new physics scale Λ.

Operators of lower dimensionality will clearly dominate the EFT extension, and we will concentrate on dimension-six operators only. In addition, we feel that the scale Λ, essentially just introduces an unnecessary complication, and can be absorbed into the coefficients themselves. This gives us:

$$\mathcal{L} = \mathcal{L}_{SM} + \sum_i c_i \, \mathcal{O}_i \tag{3.2}$$

Where c_i are the coefficients to the dimension-six operators $\mathcal{O}_i$. c_i have mass-dimension −2 (M^{-2}).

There are several dimension-six operators, that could affect a given physical process. For example, there are 59 independent B- and L-conserving dimension-six operators for one generation of quarks and leptons. We will focus on interactions with electroweak vector bosons, particularly self-interactions, as interactions with fermions are constrained by other processes. Assuming C and P conservation, there are only three independent dimension-six operators for EW self-interactions:

$$\mathcal{O}_{WWW} = Tr\left[\widehat{W}_{\mu\nu}\widehat{W}^{\nu\rho}\widehat{W}_{\rho}^{\ \mu}\right] \tag{3.3}$$

$$\mathcal{O}_W = \left(D_\mu\Phi\right)^\dagger \widehat{W}^{\mu\nu}(D_\nu\Phi) \tag{3.4}$$

$$\mathcal{O}_B = \left(D_\mu\Phi\right)^\dagger \widehat{B}^{\mu\nu}(D_\nu\Phi) \tag{3.5}$$

Where Φ is the Higgs doublet field and

$$D_\mu = \partial_\mu + \frac{i}{2} g_W \sigma^I W_\mu^I + \frac{i}{2} g' B_\mu \tag{3.6}$$

$$\widehat{W}_{\mu\nu} = \frac{i}{2} g_W \sigma^I \left(\partial_\mu W_\nu^I - \partial_\nu W_\mu^I + g_W \epsilon_{IJK} W_\mu^J W_\nu^K\right) = \frac{i}{2} g_W \sigma^I W_{\mu\nu}^I \tag{3.7}$$

$$\widehat{B}_{\mu\nu} = \frac{i}{2} g' \left(\partial_\mu B_\nu - \partial_\nu B_\mu\right) = \frac{i}{2} g' B_{\mu\nu} \tag{3.8}$$

Where D_μ is the covariant derivative. $W_{\mu\nu}^I$ and $B_{\mu\nu}$ are respectively the field-strengths tensors of the W_μ^I and B_μ gauge-bosons. g_W and g' are respectively the weak and U(1) Y coupling constants at the Z pole. σ^I are the Pauli-sigma matrixes, and ϵ_{IJK} is the Levi-Civita symbol.

$\widehat{W}_{\mu\nu}$ and $\widehat{B}_{\mu\nu}$ are defined by requiring:

$$[D_\mu, D_\nu] = \widehat{W}_{\mu\nu} + \widehat{B}_{\mu\nu} \tag{3.9}$$

Note that $\widehat{W}_{\mu\nu}$ is a 2×2 matrix, thus the use of trace in (3.3).

The coefficients corresponding to the three operators $\mathcal{O}_{WWW}$, $\mathcal{O}_W$ and $\mathcal{O}_B$ are defined as c_{WWW}, c_W, and c_B respectively.

3.1 Implementing a UFO model for EFT in FeynRules

To use the EFT model in Sherpa, we need to construct a Universal FeynRules Output (UFO) model [8]. Sherpa 2.2.0 can use a UFO model to generate a Sherpa model dynamic library for event generation. Note that before version 2.2.0 of Sherpa, this was not possible. We used FeynRules 2.3.7 to generate a UFO model for EFT, and detail our implementation in this section.

We copied the SM FeynRules file (SM.fr) to a new file (SM+EFT.fr) so that we could add our dimension-six operators to the existing SM definition. We defined each new coefficient and operator using the FeynRules extension to Mathematica. Then we checked that our implementation was Hermitian. Finally, we exported our model to the UFO format.

The following is the code used to setup the three external coefficients for our EFT model:

```
 (* ************************** *)
 (* *****   Parameters   ***** *)
 (* ************************** *)
M$Parameters = {

   (* SM paramters are here *)

   (* Effective Field Theory External Parameters *)
   (* Coefficients for dimension 6 electroweak-boson self-interactions" *)
   ocWWW == {
     ParameterType -> External,
     BlockName     -> EFTCOEF,
     OrderBlock    -> 1,
     Value         -> 0,
     TeX           -> Subscript[oc, WWW],
     Description   -> "Subscript[O,WWW] coefficient"
   },
   ocW == {
     ParameterType -> External,
     BlockName     -> EFTCOEF,
     OrderBlock    -> 2,
     Value         -> 0,
     TeX           -> Subscript[oc, W],
     Description   -> "Subscript[O,W] coefficient"
   },
   ocB == {
     ParameterType -> External,
     BlockName     -> EFTCOEF,
     OrderBlock    -> 3,
     Value         -> 0,
     TeX           -> Subscript[oc, B],
     Description   -> "Subscript[O,B] coefficient"
   },
 };
```

A parameter block called EFTCOEF was defined, with three parameters: `ocWWW`, `ocW`, and `ocB`, corresponding to our three EFT model coefficients. The o-prefix was used in-order to lengthen the parameter name and avoid any possible conflicts with pre-existing names.

The following is the code used for our three dimension-six operators:

```
(* Effective Field Theory - Dimension 6 Operators *)

LOD6 := Block[{mu,nu,rho,ii,jj,kk,feynmangaugerules,Bhat,What},
  feynmangaugerules = If[Not[FeynmanGauge], {G0|GP|GPbar ->0}, {}];

  What[mu_,nu_,ii_,jj_] :=
    Module[{aa}, -I/2 gw PauliSigma[aa,ii,jj] FS[Wi,mu,nu,aa]];

  Bhat[mu_,nu_] := -I/2 g1 FS[B,mu,nu];

  ExpandIndices[ocWWW What[mu,nu,ii,jj] What[nu,rho,jj,kk]
    What[rho,mu,kk,ii], FlavorExpand->True] +

  ExpandIndices[ocW DC[Phibar[ii],mu] What[mu,nu,ii,jj]
    DC[Phi[jj],nu], FlavorExpand->True] +

  ExpandIndices[ocB DC[Phibar[ii],mu] Bhat[mu,nu] DC[Phi[ii],nu],
     FlavorExpand->True]

  /. feynmangaugerules ];

LSM_EFT:=LSM + LOD6;
```

A block called LOD6 was defined to contain the dimension-six operators. An optional `feynmangaugerules` substitution rule is created to support both unitary and Feynman gauge, removing ghost particles when unitary gauge is selected. Note that we only used unitary gauge for this project.

The `What` and `Bhat` methods correspond to the $\widehat{W}_{\mu\upsilon}$ and $\widehat{B}_{\mu\upsilon}$ equations given in (3.7) and (3.8). `g1` and `gw` are respectively the U(1) Y coupling constant and weak coupling constant at the Z pole. The `PauliSigma` method returns the specified index (`ii,jj`) of the specified (`aa`) Pauli-sigma matrix. The `FS` FeynRules method constructs the field strength tensors.

Note that there is a sign difference between the implementation of the `What` and `Bhat` methods and the $\widehat{W}_{\mu\upsilon}$ and $\widehat{B}_{\mu\upsilon}$ equations. This is due to the sign difference in the covariant derivative in (3.6) and the definition of the FeynRules covariant derivative method `DC`:

$$DC = \partial_\mu - \frac{i}{2} g_W \sigma^I W_\mu^I - \frac{i}{2} g' B_\mu \tag{3.10}$$

This results in a change of sign in the derivation of $\widehat{W}_{\mu\upsilon}$ and $\widehat{B}_{\mu\upsilon}$ following (3.9).

The FeynRules method `ExpandIndexes` is used to contract the indexes of the three dimension-six operators. The trace given in (3.3) is acquired by contracting implicit i, j indexes of the three 2×2 $\widehat{W}_{\mu\upsilon}$ terms in a circular fashion.

Finally, our dimension-six operator Lagrangian (LOD6) is appended to the existing SM Lagrangian (LSM) to create our SM+EFT Lagrangian (LSM_EFT).

3.1.1 Generating the UFO model.

To generate the UFO model from FeynRules, the following commands were run from Mathematica:

```
$FeynRulesPath = SetDirectory["Path/feynrules-2.3.7"];
<< FeynRules`
SetDirectory[NotebookDirectory[]]
LoadModel["SM+EFT.fr"]
FeynmanGauge = False;

LoadRestriction["Sherpa_Massless.rst"]

FeynRules[LOD6]
CheckHermiticity[LOD6]

WriteUFO[LGauge, LHiggs, LFermions, LYukawa, LOD6]
```

The first few lines load the FeynRules package and our SM+EFT model into Mathematica, ensuring that the gauge is unitary. The `LoadRestriction` command loads an additional model setup file, discussed in section 3.1.2, which enforces some initial conditions for compatibility with Sherpa. The FeynRules and `CheckHermiticity` commands validate that our Lagrangian for dimension-six operators can be understood by FeynRules and that it is Hermitian. Finally, the `WriteUFO` command is used to create a UFO model comprising of the four SM Lagrangians and our dimension-six Lagrangian.

Note: One could have called `WriteUFO` with only LSM_EFT as a parameter, but this can cause the command to fail as it runs out of memory. By handling each Lagrangian separately, `WriteUFO` accomplishes the same result but processes each Lagrangian separately.

The resulting UFO model is created in a "SM+EFT_UFO" sub-folder and contains several python files listed in Table 3.1.

Table 3.1: UFO model files.

Filename	**Contents**
particles.py	Particles defined using the Particle class.
parameters.py	Parameters, like masses, coupling constants, etc., defined using the Parameter class.
vertices.py	Interactions defined using the Vertex class.
couplings.py	Coupling terms for the defined vertices.
CT_couplings.py	Counter-coupling-terms for the defined vertices.
lorentz.py	Lorentz structures for the model.
coupling_orders.py	Coupling order for e.g. QCD and QED.
decays.py	Decay widths for various particles.
propagators.py	Propagator definitions.
__init__.py function_library.py	Model independent files containing various UFO model methods.

object_library.py
write_param_card.py

The most interesting of these files are *parameters.py*, *vertices.py* and *couplings.py*. *parameters.py* contains the definitions of our 3 EFT model parameters, as well as other external SM parameters. *vertices.py* contains the definitions of all the SM and EFT model vertices. *couplings.py* contains the coupling constants used by *vertices.py*.

A quick overview of the differences in file contents between SM and SM+EFT is given in Table 3.2. As we have dimension-six operators in our EFT-model, some of the vertices are for five or six-particle interactions. The additional vertices added by our EFT model are listed in Table 3.3.

Table 3.2: Major differences between SM and SM+EFT UFO models

Filename	SM	SM+EFT
parameters.py	12 external parameters	15 external parameters
vertices.py	66 vertices 3 legs: 58 4 legs: 8	87 vertices 3 legs: 58+1=59 4 legs: 8+3=11 5 legs: 11 6 legs: 6
couplings.py	43 couplings	83 couplings

Table 3.3: List of additional vertices added by the EFT model

Vertex Legs	Particles
3	(H, Z, γ)
4	$(H, W^+, W^-, Z/\gamma)$, (H, H, Z, γ)
5	$(H, H, W^+, W^-, Z/\gamma)$, $(H, W^+, W^-, Z, Z/\gamma)$ $(W^+, W^-, Z/\gamma, Z/\gamma, Z/\gamma)$, $(W^+, W^+, W^-, W^-, Z/\gamma)$
6	(H, W^+, W^+, W^-, W^-), $(H, H, W^+, W^-, Z, Z/\gamma)$ $(W^+, W^+, W^-, W^-, Z/\gamma, Z/\gamma)$, $(W^+, W^+, W^-, W^-, H, H)$

Most of the additional vertices added by our EFT model include Higgs particles, which are rather rare. So we do not expect much affects from these at experimental energies. Further, in our evaluation of this model, we will concentrate on known processes, such as WZ-production, where these additional vertices will play no role.

The most significant change is the modifications to the couplings of the existing SM vertices, caused by the introduction of our dimension-six operators, where the three new EFT parameters now play a role.

3.1.2 Generating a Sherpa-compatible UFO model

Built-in Sherpa models treat many of the particles as massless by default in the hard-process calculation. The so-called *jet* particle container (pdg code 93), is often used to represent incoming partons in Sherpa process definitions, and automatically contains the pdg codes of all massless quarks and the gluon.

With UFO models, particles are not massless by default, and it is up to the model to set the mass to zero, for it to be treated as massless and to be included automatically in the *jet* particle container.

To accomplish this, a set of restraints or initial model values can be set after loaded the FeynRules model file, using the `LoadRestrictions` command. We defined such a set of initial conditions in the file *Sherpa_Massless.rst*:

```
(*****************************************************************)
(*     Restriction file for SM.fr                               *)
(*     Only t-quarks are massive                                *)
(*****************************************************************)

M$Restrictions = {
    Me  -> 0,  yme   -> 0,  ye   -> 0,  (* massless electron *)
    MMU -> 0,  ymm   -> 0,  ym   -> 0,  (* massless muon     *)
    MTA -> 0,  ymtau -> 0,  ytau -> 0,  (* massless tau      *)

    MU  -> 0,  ymup  -> 0,  yup -> 0,   (* massless u-quark  *)
    MD  -> 0,  ymdo  -> 0,  ydo -> 0,   (* massless d-quark  *)
    MS  -> 0,  yms   -> 0,  ys  -> 0,   (* massless s-quark  *)
    MC  -> 0,  ymc   -> 0,  yc  -> 0,   (* massless c-quark  *)
    MB  -> 0,  ymb   -> 0,  yb  -> 0    (* massless b-quark  *)
}
```

This completes our UFO model creation. In section 3.3, we will continue to refine our UFO model setup, by adjusting its external parameters to match Sherpa defaults.

3.2 Using the EFT UFO model with Sherpa

Sherpa 2.2.0 supports the importing of UFO models, by using the script *Sherpa-generate-model* and passing the path to the UFO model files generated by FeynRules. This script converts the python files into C++ files, compiles them, and links them into a Sherpa model dynamic library, which it installs in the current Sherpa installation folder. There is, however, three catches.

The first catch, is that various other software packages must be pre-installed in order for the script to run. These include python, scons, cmake, and automake, as well as these packages dependencies. Further, Sherpa must be compiled with the *--enable-pyext* option.

The second catch, is that the folder name is used to create the model name as well as a matching C++ class name. The folder name *SM+EFT_UFO* is not compatible with these scripts, as it is not a valid C++ class name. Thus, the folder name was changed to *SM_EFT*, dropping the UFO suffix.

The third catch, is that there is a bug in one of the scripts included with Sherpa that prevents linking on Mac OS X with the clang compiler. We have reported this bug to the Sherpa team, and it will likely be fixed in the next version. However, until then, this is easily fixed by modifying the Sherpa file:

```
<Sherpa installation>/lib/python2.7/site-
packages/ufo_interface/sconstruct_template
```

Find the line:

```
vars.Add('CXX','The C++ Compiler','g++ -Wl,--no-as-needed')
```

and delete the "`-Wl,--no-as-needed`" part, resulting in:

```
vars.Add('CXX','The C++ Compiler','g++')
```

Sherpa-generate-model not only generates the Sherpa model dynamic library, it also generates an example Sherpa run-card, showing how to select the EFT model, and to define the external parameters for this model. Below are the most relevant elements from the generated example run-card for our model:

```
(run){
  # model setup
  MODEL SM_EFT

  # me generator setup
  ME_SIGNAL_GENERATOR Comix
}(run)

(ufo){
block yukawa
    6 172
block ckmblock
    1 0.227736
block mass
    23 91.1876
    6  172
    25 125
block sminputs
    1 127.9
    2 1.16637e-05
    3 0.1184
block eftcoef
    1 0
    2 0
    3 0

decay 23 2.4952
decay 24 2.085
decay 6  1.50833649
decay 25 0.00407
}(ufo)
```

In the `(run)` section, the `MODEL` parameter is set to the name of the model built by the script, which in our case is `SM_EFT`. The `ME_SIGNAL_GENERATOR` parameter is set to `Comix`. *Comix* is the only built-in only Sherpa signal generator that can be used with UFO models. Strictly speaking it is not necessary to specify *Comix*, as Sherpa will use it automatically for UFO models.

The `(ufo)` section is the most important section when working with UFO models. It defines all the UFO model external parameters. For our model, the most interesting are the three EFT model parameters, specified in the `eftcoef` block. However, the other parameters are set to their UFO model defaults, which are different than the defaults used for Sherpa's built-in models.

To be compatible with the built-in Sherpa models, we need to add a few entries and modify the `(ufo)` section parameters, as follows:

```
(run){
  # model setup
  MODEL SM_EFT

  # me generator setup
  ME_SIGNAL_GENERATOR Comix;

  # set stable to 0 for those particles
  # with a non-zero decay-width
  STABLE[6]  0
  STABLE[23] 0
  STABLE[24] 0
  STABLE[25] 0

  # disable ghost particles
  # these are not used by the model anyway (unitary gauge)
  ACTIVE[82]      0
  ACTIVE[250]     0
  ACTIVE[251]     0
  ACTIVE[9000001] 0
  ACTIVE[9000002] 0
  ACTIVE[9000003] 0
  ACTIVE[9000004] 0
}(run)

(ufo){
block yukawa
    6 173.21                  # ymt
block ckmblock
    1 0.2272                  # cabi
block mass
    23 91.1876                # MZ
    6  173.21                 # MT
    25 125                    # MH
block sminputs
    1 128.8022421894320       # aEWM1
    2 1.197448274494500E-05   # Gf
    3 0.118                   # aS
block eftcoef
    1 0                       # ocWWW
    2 0                       # ocW
    3 0                       # ocB

decay 23 2.4952
decay 24 2.085
decay 6  2.0
decay 25 0.00407
}(ufo)
```

The `STABLE` and `ACTIVE` parameters in the `(run)` section, are strictly speaking not necessary. However, they do ensure that the default setup for our UFO model is identical to that of the Sherpa built-in SM.

The parameters in the `(ufo)` section have been updated to match the defaults used by Sherpa 2.2.0. Comments have been added to help the reader identify the parameters in each block.

Note that one parameter is not the same as the Sherpa default, and that is the Cabbibo angle (`cabi`), which defines the CKM matrix. In Sherpa, the default CKM matrix is a unit matrix, disabling quark-mixing. However, in this project, when working with built-in models, we always enable quark-mixing and set the Sherpa parameter CKMORDER = 1, which effectively sets the Cabbibo angle to 0.2272. Thus for compatibility, we have also configured all EFT models to use the same value.

Note that for our SM+EFT UFO model, the CKM matrix is given by:

$$\begin{bmatrix} d' \\ s' \end{bmatrix} = \begin{bmatrix} \cos\theta_c & \sin\theta_c \\ -\sin\theta_c & \cos\theta_c \end{bmatrix} \begin{bmatrix} d \\ s \end{bmatrix} = \begin{bmatrix} 0.9743 & 0.2253 \\ -0.2253 & 0.9743 \end{bmatrix} \begin{bmatrix} d \\ s \end{bmatrix} \tag{3.11}$$

Where θ_c is the Cabbibo angle of 0.2272.
However, in Sherpa, the CKM matrix for built-in models is instead approximated by the leading order terms of the Taylor-series expansion of cosine and sine:

$$\begin{bmatrix} d' \\ s' \end{bmatrix} = \begin{bmatrix} 1-\frac{1}{2}\theta_c^2 & \theta_c \\ -\theta_c & 1-\frac{1}{2}\theta_c^2 \end{bmatrix} \begin{bmatrix} d \\ s \end{bmatrix} = \begin{bmatrix} 0.9484 & 0.2272 \\ -0.2272 & 0.9484 \end{bmatrix} \begin{bmatrix} d \\ s \end{bmatrix} \tag{3.12}$$

The difference is rather small, but it could lead to small differences between the events generated for the built-in SM and those for the SM+EFT UFO model.

3.3 Validating EFT model using WZ production

In order to test that we have built the EFT model correctly, we need to compare the output generated by this model with an equivalent model. Fortunately, Sherpa has a built-in model that is compatible with EFT: anonymous gauge couplings (AGC). The coefficients of EFT are convertible to equivalent parameters in AGC. We can generate events with both models using equivalent parameters, and compare the cross-sections as well as the distributions of several different observables.

Without getting into too much details on the AGC model, the built-in Sherpa model SM+AGC has five C- and P-conserving parameters that can be converted to our three EFT parameters and visa-versa: $g_1^Z, \lambda_Z, \lambda_\gamma, \kappa_\gamma, \kappa_Z$. Other AGC parameters are available, but they violate C- and/or P-conservation. The relationships between the two sets of model parameters is given by:

$$c_{WWW} = \frac{2}{3g_W^2 m_W^2}\,\lambda_{\gamma/Z} \tag{3.13}$$

$$c_W = \frac{2}{m_Z^2}\,\Delta g_1^Z \tag{3.14}$$

$$c_B = \frac{2}{m_Z^2}\,(\Delta\kappa_\gamma - \Delta\kappa_Z) \tag{3.15}$$

Where $\Delta g_1^Z = g_1^Z - 1$ and $\Delta\kappa_{\gamma/Z} = \kappa_{\gamma/Z} - 1$ and $\Delta g_1^Z = \Delta\kappa_Z + \tan^2\theta_W\ \Delta\kappa_\gamma$. Note that $\lambda_\gamma = \lambda_Z$. AGC parameters for the SM are $\lambda_{\gamma/Z} = \Delta g_1^Z = \Delta\kappa_{\gamma/Z} = 0$.

These relationships are only valid below the scale of new physics Λ, and only if we restrict ourselves exclusively to dimension-six operators. They also require that we treat the anomalous couplings as constants independent of energy.

UFO models, such as our EFT model, are only supported in Sherpa 2.2.0. Unfortunately, the built-in AGC model is only supported in earlier versions of Sherpa, such as version 2.1.1. Thus we will have to generate and compare events from two different versions of Sherpa. These versions of Sherpa have slightly different defaults, such as particle masses, and parton PDF. For example, the default PDF in Sherpa 2.1.1 is *CT10*, but in Sherpa 2.2.0 it is *NNPDF30*. These differences need to be aligned in order to reduce differences between the two versions. In general, we have chosen to align to Sherpa version 2.2.0 where possible. Section 3.3.3 contains more details on the run-cards used and the alignment choices made.

We have chosen parton-level WZ-production for comparing models, as this process includes many of the vertices affected by EFT and AGC.

For observables, we will focus on three phase-space observables, $P_T(Z)$ – the transverse momentum of Z, M(WZ) – the invariant mass of the WZ-pair, and y(Z) – the rapidity of Z.

3.3.1 Sherpa 2.2.0 SM (UFO) vs. Sherpa 2.1.1 SM (AGC)

We need to gauge any differences between the built-in Sherpa 2.1.1 SM and our UFO SM implementation, so as to ensure that the SM parts of our two models are nearly identical, before we can measure the non-SM parts. To do this events were generated using Sherpa 2.2.0 with our EFT UFO model and with Sherpa 2.1.1 with its built-in SM+AGC model, where all model parameters were set to their SM values. A summary of the generated event datasets is shown in Table 3.4.

Figure 3.1 shows a comparison of the events generated by the two models for our three chosen observables. For comparison, the histograms for each model have been scaled to a luminosity 10 fb^{-1}. Various comparison statistics have been generated and are both shown in the figure and summarized in Table 3.5. Note that the errors shown in the figure are scaled by the same factor used to scale the content to the common luminosity.

The first comparison statistic is a Kolmogorov-Smirnov test, that gives the probability of equality for two distributions. Second, is a standard χ^2-test between the two distributions. A ratio of the two distributions is shown in the lower-half of the figure, and two χ^2-fits are performed to two horizontal-lines, one fixed to a value of one, and the other parameterized by a free constant. The fit results give a third and fourth comparison statistic. The fit parameter of the second fit result also gives us a fifth statistic.

Note that bins with less than 10 events, before luminosity scaling, are excluded from the statistics, and marked on the figure with a circle around the bin value (unless the bin is empty). This is done to ensure that bins with low statistics do not bias the results. Most of our observables have long sparse tails in either end of their data range that result in low statistic bins. The count of 10 was chosen as it is more than 3 sigma from the value zero, given $\sqrt{n}$ errors.

Table 3.4: Sherpa event datasets for testing parton-level WZ-production with SM models.

Dataset	**Sherpa version**	**Number of Events**	**Cross-section [pb]**	**Effective Luminosity [fb^{-1}]**
SM-EFT	2.2.0	1M	18.543 ± 0.015	53.929 ± 0.044
SM-AGC	2.1.1	1M	18.554 ± 0.016	53.897 ± 0.046

Table 3.5: Comparison statistics of SM+EFT to SM+AGC model with SM model parameters.

Observable	**Kolmogorov-Smirnov**	χ^2/ndf	**Ratio Fit to 1** χ^2/ndf	**Ratio Fit to c** χ^2/ndf	**Ratio Fit to c Value**
$P_T(Z)$	0.752	0.9597	0.9807	0.9822	1 ± 0.0014
M(WZ)	0.754	0.8803	0.9035	0.9029	1 ± 0.0014
y(Z)	0.0491	1.24	1.211	1.216	1 ± 0.0014

The $P_T(Z)$ observable indicates a very good match between the two models, with Kolmogorov-Smirnov probabilities of 0.75 and χ^2/ndf results fairly close to 1. However, both the M(WZ) and y(Z) observables match less well, though still fairly good. For these two observables, the small differences between the two models are often outside the error-bars, resulting in a poorer statistical match. All-in-all, there is a good match between the two SM models. Differences for $P_T(Z)$ are within expected statistical fluctuations and less than two sigma. The small but larger difference in

M(WZ) and y(Z) are less than three sigma, and thus not large enough to conclude that the two models are not equivalent.

The larger differences in M(WZ) and y(Z) could be systematic, possibly caused by a slightly different phase-space integration optimization prior to the generation of the events in the two datasets.

We conclude that the SM-parts of the two models are close to equivalent, with only a small systematic error.

3.3.2 SM+EFT vs. SM+AGC

We can now compare and validate the non-SM part of our SM+EFT model with the SM+AGC model, confident that the SM-parts of the two models are nearly equivalent. To test our three EFT parameters, separately and together, we generated four datasets for each model using equivalent EFT and AGC model parameters, as shown in Table 3.6 and Table 3.7. We chose EFT parameters that approximately double the SM cross-section for 1M events, to ensure datasets with a visible and quantifiable difference from the SM.

Table 3.6: Equivalent EFT and AGC model parameters for testing the SM+EFT model.

EFT Test	EFT Model Parameters			AGC Model Parameters			
	c_{WWW} [GeV^{-2}]	c_W [GeV^{-2}]	c_B [GeV^{-2}]	$\lambda_{\gamma/Z}$	Δg_1^Z	$\Delta\kappa_\gamma$	$\Delta\kappa_Z$
c_{WWW}	3×10^{-5}	0	0	0.127275	0	0	0
c_W	0	5×10^{-5}	0	0	0.207879	0.161544	0.161544
c_B	0	0	9×10^{-4}	0	0	2.907787	−0.834044
all	3×10^{-5}	5×10^{-5}	9×10^{-4}	0.127275	0.207879	3.069330	−0.672500

Table 3.7: Sherpa event datasets for testing parton-level WZ-production for SM+AGC and SM+EFT models.

EFT Test	Dataset	Sherpa version	Number of Events	Cross-section [pb]	Effective Luminosity [fb^{-1}]
c_{WWW}	EFT cWWW	2.2.0	1M	30.426 ± 0.027	32.867 ± 0.029
	AGC lambda	2.1.1	1M	30.466 ± 0.027	32.823 ± 0.029
c_W	EFT cW	2.2.0	1M	30.717 ± 0.027	32.555 ± 0.029
	AGC g1	2.1.1	1M	30.715 ± 0.027	32.557 ± 0.029
c_B	EFT cB	2.2.0	1M	31.671 ± 0.026	31.575 ± 0.026
	AGC kappa	2.1.1	1M	31.725 ± 0.026	31.521 ± 0.026
all	EFT all	2.2.0	1M	42.805 ± 0.038	23.362 ± 0.021
	AGC all	2.1.1	1M	42.829 ± 0.038	23.349 ± 0.021

Table 3.8: Comparison statistics of SM+EFT to SM+AGC model with equivalent model parameters.

EFT Test	Observable	Kolmogorov-Smirnov	χ^2/ndf	Ratio Fit to 1 χ^2/ndf	Ratio Fit to c χ^2/ndf	Ratio Fit to c Value
c_{WWW}	$P_T(Z)$	0.877	1.042	1.046	1.040	1 ± 0.0015
	M(WZ)	0.285	1.026	1.058	1.047	0.99 ± 0.0014

	y(Z)	0.560	1.081	1.094	1.090	1 ± 0.0014
	P_T(Z)	0.891	0.9811	0.9863	0.9843	1 ± 0.0015
c_W	M(WZ)	0.990	1.015	1.029	1.024	1 ± 0.0014
	y(Z)	0.798	1.030	1.066	1.071	1 ± 0.0014
	P_T(Z)	0.935	0.9757	1.025	1.015	1 ± 0.0014
c_B	M(WZ)	0.105	0.9191	0.9685	0.9587	1 ± 0.0014
	y(Z)	0.147	1.101	1.093	1.084	1 ± 0.0014
	P_T(Z)	0.693	1.065	1.064	1.061	1 ± 0.0015
all	M(WZ)	0.761	0.9599	0.9924	0.9853	1 ± 0.0014
	y(Z)	0.383	0.9523	0.944	0.946	1 ± 0.0014

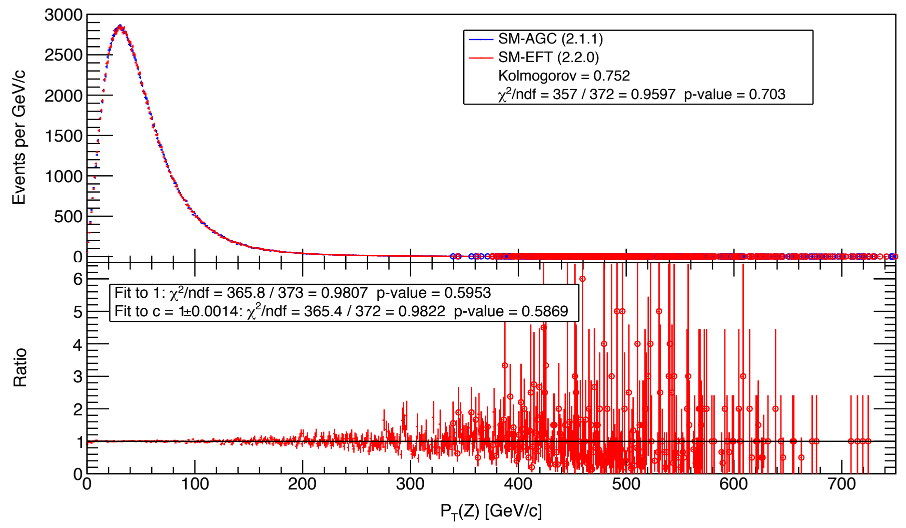
Comparisons of the resulting distributions for our four tests are shown in Figure 3.2 through Figure 3.5, and the comparison statistics are summarized in Table 3.8. See section 3.3.1 for comments on the comparison statistics. Note that the errors shown in the figures are scaled by the same factor used to scale the content to the common luminosity.

For all tests and observables, the resulting comparisons shows a very high degree of equivalence. Kolmogorov-Smirnov probabilities vary from ca. 0.10 to 0.99. χ^2/ndf results vary from ca. 0.92 to 1.10 with p-values from 0.06 to 0.96. Most observables have differences less than two sigma, and a few have less than three.

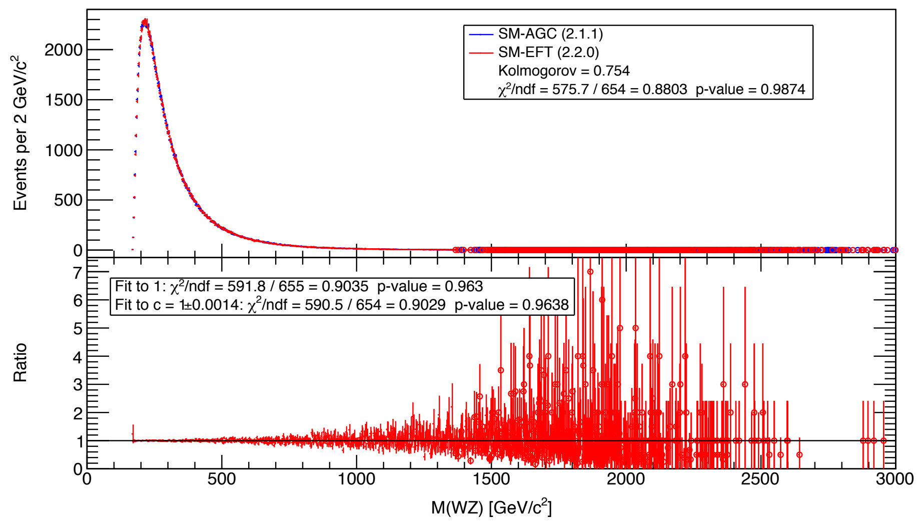
The comparison results are slightly better than when only the SM-parts were compared in section 3.3.1, particularly for y(Z). This might possibly indicate that the "noise" from the differences in SM-parts has been reduced by the the addition of equivalent EFT and AGC distributions.

We conclude, that our EFT model and the AGC model are statistically equivalent. We can now use our UFO EFT model trusting in its implementation.

(a)



(b)



(c)

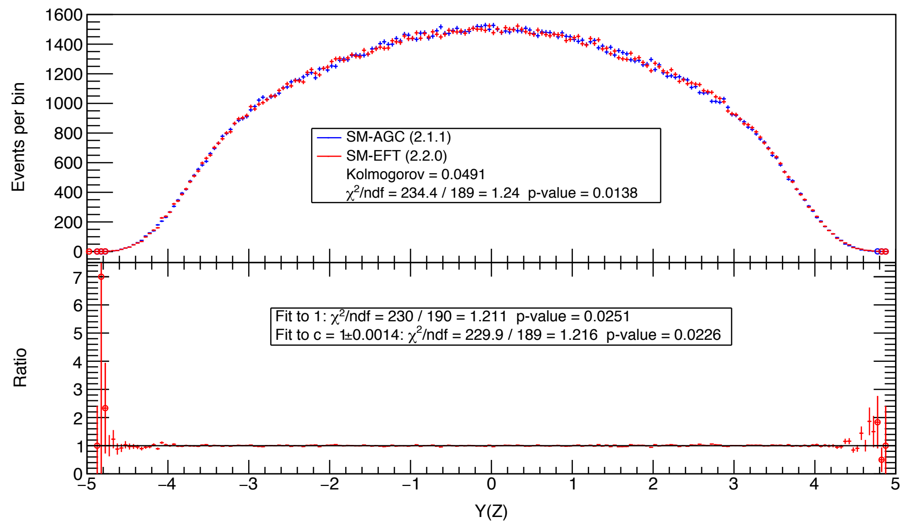


Figure 3.1: Comparison of SM (EFT) to SM (AGC) for parton-level WZ-production for: (a) P_T(Z), (b) M(WZ) and (c) y(Z).

The upper-half of each sub-figure shows the event distributions for both models scaled to a luminosity of 10 fb^{-1} from 1M generated events, as well as Kolmogorov-Smirnov and χ^2 comparison statistics. The lower-halves show the ratio of the two distributions, as well as fit results for fitting the ratio to one and to a constant. Bins with less than 10 events (before luminosity scaling) are excluded from all statistics and marked with circles. Events were generated using SM values for model parameter using the SM+EFT (Sherpa 2.2.0) and SM+AGC (Sherpa 2.1.1) models. Bin widths are (a) 1 GeV/c, (b) 2 GeV/c^2 and (c) 0.05.

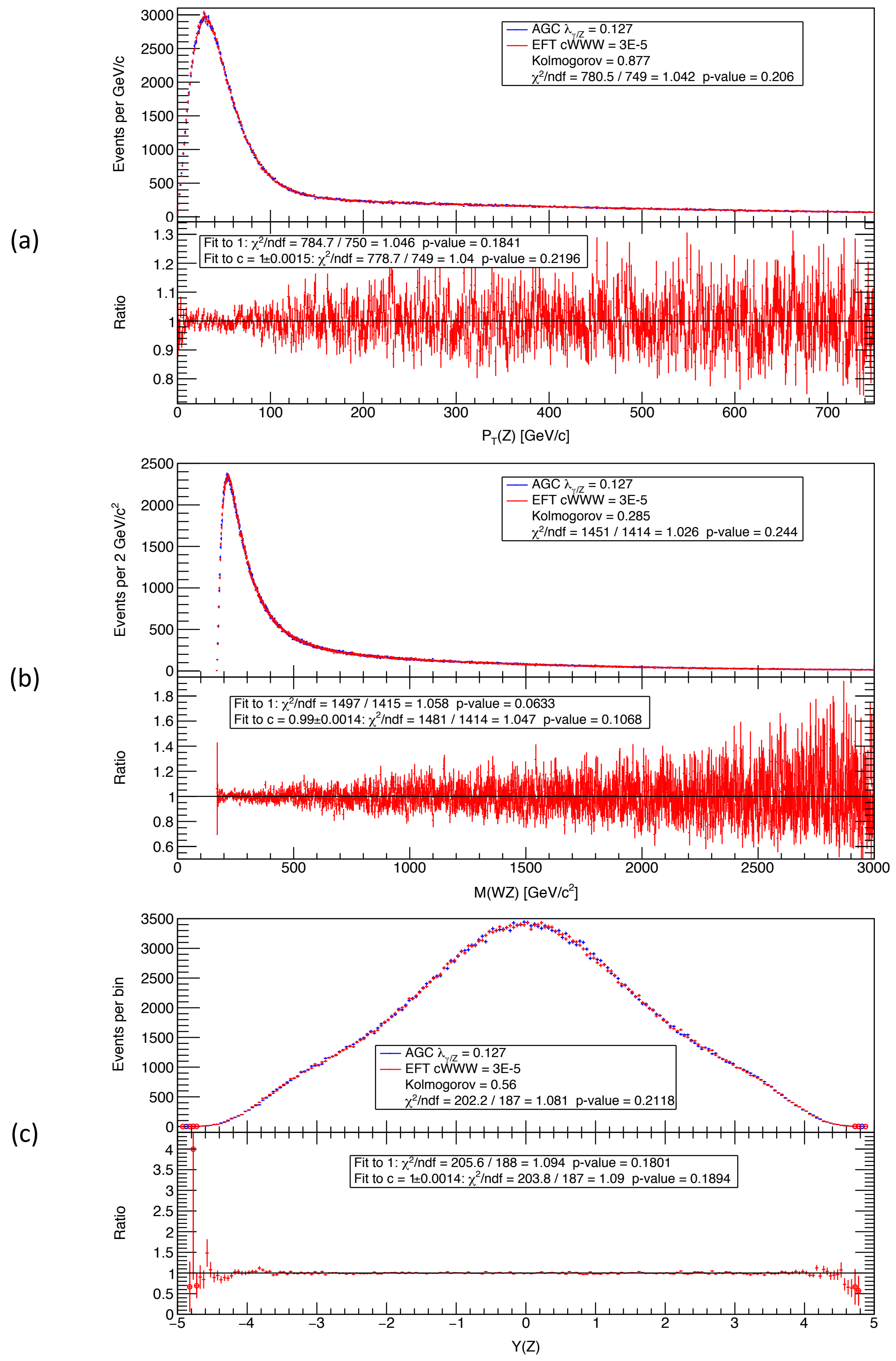
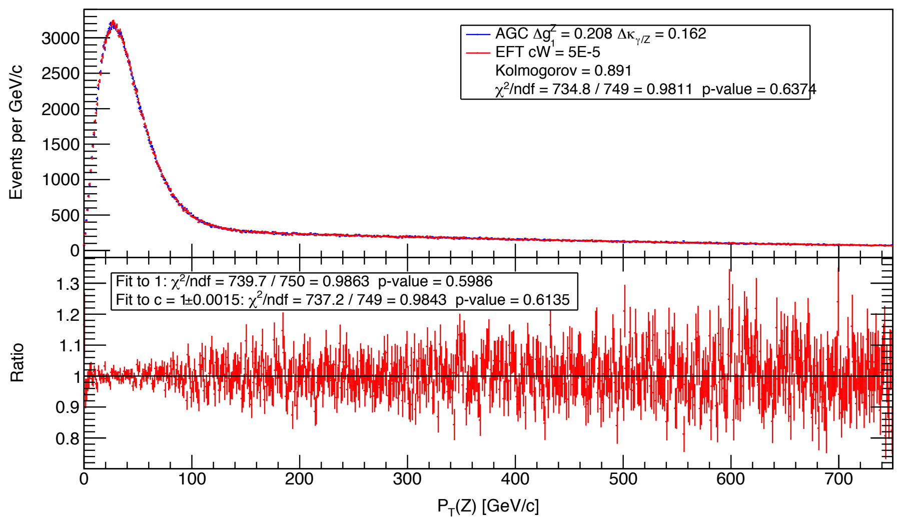
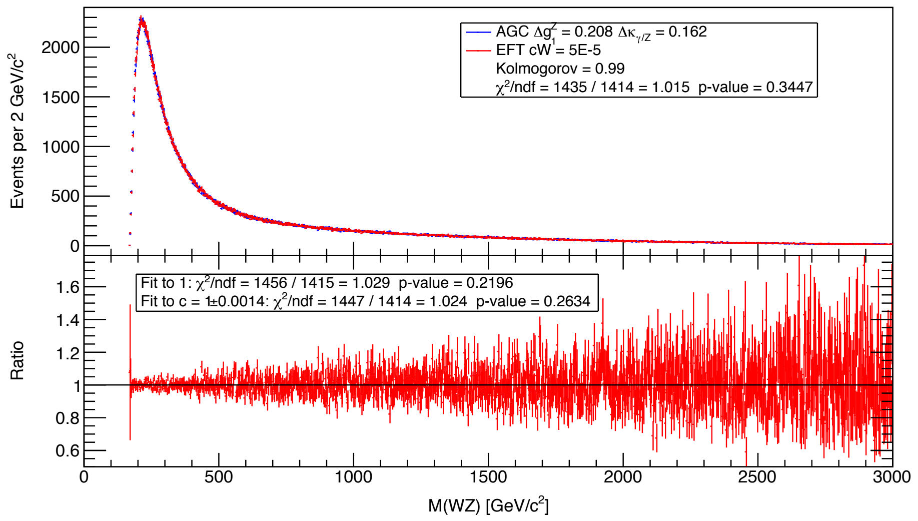


Figure 3.2: Comparison of SM+EFT to SM+AGC for parton-level WZ-production, with c_{WWW} and $\lambda_{\gamma/Z}$ set to equivalent non-SM values, for: (a) P_T(Z), (b) M(WZ) and (c) y(Z).
See Table 3.6 for model parameter values. The upper-half of each sub-figure shows the event distributions for both models scaled to a luminosity of 10 fb^{-1} from 1M generated events, as well as Kolmogorov-Smirnov and χ^2 comparison statistics. The lower-halves show the ratio of the two distributions, as well as fit results for fitting the ratio to one and to a constant. Bins with less than 10 events (before luminosity scaling) are excluded from all statistics and marked with circles. Events were generated for SM+EFT and SM+AGC using Sherpa version 2.2.0 and 2.1.1 respectively. Bin widths are (a) 1 GeV/c, (b) 2 GeV/c^2 and (c) 0.05.

(a)



(b)



(c)

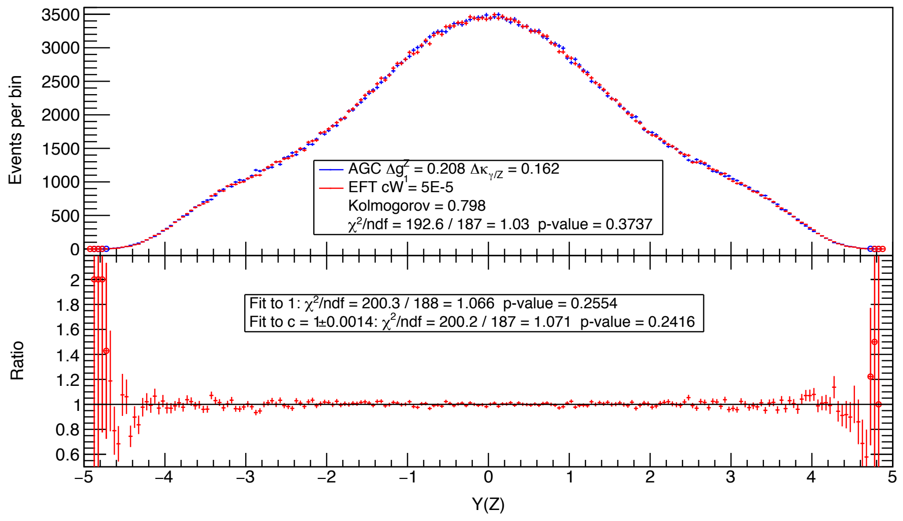
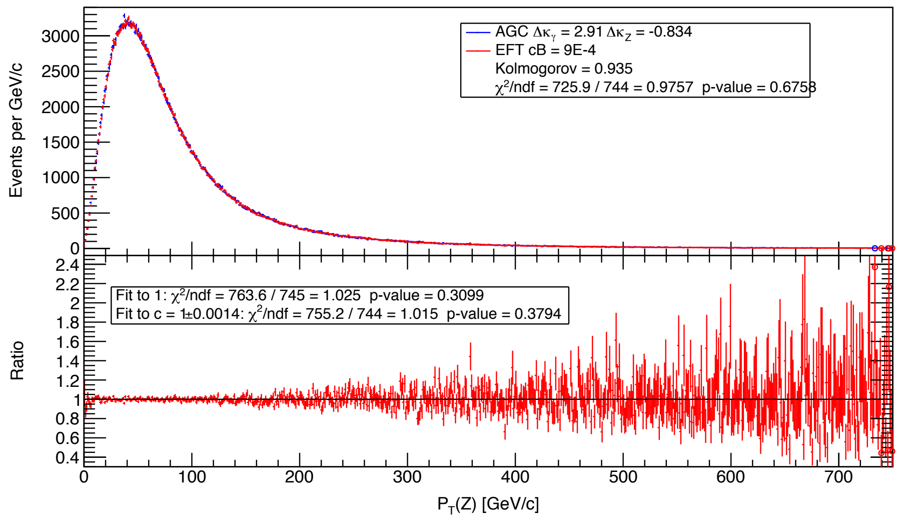


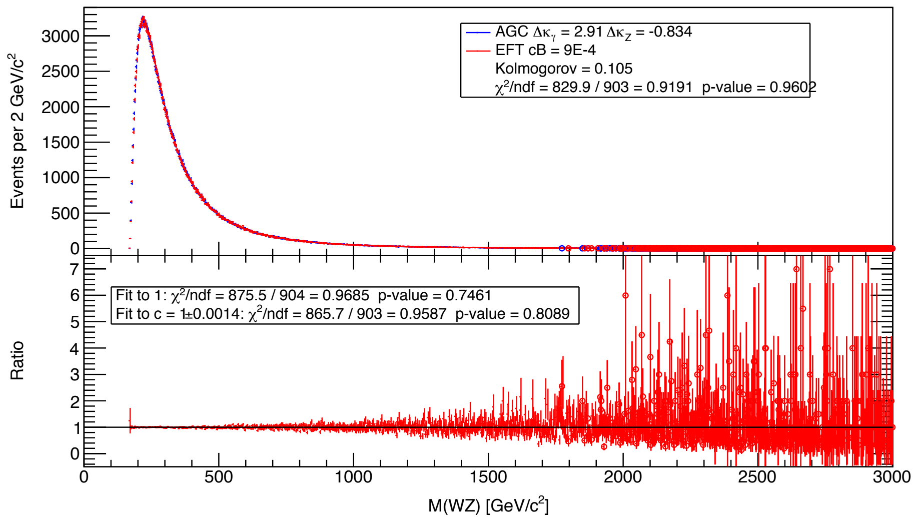
Figure 3.3: Comparison of SM+EFT to SM+AGC for parton-level WZ-production, with c_W , Δg_1^Z and $\Delta\kappa_{\gamma/Z}$ set to equivalent non-SM values, for: (a) $P_T(Z)$, (b) M(WZ) and (c) y(Z).

See Table 3.6 for model parameter values. The upper-half of each sub-figure shows the event distributions for both models scaled to a luminosity of 10 fb^{-1} from 1M generated events, as well as Kolmogorov-Smirnov and χ^2 comparison statistics. The lower-halves show the ratio of the two distributions, as well as fit results for fitting the ratio to one and to a constant. Bins with less than 10 events (before luminosity scaling) are excluded from all statistics and marked with circles. Events were generated for SM+EFT and SM+AGC using Sherpa version 2.2.0 and 2.1.1 respectively. Bin widths are (a) 1 GeV/c, (b) 2 GeV/c^2 and (c) 0.05.

(a)



(b)



(c)

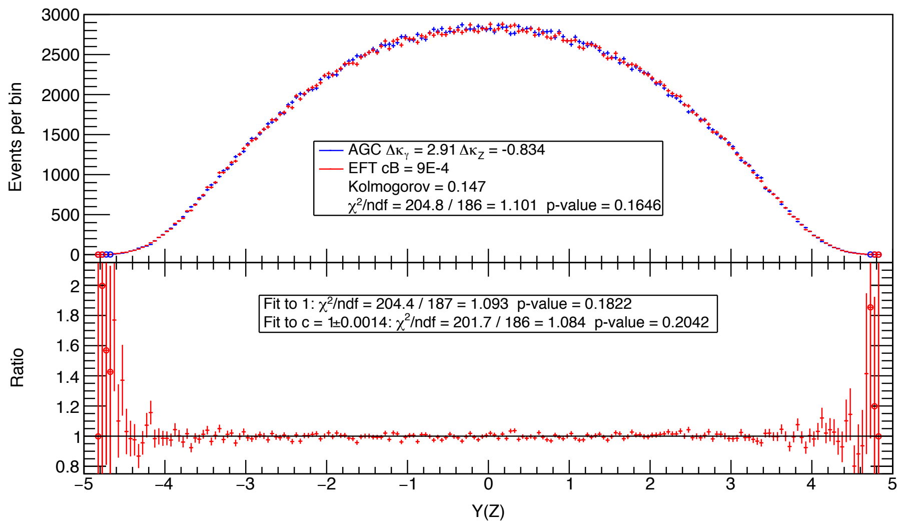
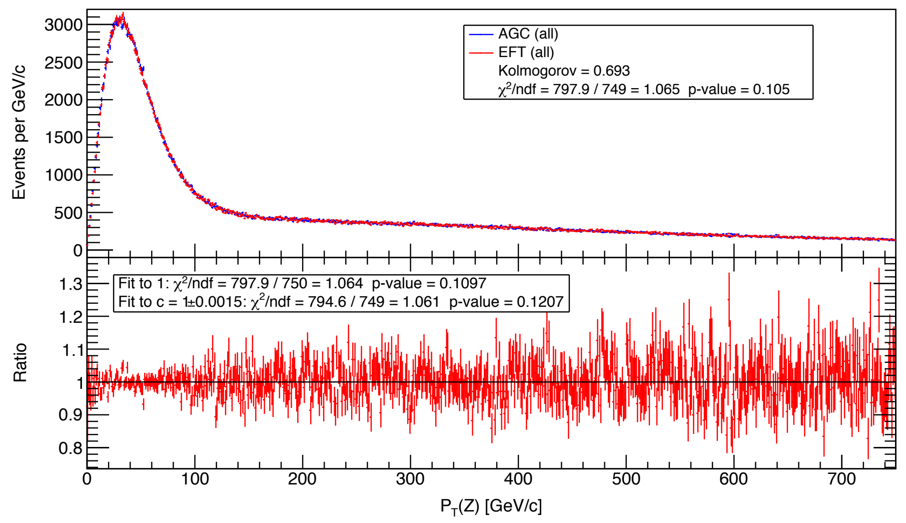


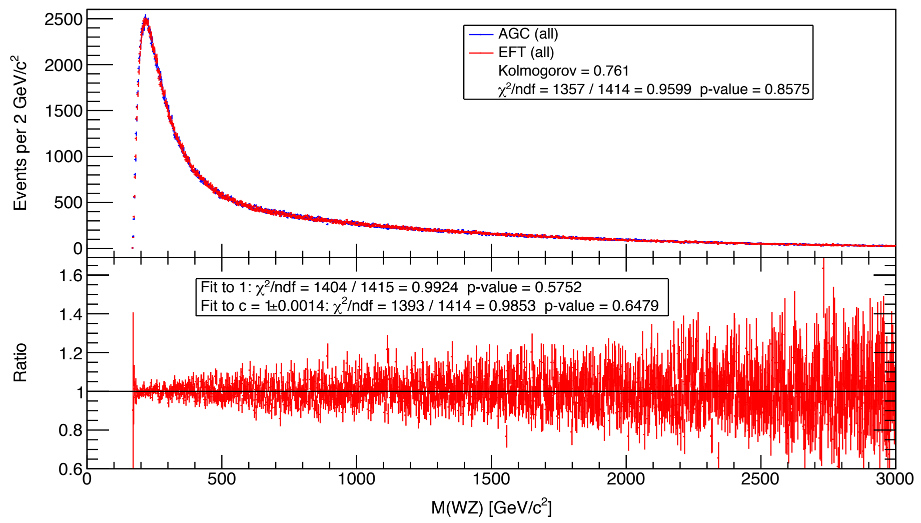
Figure 3.4: Comparison of SM+EFT to SM+AGC for parton-level WZ-production, with c_B and $\Delta\kappa_{\gamma/Z}$ set to equivalent non-SM values, for: (a) P_T(Z), (b) M(WZ) and (c) y(Z).

See Table 3.6 for model parameter values. The upper-half of each sub-figure shows the event distributions for both models scaled to a luminosity of 10 fb^{-1} from 1M generated events, as well as Kolmogorov-Smirnov and χ^2 comparison statistics. The lower-halves show the ratio of the two distributions, as well as fit results for fitting the ratio to one and to a constant. Bins with less than 10 events (before luminosity scaling) are excluded from all statistics and marked with circles. Events were generated for SM+EFT and SM+AGC using Sherpa version 2.2.0 and 2.1.1 respectively. Bin widths are (a) 1 GeV/c, (b) 2 GeV/c^2 and (c) 0.05.

(a)



(b)



(c)

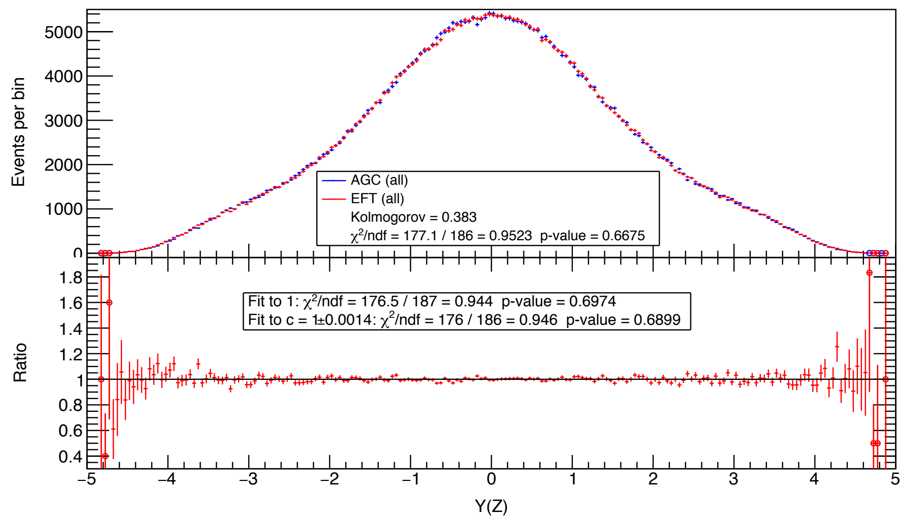


Figure 3.5: Comparison of SM+EFT to SM+AGC for parton-level WZ-production, with all parameters set to equivalent non-SM values, for: (a) $P_T(Z)$, (b) M(WZ) and (c) y(Z). See Table 3.6 for model parameter values. The upper-half of each sub-figure shows the event distributions for both models scaled to a luminosity of 10 fb^{-1} from 1M generated events, as well as Kolmogorov-Smirnov and χ^2 comparison statistics. The lower-halves show the ratio of the two distributions, as well as fit results for fitting the ratio to one and to a constant. Bins with less than 10 events (before luminosity scaling) are excluded from all statistics and marked with circles. Events were generated for SM+EFT and SM+AGC using Sherpa version 2.2.0 and 2.1.1 respectively. Bin widths are (a) 1 GeV/c, (b) 2 GeV/c^2 and (c) 0.05.

3.3.3 Sherpa run-cards for event generation of parton-level WZ-production

To ensure alignment between events generated with Sherpa 2.1.1 and 2.2.0, we chose to align Sherpa 2.1.1 run-cards to the Sherpa 2.2.0 defaults, except with regards to the parton PDF.

For the parton PDF we chose to use the Sherpa 2.1.1 default, as NNPDF30 is not available in Sherpa 2.1.1. One could use the LHAPDF library to import NNPDF30 into Sherpa 2.1.1, but experience has proven that this is not so easy a task. Many internal parameters are tuned to the PDF automatically, when a built-in PDF is used. When using LHAPDF to load an external PDF, one must tune these parameters oneself. Further, in our experience Sherpa tends to crash when using a PDF via the LHAPDF library.

The following run-cards were used for generating events for WZ-production in Sherpa 2.1.1 and Sherpa 2.2.0.

3.3.3.1 Sherpa 2.1.1 SM+AGC run-card

This run card was used for both Sherpa 2.1.1 SM and SM+AGC event generation, by appropriately setting the AGC parameters.

```
(run){
  # general settings
  EVENTS 0;
  EVENT_OUTPUT HepMC_GenEvent[AGC_211]

  PRETTY_PRINT Off

  # me generator setup
  ME_SIGNAL_GENERATOR Amegic;
  SCALES VAR{Abs2(p[0]+p[1])};

  # model setup
  MODEL SM+AGC
  CKMORDER 1

  LAMBDA_GAMMA = 1.272753587483230E-01    # cWWW = 3E-5
  LAMBDA_Z     = 1.272753587483230E-01

  KAPPA_GAMMA  = 4.069330406875000E+00    # cB = 9E-4
  KAPPA_Z      = 3.275001296830000E-01

  G1_Z         = 1.207879459844000E+00    # cW = 5E-5

  UNITARIZATION_SCALE  = 1E9
  UNITARIZATION_SCALE3 = 1E9
  UNITARIZATION_SCALE4 = 1E9

  # align particle defaults to Sherpa version 2.2.0

  MASS[6]  173.21;  WIDTH[6]  2.0;          STABLE[6]  0
                    WIDTH[15] 2.26735e-12;
  MASS[23] 91.1876; WIDTH[23] 2.4952;       STABLE[23] 0
  MASS[24] 80.385;  WIDTH[24] 2.085;        STABLE[24] 0
                                            STABLE[25] 0

  # set PDF to Sherpa version 2.1.1 default
  PDF_LIBRARY CT10Sherpa
  PDF_SET     ct10

  # generate only parton-level process
  MI_HANDLER        None
  FRAGMENTATION     Off
  SHOWER_GENERATOR  None
  ME_QED            Off
  BEAM_REMNANTS     0

  # LHC beam setup (13 TeV)
  BEAM_1 2212; BEAM_ENERGY_1 6500;
  BEAM_2 2212; BEAM_ENERGY_2 6500;
}(run)

(processes){
  Process 93 93 -> 24 23
  End process;
}(processes)
```

The relevant AGC model parameters are set via `LAMBDA_GAMMA`, `LAMBDA_Z`, `KAPPA_GAMMA`, `KAPPA_Z`, and `G1_Z`. The default values used by the SM are 0 for the lambda parameters, and 1 for the kappa and G1 parameters.

The three `UNITARIZATION_SCALE` parameters are used by the SM+AGC model to set an upper energy cut-off that the ACG contributions are valid up to. As no such cut-off is applied to our UFO model, we have effectively disabled this by setting the AGC energy cut-off to 10^9 GeV, well above any energy in our project.

The signal generator (`ME_SIGNAL_GENERATOR`) is set to `Amegic`, as this is required to use the SM+AGC model.

`SCALES`, defines the renormalization and factorization scale, and is set to the invariant mass of the two incoming partons. The scale must be manually set when disabling showers.

`CKMORDER` is set to 1, to disable a unitary CKM matrix, as described in section 3.2.

The `MASS`, `WIDTH`, and `STABLE` parameters are used to align the particle configuration to that used in Sherpa 2.2.0.

The remaining parameters, setup a 13 TeV proton-proton beam, set the process to WZ-production, and disable all but parton-level particles in the output file.

3.3.3.2 Sherpa 2.2.0 UFO SM+EFT run-card

This run card was used for both Sherpa 2.2.0 SM and SM+EFT event generation, by appropriately setting the EFT parameters.

```
 (run){
  # general settings
  EVENTS 0;
  EVENT_OUTPUT HepMC_GenEvent[EFT_220]

  PRETTY_PRINT Off

  # me generator setup
  ME_SIGNAL_GENERATOR Comix;
  SCALES VAR{Abs2(p[0]+p[1])};

  # model setup
  MODEL SM_EFT

  # set PDF to Sherpa version 2.1.1 default
  PDF_LIBRARY CT10Sherpa
  PDF_SET     ct10

  # set stable to 0 for those particles with a non-zero decay-width
  STABLE[6]  0
  STABLE[23] 0
  STABLE[24] 0
  STABLE[25] 0

  # disable ghost particles, which are not used by the model anyway (unitary gauge)
  ACTIVE[82]     0
  ACTIVE[250]     0
  ACTIVE[251]     0
  ACTIVE[9000001] 0
  ACTIVE[9000002] 0
  ACTIVE[9000003] 0
  ACTIVE[9000004] 0

  # generate only parton-level process
  MI_HANDLER None;
  FRAGMENTATION Off;
  SHOWER_GENERATOR None;
  ME_QED Off
  BEAM_REMNANTS 0

  # LHC beam setup (13 TeV)
  BEAM_1 2212; BEAM_ENERGY_1 6500;
  BEAM_2 2212; BEAM_ENERGY_2 6500;
}(run)

(processes){
  Process 93 93 -> 24 23
  End process;
}(processes)

(ufo){
block yukawa
    6 173.21                   # ymt
block ckmblock
    1 0.2272                   # cabi
block mass
    23 91.1876                 # MZ
    6  173.21                  # MT
    25 125                     # MH
block sminputs
    1 128.8022421894320        # aEWM1
    2 1.197448274494500E-05    # Gf
    3 0.118                    # aS
block eftcoef
    1 3E-5                     # ocWWW
    2 5E-5                     # ocW
    3 9E-4                     # ocB

decay 23 2.4952
decay 24 2.085
decay 6  2.0
decay 25 0.00407
}(ufo)
```

The three EFT model parameters are set via the `eftcoef` block parameters in the `(ufo)` section. The standard model values for all three parameters is zero.

The signal generator (`ME_SIGNAL_GENERATOR`) is set to `Comix`, as this is required to use a UFO model.

`SCALES`, defines the renormalization and factorization scale, and is set to the invariant mass of the two incoming partons. The scale must be manually set when disabling showers.

The parton PDF is set to CT10 to align to the default PDF used by Sherpa 2.1.1. The `STABLE` and `ACTIVE` parameters align the UFO model to the same defaults used by Sherpa 2.2.0 built-in models, as discussed in section 3.2.

The remaining parameters, setup a 13 TeV proton-proton beam, set the process to WZ-production, and disable all but parton-level particles in the output file.

4 Implementing Reweighting

SherpaWeight can be used to add reweight coefficients to the event datasets. However, we also need a library of routines that can read our event data and the added reweight coefficients, and generate reweighted event histograms for any specified set of model parameters. We developed such a reweighting library and made use of our EFT model to test both *SherpaWeight* and the reweighting library.

In this section, we will give a brief overview of this reweighting library and test its capability to reweight EFT event data to SM model parameters, comparing the reweight result to SM event data. Further, we will give some notes on developing reweighting code using ROOT.

The library of reweighting and histogram support methods was developed in C++ with the assistance of ROOT and HepMC libraries. All the classes and methods are generic and can be re-used in any application. To test the routines, we developed a test application called *ReweightEFT*.

The reweighting library contains the following functionality:

1. Classes to create a generic set of observables and generic methods for filling histograms or profiles with observable data.

   Note that we desire to examine different observables both in histogram but also profile form. A profile is a histogram displaying the mean and error-of-the-mean of the content in each bin, instead of simply the sum. We use ROOT's *TProfile* and *TH1D* classes to store profiles and histograms respectively.

   In general, we use the term histogram to refer to both histograms and profiles.

2. Classes to define model event datasets, including cross-section information.

3. Methods to load observable histograms from event datasets, with optional cache support for higher performance on subsequent runs (important). HepMC and gzip compressed HepMC event files are supported.

   In addition to the observable histogram, these methods also construct reweight-coefficient histograms from the reweight-coefficients stored with each event. Thus a set of observable and reweight-coefficient histograms are constructed for each model dataset and observable.

4. A method for scaling the luminosity of a histogram to any specified value, based upon the cross-section and number of events in the model dataset definition.

   This method takes care to scale the errors such that the result after scaling is a pseudo-dataset, aka. an Asimov dataset. See the discussion in sections 4.2 and 4.3.2.

5. A method to create a reweighted histogram from the set of reweight-coefficient histograms and a set of model parameters.

   Care must be taken in how the errors are defined for the resulting reweighted histogram. Our method recalculates the errors following reweighting, resulting in a pseudo-dataset, aka. an Asimov dataset. See the discussion in sections 4.2 and 4.3.3.

6. Methods for creating comparison figures, calculating ratios of histograms, and comparison statistics.

   These methods were also used in section 3.3 to construct the comparison figures used there.
7. General histogram helper methods, such as logging histogram data, measuring different histogram statistics, loading and saving histograms to ROOT data files.

We will not describe in detail all of the above functionality, however, source code is available for download as described in the introduction. In section 4.3, notes are given regarding the implementation of certain key elements of reweighting using ROOT classes.

4.1 Adding reweight-coefficients to SM+EFT datasets

Before we can reweight the events in a dataset, reweight-coefficients must be calculated and stored with the events. Here is where *SherpaWeight* is used.

Adding the reweight-coefficients to events is simply a matter of running SherpaWeight with an input and output event file and adding a `(SherpaWeight)` section to the run-card used to generate the events, as described in 2.3.

We chose to test reweighting using the same SM+EFT generated events used in section 3.3 for parton-level WZ-production, specifically the dataset where all EFT parameters were set to non-SM values.

The following `(SherpaWeight)` section was added to the run-card used to generate SM+EFT events detailed in section 3.3.3.2:

```
(SherpaWeight){
   ocWWW  0  1E-5
   ocW    0  1E-5
   ocB    0  1E-4
}(SherpaWeight)
```

Here we made use of *SherpaWeight's* support for using the comment following a `(ufo)` block parameter as the name of the parameter. As a reminder, the two terms following the parameter name are the center-value and the delta-value.

Running *SherpaWeight* on the 1M event file for SM+EFT, ran through 10 different configurations of EFT parameters for all events, and took about 53 minutes, on a 2.8 GHz Mac Book Pro with SSD hard-drive. Note that reading and writing 1M HepMC events are the main bottlenecks.

4.2 Comparison of reweighted EFT to SM for WZ-production

In this section, we examine the results of reweighting a SM+EFT dataset, generated with non-SM parameter values, to a reweighted dataset with SM parameter values. We compare the reweighted SM dataset with a generated SM dataset, using the same observables and comparison statistics used previously in section 3.3.

We have chosen to again use parton-level WZ-production as the test process, using the same SM+EFT dataset used previously in section 3.3. Specifically, the dataset where all EFT parameters are set to non-SM values. The selected EFT parameters used in this dataset more than double the cross-section of 1M generated events from 18 to 42 pb, so this dataset should be a good test of reweighting.

Reweighting was tested with 21 different observables (see Table 6.1), however, in this section we will only present the results for $P_T(Z)$ – the transverse momentum of Z, M(WZ) – the invariant mass of the WZ-pair, and y(Z) – the rapidity of Z. The results for the other observables are nearly identical to these three and to save space, these three will be used as representative.

The datasets we used are shown in Table 4.1. We reweighted the "EFT (all)" dataset, using the SM values of zero for all three EFT parameters. The reweight result was compared to a "SM-EFT" dataset that was generated separately with these same SM parameter values. Figure 4.1 shows the comparison results, with both datasets scaled to a luminosity of 10 fb^{-1}. The corresponding comparison statistics are summarized in Table 4.2. See section 3.3 for a description of the comparison statistics.

Table 4.1: Sherpa event datasets for testing reweighting of parton-level WZ-production.

Dataset	EFT Model Parameters			Number of Events	Cross-section [pb]	Effective Luminosity [fb^{-1}]
	c_{WWW} [GeV^{-2}]	c_W [GeV^{-2}]	c_B [GeV^{-2}]			
EFT (all)	3×10^{-5}	5×10^{-5}	9×10^{-4}	1M	42.805 ± 0.038	23.362 ± 0.021
SM-EFT	0	0	0	1M	18.543 ± 0.015	53.929 ± 0.044

Table 4.2: Comparison statistics of reweighting EFT events with SM model parameters to generated SM events, excluding low-statistic bins.

Observable	Kolmogorov-Smirnov	χ^2/ndf	Ratio Fit to 1 χ^2/ndf	Ratio Fit to c χ^2/ndf	Ratio Fit to c Value
$P_T(Z)$	0.999	0.1856	0.1991	0.1949	1 ± 0.0033
M(WZ)	0.854	0.2663	0.3400	0.3345	0.99 ± 0.0033
y(Z)	0.978	0.3029	0.3152	0.3084	1 ± 0.0033

The comparison results indicate a very high equivalence between reweighted EFT-to-SM events and directly generated SM events. The χ^2/ndf results are much lower than 1, giving p-values of 1 in every case. Note that the error-bars in the comparison figures in this section are larger than those shown for the same observables in section 3.3. This could help explain why the χ^2/ndf results are so low.

Bin-errors in this section have been recalculated after luminosity scaling and reweighting. The resulting distributions thus represent pseudo-data, otherwise known as an Asimov dataset, as the errors match those that would be given to the content if it had been measured or generated directly at the given luminosity. The simple reason for this is that bin-errors cannot be reweighted, and thus one must recalculate the bin-errors after reweighting. It follows that the generated SM dataset must have its errors recalculated as well, so that the comparison is pseudo-data to pseudo-data.

As before, bins with less than 10 events before luminosity scaling or reweighting are excluded from the comparison statistics, and marked with circles on the figures. We can see that these low-statistic bins result in many outliers in the ratio comparison, with large error-bars. In the three observables shown, these outliers also appear to cluster more to one side of the horizontal line drawn at a ratio of 1, than being evenly spread.

We can infer that we have a very good match in areas of high-statistics, but have problems in areas of low-statistics. This is one of the fundamental problems with reweighting. It is impossible to reweight an empty bin, and reweighting a bin with low-statistics is less certain to give a correct result.

Statistically, though including the low-statistic bins does not make much difference in the results. The comparison statistics calculated from all bins, including those with low-statistics, are shown in Table 4.3. The results are fairly much the same as those where low-statistic bins are excluded – all χ^2 p-values are still 1. Even though the χ^2 statistic increases from the outliers, so do the number of degrees of freedom. Statistics obviously does not tell the whole picture; one must also visually assess the results.

Table 4.3: Comparison statistics of reweighting EFT events with SM model parameters to generated SM events, including low-statistic bins.

Observable	**Kolmogorov-Smirnov**	χ^2/ndf	**Ratio Fit to 1** χ^2/ndf	**Ratio Fit to c** χ^2/ndf	**Ratio Fit to c Value**
$P_T(Z)$	0.999	0.144	0.1756	0.1723	1 ± 0.0033
M(WZ)	0.796	0.234	0.7493	0.7342	0.99 ± 0.0033
y(Z)	0.976	0.3033	0.3413	0.3344	1 ± 0.0033

We conclude that reweighting produces highly accurate results in areas of high statistics.

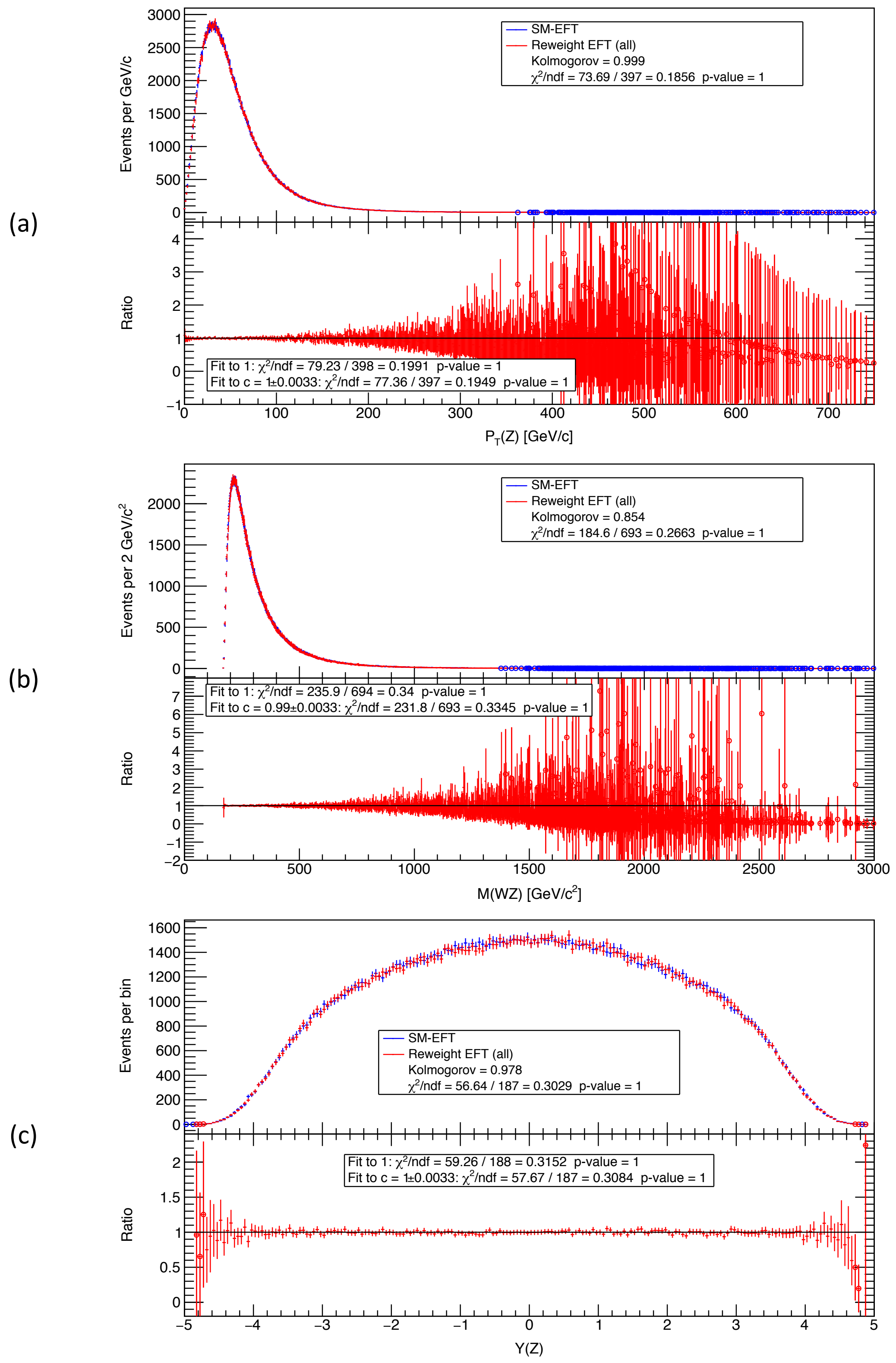


Figure 4.1: Comparison of an EFT dataset reweighted to SM vs. a SM dataset, for parton-level WZ-production, for: (a) P_T(Z), (b) M(WZ) and (c) y(Z).

See Table 4.1 for model parameter values for each dataset. The upper-half of each sub-figure shows the event distributions for both models scaled to a luminosity of 10 fb^{-1} from 1M generated events, as well as Kolmogorov-Smirnov and χ^2 comparison statistics. The lower-halves show the ratio of the two distributions, as well as fit results for fitting the ratio to one and to a constant. Bins with less than 10 events (before luminosity scaling) are excluded from all statistics and marked with circles. Bin widths are (a) 1 GeV/c, (b) 2 GeV/c^2 and (c) 0.05.

4.3 Notes on reweighting using ROOT

4.3.1 Histograms and profiles

As previously mentioned, ROOT classes *TH1D* and *TProfile* were used to store and provide the main functionality for histograms and profiles. Generally, the classes were used as is, however, some instances helper methods were developed to handle special cases, such as luminosity scaling and re-weighting.

Each bin of a *TH1D* histogram contains two simple sums of the event-weights for the bins events:

$$S_1 = \sum_i w_i \qquad (\textit{TH1D} \text{ and } \textit{TProfile}) \qquad (4.1)$$

$$S_2 = \sum_i w_i^2 \qquad (\textit{TH1D} \text{ and } \textit{TProfile}) \qquad (4.2)$$

The bin content of a *TH1D* is given by S_1, and the bin error by $\sigma = \sqrt{S_2}$. The S_2 sum can be disabled, resulting in the bin error being defined as $\sigma = \sqrt{S_1}$, but generally we always enabled it.

For both *TH1D* and *TProfile*, the effective number of entries in a bin is:

$$n_{eff} = \frac{S_1^2}{S_2} \qquad (\textit{TH1D} \text{ and } \textit{TProfile}) \qquad (4.3)$$

In the case where all event-weights are 1, then $S_1 = S_2 = n_{events} = n_{eff}$.

Each bin of a *TProfile* profile contains the same sums as a *TH1D*, as well as two others containing summing the y-values associated with each bin-event:

$$S_3 = \sum_i w_i\, y_i \qquad (\textit{TProfile}) \qquad (4.4)$$

$$S_4 = \sum_i w_i\, y_i^2 \qquad (\textit{TProfile}) \qquad (4.5)$$

The bin content of *TProfile* is defined as the weighted mean μ of the bins y-values. The default bin error is by default defined to be the standard-deviation-of-the-mean (SDOM) $\hat{\sigma}$ given by:

$$\mu = \frac{S_3}{S_1} \qquad (\textit{TProfile}) \qquad (4.6)$$

$$\sigma = \sqrt{\frac{S_4}{S_1} - \mu^2} \;\; ; \quad \hat{\sigma} = \frac{\sigma}{\sqrt{n_{eff}}} \qquad (\textit{TProfile}) \qquad (4.7)$$

Note *TProfile* supports different bin error definitions, with *SDOM* as the default, which we use exclusively in this project.

4.3.2 Scaling luminosity

To compare the event counts of different Monte-Carlo datasets, either to each other or to experimental data, they must first be scaled to a common luminosity. Normally, model data is scaled to the luminosity of experimental data. In our case, all our data is MC-generated, and we chose to compare at a luminosity of 10 fb^{-1}.

The effective luminosity of a generated dataset is given by:

$$L_{eff} = \frac{N_{gen}}{\sigma_{gen}} \tag{4.8}$$

Where N_{gen} is the number of events generated and σ_{gen} is the integrated cross-section for the generated events. Note that the effective luminosity for a dataset is unchanged, even if events are later cut by additional selection criteria.

To scale a dataset from its effective luminosity to a desired luminosity *L*, one scales the effective number of events in the dataset by the scaling factor:

$$l_s = \frac{L}{L_{eff}} \tag{4.9}$$

TH1D and *TProfile* have a method called *Scale* that on the surface would appear to be useful for doing the luminosity scaling. *Scale* modifies the the sums stored in *TH1D* and *TProfile* in the following manner:

$$\begin{array}{lll} S_1' = l_s\, S_1 & S_2' = l_s^2\, S_2 & (\mathit{TH1D}) \\ S_3' = l_s\, S_3 & S_4' = l_s^2\, S_4 & (\mathit{TProfile}) \end{array} \tag{4.10}$$

Firstly, the *Scale* method does not actually change number of effective entries defined by (4.3). Secondly, it scales the mean bin values for a *TProfile*, which are independent of luminosity, and should not change.

To perform luminosity scaling we implemented our own method called *ScaleHistToLuminosity*, which scales all sums identically:

$$\begin{array}{lll} S_1' = l_s\, S_1 & S_2' = l_s\, S_2 & (\mathit{TH1D}) \\ S_3' = l_s\, S_3 & S_4' = l_s\, S_4 & (\mathit{TProfile}) \end{array} \tag{4.11}$$

This simulates a change to the number of events that went into a sum, but does not change to the distribution of weights or y-values for those events. It has the desired effect that the number of effective entries in each bin is scaled by l_s, and the mean values for a *TProfile* are not affected.

$$n'_{eff} = \frac{l_s^2\, S_1^2}{l_s\, S_2} = l_s\, n_{eff} \qquad (\mathit{TH1D} \text{ and } \mathit{TProfile}) \tag{4.12}$$

Our method also scales bin-errors in a way that maintains a pseudo-dataset, or Asimov dataset as introduced previously. That is the resulting bin-errors are as if the dataset had been generated at the desired luminosity in the first place. The scaled bin errors are:

$$\sigma' = \sqrt{l_s\, S_2} = \sqrt{l_s}\,\sigma \qquad (\mathit{TH1D}) \tag{4.13}$$

$$\sigma' = \sqrt{\frac{l_s\, S_4}{l_s\, S_1} - \mu^2} = \sigma \;\; ; \quad \hat{\sigma}' = \frac{\sigma}{\sqrt{l_s\, n_{eff}}} = \frac{\hat{\sigma}}{\sqrt{l_s}} \qquad (\mathit{TProfile}) \tag{4.14}$$

For *TH1D* bins, our method scales bin errors with$\sqrt{l_s}$, instead of with l_s as implemented by the *Scale* method. For TProfile bins, the bin errors scale with $1/\sqrt{l_s}$. Normally, we are scaling to a lower luminosity than the datasets's effective luminosity. In this case $l_s < 1$, and we are reducing the number of effective events, and the errors on *TH1D* bins are reduced, and those on *TProfiles* are increased.

4.3.3 Reweighting histograms

To create a reweighted histogram or profile, we can simply use the relative probability for an event, for a given set of reweight-parameter values, as the weight of each the event, as described in section 1.1.2. However, there is a much faster method to create reweighted histograms and profiles, without running through all the events each time the model parameters are changed. This is particularly important when fitting reweighted models to data, as fits need to iterate through many different candidate parameter values to find the ones that minimize the desired objective. This faster method is described in this section, as well as how to calculating bin-errors for reweighted event distributions.

The matrix element for a given event can be calculated from (1.6) through (1.8), repeated here for convenience:

$$\bar{f} = [F_{0,0} \quad F_{0,1} \quad F_{0,2} \quad \cdots \quad F_{1,1} \quad F_{1,2} \quad \cdots \quad F_{2,2} \quad F_{2,3} \quad \cdots \quad F_{N_P,N_P}]^T \tag{1.6}$$

$$\bar{a} = \begin{bmatrix}1 & \phi_1 & \phi_2 & \cdots & \phi_1^2 & \phi_1\phi_2 & \cdots & \phi_2^2 & \phi_2\phi_3 & \cdots & \phi_{N_P}^2\end{bmatrix} \tag{1.7}$$

$$|ME|^2 = \bar{a}\bar{f} \tag{1.8}$$

Where $\bar{f}$ is the reweight-coefficient vector for each event, and $\bar{a}$ is an evaluation vector derived from appropriate combinations of the values of the reweight-parameters. Both vectors contain N_F terms. We can use *SherpaWeight* to calculate and embed the reweight-coefficient vector for each event in the datasets we wish to reweight.

The event-weight for each event is given by:

$$w_i = \frac{(|ME|^2)_{rw}}{(|ME|^2)_{gen}} = \frac{\left(\bar{a}\bar{f}\right)_{rw}}{(|ME|^2)_{gen}} \tag{4.15}$$

Where $(|ME|^2)_{rw}$ and $(|ME|^2)_{gen}$ are respectively the reweighted and generated matrix-elements. The generated matrix-element is calculated by using the same reweight-parameter values that the events were generated with.

Note that both $\bar{f}$ and $(|ME|^2)_{gen}$ are constant for each event, and event-weights only depend upon the evaluation vector $\bar{a}$, which depends upon reweight-parameters. If we take the evaluation vector out of (4.15), we end up with a constant weight for each coefficient, and we can create separate histograms and profiles for each reweight-coefficient using this constant weight. As these reweight-coefficient histograms are independent of the choice of reweight-parameters, they do not need to be recalculated

Then to acquire a final reweighted event-distribution, we simply multiply each reweight-coefficient histogram by its corresponding term in the evaluation vector and sum all the histograms together. This simple weighted-sum operation can be repeated for each iteration of reweight-parameter values, without loading the event dataset. It is extremely fast, taking only milliseconds to construct a reweighted event-distribution.

One would be tempted to use the class methods *Scale* and *Add* available in *TH1D* and *TProfile to perform the weighted-sum of coefficient histograms*. However, they both suffer from the same issues introduced in the previous section. ROOT's methods are principally designed to be mathematically correct for the purposes of applying equations to histograms, and propagating errors. However, we want to construct a final reweighted histogram from the weighted-sum of several other histograms, in such a way that the reweighted result is equivalent to the result that we would have achieved if constructing the histogram in one pass. In particular, we want the number of effective entries to be the same, whether we calculate a reweighted histogram directly, or via a weighted-sum of histograms.

As an example, if we subtract a histogram from itself using ROOT's methods, we will not get an empty histogram, with all internal sums empty. Sums of squares will remain, and for *TProfile* in particular the number of effective entries will double. We need to be able to handle negative as well as positive contributions to the number of effective entries to get the correct result. ROOT's methods are not designed for this.

In fact, it is impossible to obtain the same bin errors when using a weighted-sum of coefficient histograms as compared to constructing a single histogram directly, using TH1D and TProfile. This is because only sums, such as S_1, S_3 and S_4, that use event-weights linearly can be constructed identically by both methods. The sum S2, which squares the event-weights is different when calculated by the two methods. This is shown in the following equations for a linear and quadratic sum of weights:

$$Sum(w) = \sum_i w_i\, c_i = \sum_i \left(\sum_k a_k\, f_{i,k} \right) c_i = \sum_k a_k \left(\sum_i f_{i,k}\, c_i \right) \qquad (4.16)$$

$$Sum(w^2) = \sum_i w_i^2\, c_i = \sum_i \left(\sum_k a_k\, f_{i,k} \right)^2 c_i \neq \sum_k a_k \left(\sum_i f_{i,k}^2\, c_i \right) \qquad (4.17)$$

Where i is an event index, and k is a reweight-coefficient index. c_i is an arbitrary term, which is 1 in S_1 and S_2 and y_i and y_i^2 in S_3 and S_4.nOn the left, the sums are acquired by calculating the weight using the reweight-parameters and the coefficients for each event, and summing over events. On the right, the sums are acquired by using the reweight-coefficients alone as the weight, summing over events, and then creating a weighted-sum using the reweight-parameters. The right-hand-side represents our optimized reweighting method.

Fortunately, we are not interested in a reweighted histogram, where the bin-errors are calculated from our weights. We are interested in reconstructing a new dataset that is representative of what would have been generated with the new set of reweight-parameter values was used. We require a pseudo-dataset where the bin-errors correspond to the errors that would be there if the bin was filled with weights of one.

We developed our own methods called *HistReweightAdd* and *CalculateReweightHist* to perform the weighted-sum of histograms and create the resulting reweighted histogram. These resulting weighting can be summarized as follows:

$$S_1 = \sum_k a_k \left(\sum_i f_{i,k} \right) \tag{4.18}$$

$$S_2 = S_1 \tag{4.19}$$

$$S_3 = \sum_k a_k \left(\sum_i f_{i,k}\, y_i \right) \tag{4.20}$$

$$S_4 = \sum_k a_k \left(\sum_i f_{i,k}\, y_i^2 \right) \tag{4.21}$$

Effectively, we perform the weighted-sums as described, but do so by modifying the internal sums maintained within TH1D and TProfile. Only S_2 is treated differently, and is set to the same value as S_1, which is equivalent to creating a distribution with event-weights of 1. This has the effect that the bin-errors for a reweighted *TH1D* is simply $\sigma = \sqrt{S_1}$, and the SDOM for a TProfile is calculated using $n_{eff} = S_1$. This gives us the pseudo-data result that we want.

5 Optimal Observables

In this section we introduce a new observable type, called optimal observables, which we will use later during our fit analysis. Optimal observables are designed to give the smallest possible statistical error on model parameters. They are very useful for determining values or constraints for a set of model parameters from experimental data, or calculating theoretical sensitivities.

5.1 Derivation of first-order optimal observables

In the following, we present the derivation of optimal observables given in [13] and [14]. Optimal observables can be used with any process where the differential cross-section is dependent upon a set of parameters with small values. For a given process the differential cross-section is given by:

$$S(\phi) \equiv \frac{d\sigma}{d\phi} \tag{5.1}$$

Where ϕ is the set of all measured phase-space variables, for example, the incoming and outgoing four-momenta for the process. For an isolated hard-process, $S(\phi)$ is equivalent to the matrix-element for the process.

$S(\phi)$ is a polynomial of second-order in terms of EFT parameters as well as TGC parameters:

$$S(\phi) = S_0(\phi) + \sum_i S_{1i}(\phi)\, h_i + \sum_{i,j} S_{2ij}(\phi)\, h_i h_j \tag{5.2}$$

Where h_i are the model parameters, $S_0(\phi)$ is the differential cross-section in the SM, and $S_{1i}(\phi)$ and $S_{2ij}(\phi)$ are the first and second-order modifications. For an isolated hard-process, this equation is essentially equivalent to (1.3) presented in our discussion on reweighting, where ϕ_i in (1.3) is h_i in (5.3).

One way one can measure the parameters is to look for a set of observables $\mathcal{O}_i$ whose expectation values are sensitive to the dependence of $S(\phi)$ on h_i:

$$E[\mathcal{O}_i] = \frac{1}{\sigma}\int d\phi\; S(\phi)\, \mathcal{O}_i(\phi) \tag{5.3}$$

Where $\sigma = \int d\sigma = \int d\phi\; S(\phi)$ is the total cross-section. Restricting ourselves to first-order, for the time-being, and combining (5.2) and (5.3) we get:

$$E[\mathcal{O}_i] = E_0(\mathcal{O}_i) + \sum_j c_{ij} h_j + O(h^2) \tag{5.4}$$

With:

$$E_0(\mathcal{O}_i) = \frac{1}{\sigma_0}\int d\phi\; S_0(\phi)\, \mathcal{O}_i(\phi) \tag{5.5}$$

$$c_{ij} = \frac{1}{\sigma_0}\int d\phi\; \mathcal{O}_i(\phi)\, S_{1j}(\phi) - \frac{\sigma_{1j}}{\sigma_0^2}\int d\phi\; S_0(\phi)\, \mathcal{O}_i(\phi) \tag{5.6}$$

$$\sigma_0 = \int d\phi\; S_0(\phi) \tag{5.7}$$

$$\sigma_{1j} = \int d\phi \, S_{1j}(\phi) \tag{5.8}$$

Where $E_0(\mathcal{O}_i)$ is the expectation value for no model parameters, or for EFT, when all parameter values are zero. c_{ij} is the sensitivity of $E[\mathcal{O}_i]$ to h_j.

Solving (5.4) for the parameters, we get estimators for h_j whose covariance matrix, using matrix notation, is:

$$V(h) = \frac{1}{N}\,(c^{-1})\,V(\mathcal{O})\,(c^{-1})^T \tag{5.9}$$

Where *N* is the number of events, *c* is a constant matrix, and $V(\mathcal{O})$ is the the covariance matrix of the observables, expanded around its value in the SM:

$$V(\mathcal{O})_{ij} = \frac{1}{\sigma_0}\int d\phi\, S_0(\phi)\, \mathcal{O}_i(\phi)\, \mathcal{O}_j(\phi) - E_0(\mathcal{O}_i)\, E_0\big(\mathcal{O}_j\big) + O(h) \tag{5.10}$$

For observables, if we chose:

$$\mathcal{O}_i(\phi) = \frac{S_{1j}(\phi)}{S_0(\phi)} \tag{5.11}$$

This gives us from (5.6) and (5.10) the covariance matrixes:

$$V(\mathcal{O}) = c + O(h) \tag{5.12}$$

$$V(h) = \frac{1}{N}\,(c^{-1}) + O(h) \tag{5.13}$$

The observables in (5.11) are "optimal" in that for $h_i \to 0$, the errors on the parameters (5.13) are as small as they can be for a probability distribution [16]. Further, phase space cuts, detector efficiency and acceptance effects cancel in the ratio in (5.11), and thus do not affect the optimal observables. However, detector resolution effects, if significant, may distort the distributions $S_{1j}(\phi)$ and $S_0(\phi)$ in a way that does not cancel in the ratio, and should still be taken into account, to avoid biasing the estimators.

5.2 Derivation of second-order optimal observables

We can also define a second-order optimal observable. By definition, (5.2) is a second-order Taylor series expansion of $S(\phi)$ around $h_i = 0$. The optimal observable for h_i given in (5.11) is seen to be the first-order term for h_i, divided by the zeroth-term. As the first-order term is the partial-derivative of $S(\phi)$ with regards to h_i, and is most sensitive to h_i, it makes sense that the optimal observable for h_i would contain it. We can generalize this observation to define an optimal observable for h_i^2.

$$O_1(h_i) = \frac{\left.\frac{\partial S(\phi)}{\partial h_i}\right|_{h_i=0}}{S_0(\phi)} = \frac{S_{1j}(\phi)}{S_0(\phi)} \tag{5.14}$$

$$O_2(h_i) = \frac{\left.\frac{\partial^2 S(\phi)}{\partial h_i^2}\right|_{h_i=0}}{S_0(\phi)} = \frac{S_{2ii}(\phi)}{S_0(\phi)} \tag{5.15}$$

Where $O_1(h_i)$ and $O_2(h_i)$ are the first- and second-order optimal observables.

A detailed analysis on higher-order optimal observables is given in [17], where it is shown that although first-order optimal observables can be conclusively shown to give the least statistical error, the same cannot be said of second-order optimal observables. Higher-order optimal observables can be increased in sensitivity by expanding $S(\phi)$ around a good estimate for h_i. In our case, h_i is small and expanding around zero is a good estimate, so we expect that second-order optimal observables will still give a low statistical error, if not the minimum.

5.3 Optimal observables and reweight coefficients

For a single event $S(\phi)$ is proportional to the squared matrix-element for that event. Comparing the above equations to those for reweighting in section 1.1.1 (being careful not to confuse the phase-space ϕ used here with the model parameter ϕ_i used there), we find that $S_0(\phi)$, $S_{1j}(\phi)$ and $S_{2ij}(\phi)$ are proportional to the terms of the reweight-coefficient matrix **F**. Thus for each event the first- and second-order optimal observables for a given reweight-parameter are given by:

$$O_1(h_i) = \frac{F_{0i}}{F_{00}} \tag{5.16}$$

$$O_2(h_i) = \frac{F_{ii}}{F_{00}} \tag{5.17}$$

As the reweight-coefficient matrix **F** is constant for a specific event, so are $O_1(h_i)$ and $O_2(h_i)$ fixed for this event, just as all other observables are. Like all other observables, $O_1(h_i)$ and $O_2(h_i)$ have their own event distributions for a given process. Further, these event-distributions depend upon all model-parameters, not just the model-parameter associated with the optimal observable. The reweight-coefficients for an event given by the matrix, not only determine the values of the optimal observables for that event, they also determine the weight of the event for a given set of model-parameters.

5.4 Optimal observables in this project

In section 7, when we perform our fit analysis for reweighted EFT, we will examine the fit characteristics of optimal observables for our three EFT model parameters.

As stated above, the expectation values of an optimal observable is sensitive to the dependence of $S(\phi)$ on h_i. Therefore, we will also use the mean of each optimal observable $E[\mathcal{O}_i]$, plotting it versus the invariant mass of events, as another set of observables for our fit analysis.

Finally, we will divide each events optimal observable by the invariant mass for the event to see if by factoring out an energy dependency, if we can improve the results. For second-order optimal observables we divide by invariant mass squared.

6 Chosen observables for parton-level WZ-production

In the following section we will measure the fit characteristics of reweighting EFT to SM data. To do so we need to select some observables to perform fits with. In addition to a handful of phase-space observables, we will also examine optimal observables. We also need to select for each observable, the range and bin sizes, in an as objective a way as possible, as these will affect the fit results to some degree.

We will continue to use the SM and EFT data samples used previously. Both samples contain 1M events, which we will scale to a luminosity of 10 fb^{-1}. Using only the SM data sample to determine data range and binning, the chosen observables and their characteristics are shown in Table 6.1.

We have chosen 3 phase-space observables, $P_T(Z)$ – the transverse momentum of Z, M(WZ) – the invariant mass of the WZ-pair, and y(Z) – the rapidity of Z. We have also added the first- and second-order optimal observables for the 3 EFT model parameters c_{WWW}, c_W, and c_B to our table of observables.

Since fitting to the mean of the optimal observables should give the least statistical error, we have included in our table of observables, profile histograms of the mean of each optimal observable plotted against the invariant mass of the hard-process. Note the Mandelstam variable s used here corresponds to the hard-process and not the proton beams.

In addition, as each optimal observable is a function of the invariant mass, we have divided out that dependency for each event, and included profile histograms of the mean across events of each optimal observable divided by their invariant mass dependency, again plotted against the invariant mass. These profiles should be nearly constant, or asymptotic to a constant at high energy.

In summary, the selected objectives can be divided into two broad groups: event-counted objectives stored in histograms, and mean-objectives, which are stored in profiles that are effectively equivalent to a data graph with vertical error bars.

Table 6.1: Observables for parton-level WZ-production, characteristics and selected range.

Observable		Units y-axis	Units x-axis	Mean [a] (x-axis)	Std. Dev. [a] (x-axis)	IQR [a] (x-axis)	Selected Range (x-axis)	Under/ overflow Count [b]
Phase-space observables								
1	$P_T(Z)$		GeV/c	54.6	42.5	41.7	0 to 750	0 / 8.2
2	M(WZ) = $\sqrt{s}$		GeV/c^2	325	167	138	0 to 3000	0 / 6.3
3	y(Z)			1.24×10^{-4}	2.02	3.21	−5 to +5	0 / 0
Optimal observables								
4	$O_1(c_{WWW})$		GeV^2	−1340	5660	3210	(−6 to 1) $\times10^4$	6.7 / 0.2
5	$O_2(c_{WWW})$		GeV^4	8.13×10^8	4.90×10^{10}	8.35×10^7	(0 to 14) $\times10^{11}$	0 / 9.5
6	$O_1(c_W)$		GeV^2	−5540	26700	7280	(−12 to 1) $\times10^5$	7.6 / 0
7	$O_2(c_W)$		GeV^4	3.93×10^8	1.86×10^{10}	8.08×10^7	(0 to 53) $\times10^{10}$	0 / 9.8
8	$O_1(c_B)$		GeV^2	184	574	418	(−1 to 5) $\times10^3$	0.2 / 8.2
9	$O_2(c_B)$		GeV^4	6.78×10^5	2.76×10^6	4.00×10^5	(0 to 13) $\times10^7$	0 / 9.1
Mean optimal observables vs. invariant mass								
10	$\langle O_1(c_{WWW})\rangle$ vs $\sqrt{s}$	GeV^2	GeV	Same as M(WZ)				

11	$\langle O_2(c_{WWW})\rangle$ vs $\sqrt{s}$	GeV^4	GeV	
12	$\langle O_1(c_W)\rangle$ vs $\sqrt{s}$	GeV^2	GeV	
13	$\langle O_2(c_W)\rangle$ vs $\sqrt{s}$	GeV^4	GeV	
14	$\langle O_1(c_B)\rangle$ vs $\sqrt{s}$	GeV^2	GeV	
15	$\langle O_2(c_B)\rangle$ vs $\sqrt{s}$	GeV^4	GeV	
Mean optimal observables divided by powers of invariant mass vs. invariant mass				
16	$\langle O_1(c_{WWW})/s\rangle$ vs $\sqrt{s}$		GeV	Same as M(WZ)
17	$\langle O_2(c_{WWW})/s^2\rangle$ vs $\sqrt{s}$		GeV	
18	$\langle O_1(c_W)/s\rangle$ vs $\sqrt{s}$		GeV	
19	$\langle O_2(c_W)/s^2\rangle$ vs $\sqrt{s}$		GeV	
20	$\langle O_1(c_B)/s\rangle$ vs $\sqrt{s}$		GeV	
21	$\langle O_2(c_B)/s^2\rangle$ vs $\sqrt{s}$		GeV	

[a] For all events including those outside the selected range.

[b] SM data sample at 10 fb^{-1}.

6.1 Data range

Most of our observables have long tails approaching a sparse distribution of single events separated by areas with no events. Once binned, these sparse areas will result in bins with low statistics and empty bins. When fitting, all bins are treated equally and if we have a relatively high number of low statistic bins, these can dominate the fit results. With some fit objectives, such as minimizing χ^2, empty data bins are skipped, reducing the degrees of freedom for the fit.

One can solve such problems, by using different bin sizes across the data range to ensure that no bins have low statistics. This is discussed in the next section. For simplicity though, we have chosen to use equal-width bins for our histograms, thus sparse data tails remain an issue, and the data range must be carefully chosen.

To set the data range in a somewhat objective manner, we have chosen ranges that exclude less than 10 events on either end, after scaling to the test luminosity of 10 fb^{-1}. Thus the underflow and overflow bins will have less than 10 events. Why use 10 events as the criteria? Simply because, a bin with 10 events and thus an error of $\sqrt{10}$ has more than a 3 sigma probability of not having a true value of zero. Bins with less than 10 events have a small but significant probability of being zero.

In addition, we have set the range based solely upon the SM data sample, that is the data sample that we are fitting to and not the EFT data sample that we are reweighting. A common range and bin size must be used to perform binned fits. Normally one bins one's data (the SM sample) according to its characteristics, and then bins the model (the reweighted EFT sample) equivalently.

6.2 Binning for binned fits

All choices in binning affect fit results to some degree, and unfortunately it is not easy to gauge these effects. We desire that fit results remain fairly stable as a function of binning near our binning choice.

We have several choices when binning our data. The **first choice** is whether a single bin-width should be used across the entire range, equivalent to dividing the data range equally into a specified number of bins, or whether different bin-widths should be used through the data range.

The choice of a single bin-width is obviously the simplest. However, as mentioned in the previous section, our observables often have areas of sparse events at one or both ends of their data range. These areas will contain bins that are empty or with low statistics. One cannot reweight an empty bin, and bins with low statistics have larger relative errors. When fitting, all bins are treated equally and if we have a relatively high number of low statistic bins, these can dominate the fit results. Also with some fit objectives, such as minimizing χ^2, empty data bins are skipped, reducing the degrees of freedom for the fit.

One can solve such problems, by using different bin sizes across the data range to ensure that no bins have low statistics. Thus there will be more bins in areas of high density and fewer in areas of low density. One can divide each bin's height by its bin width, to create a density distribution, but this is not necessary for fitting, and will cannot be used with a likelihood fit objective that compares event counts. However, gauging the binning effects of a specific choice of variable bin-widths on a fit result is not an easy task.

For this project, we have chosen for simplicity to use equal-width bins for our histograms, despite the potential benefits of variable bin-widths. This means that we must be aware of the sparse regions in our data, and take care to handle empty and low-statistic bins appropriately. This is discussed further in the sections describing the fits performed.

One potential compromise between variable and fixed bin-widths, is to use fixed bin-widths, and then to fill in empty bins with events from adjacent non-empty bins, spreading the events evenly across the empty areas. This has the same affect of creating wider pseudo-bins locally where the density is lower, without actually changing the binning scheme. This obviously can only be done on model histograms and not data histograms, as data should never be modified. This technique is used where appropriate, and discussed further in sections describing the fits performed.

One can also obviously avoid low-statistic and empty bins by using more events, or fewer number of bins.

6.2.1 Optimal number of bins

The **second choice**, when binning our data using fixed bin-widths, is the number of bins to use across the data range. There are widely different suggestions on how to determine the optimal number of bins for a given data set [REF]. For example, there is the very simple square root choice, used by Microsoft Excel:

$$K = \sqrt{N} \tag{6.1}$$

where *K* is the number of bins, and *N* the number of data points. For 1M data points, this gives *k* = 1000.

There is *Rice's rule*, for approximately normal distributions:

$$K = 2N^{1/3} \tag{6.2}$$

which gives k = 200 for 1M data points.
There is *Scott's normal reference rule* for normal distributions:

$$W = \frac{2\sigma}{N^{1/3}} \tag{6.3}$$

where *W* is the optimal width and σ is the standard deviation of the data sample.
For non-normal distributions, there is the *Freedman-Diaconis rule*:

$$W = \frac{2\,IQR(x)}{N^{1/3}} \tag{6.4}$$

where IQR is the interquartile range of the data sample.
The above rules are straight-forward to calculate, and the results for Scott's and Freedman-Diaconis rule are included in table Table 6.2.

Somewhat more complicated are functions that when minimized give the optimal bin-width, or in our case the optimal number of bins. Here are two such functions, based upon minimizing the mean integrated square error (MISE). That is, they try to minimize the mean error between a binned histogram and the true continuous distribution it estimates. Since the true distribution is not known, they estimate the true distribution from the data itself.

The first method [REF], minimizes the MISE cost function given by:

$$C(W) = \frac{2\mu - v}{W^2} \tag{6.5}$$

Where $\mu = \frac{1}{K}\sum_{i=1}^{K} n_i$ is the mean events per bin, $v = \frac{1}{K}\sum_{i=1}^{K}(n_i - \mu)^2$ is the variance in the events per bin, n_i are the number of events in bin i, and K is the total number of bins.

The second method, know as the cross-validation method [REF], minimizes:

$$J(W) = \frac{1}{W(N-1)}\left(2 - \frac{N+1}{N^2}\sum_{i=1}^{K} n_i\right) \tag{6.6}$$

During testing, it was found that both functions effectively generate the same minimization curve, differing only in scale and offset, but having the same shape and minimal point. The scale of the cross-validation function was fairly independent of the observable, whereas the scale of the MISE cost function was highly dependent upon the observable chosen, mostly due to the higher power of the W^2 term in the denominator. Therefore, we chose to only minimize the cross-validation function.

The results of Scott's and the *Freedman-Diaconis rule* along with the results of minimizing the cross-validation function are shown in Table 6.2. The following section details how the cross-validation function was minimized.

6.2.1.1 Minimizing the cross-validation function

Initially, we attempted to use ROOT's fit methods to minimize the cross-validation function. To keep the function smooth, minimization was performed on a continuous number of bins, and all but the last bin were equal in size. For example, to achieve 12.2 bins, the range was divided by 12 equal sized bins, and one extra bin that was 0.2 the size of the other bins. As a cross-check, to ensure that we were minimizing the proper function parameter, we also ran the minimization using continuous bin-widths, instead of the number of bins. Fortunately, the results of minimizing the cross-validation function using either the number of bins or the bin-width as the function parameter were identical, as one would expect.

Using ROOT’s fit methods and its default Minuit minimizer, we attempted to find the optimal number of bins between 10 and 1M. The cross-validation function for most observables descends rapidly to a nearly constant plateau after about 100 bins. This plateau is nearly flat, descending very slowly as the number of bins increase, wobbling through numerous local minima, to an ultimate global minimum, before rising very slowly afterwards.

The nearly flat nature of this plateau, often resulted in ROOT’S fit function converging on some local minima. Hesse errors were enabled to make a parabolic estimate of the errors, and these were always close to the order of the entire allowed range of the number of bins. However, it was unclear whether we could rely on the error estimates at all, as they are dependent upon a Gaussian distribution around the converged upon parameter result, and the corresponding definition of the equivalent height in the cost function corresponding to the desired confidence level. The fit results were also fairly dependent upon the tolerance setting for the minimizer, the fit range, and the initial step.

We suspected that perhaps the issue is the continuous nature of the fit parameter, as opposed to the integer nature of the number of bins the original cross-validation objection function was derived with. In an attempt to solve this, we changed the objective function to return an interpolated result between the two nearest integer values for the number of bins. For example, if the minimizer requested 12.2 bins, we would calculate the cross-validation function for 12 and 13 bins, and return a weighted average using 80% of the 12 result and 20% of the 13 result. This helped smooth the minimization function somewhat, but not sufficiently that it helped solve the issues we were having in getting consistent fit results.

In the end, we concluded that using a minimizer to find the optimal number of bins was overly complicated and untrustworthy given the circumstances. Instead, we programmed a very simple scan of the cross-validation function across an allowed range of the number of bins using only integer values in steps that were humanly selected. For example, in the 10s we scanned by steps of 5, in the 100s by steps of 25, in the 1000s by steps of 250, and so on. We scanned from 10 to 1M bins and recorded the point with the minimal cross-validation value. The cross-validation results from this simple scan are shown in Table 6.2.

For the phase-space observables, reasonably low number of bins is chosen by this method. However, for the optimal observables, particularly the second-order, the number of bins is very high. The minimization plateau of two of second order optimal observables, was observed to continue descend past our upper limit of 1M bins. This casts some doubt on the usability of the cross-validation method for our data. The long flat plateau in its entirety can be seen as roughly the functions minimum, indicating that any number of bins from somewhere between 100 and 1000 to 1M could conceivably be used.

6.2.2 Selection of the number of bins

The different results for optimal bins by Scott's rule, the Freedman-Diaconis rule, and by minimizing the cross-validation function, are fairly varied. The broad minimization plateau for the cross-validation, suggests that a broad range of number of bins will be effectively the same. So despite all the mathematics to predict the optimal number of bins, we will end up choosing the number ourselves, guided by the mathematical suggestions.

The fewer bins we have the faster the performance of the model fits we will perform in the next section. In that interest, we have chosen to limit the number of bins to a maximum of 10000, which is a quite likely limit for most data samples. Finally, we have chosen numbers of bins that are rounded to humanly pleasing numbers, that fit the range nicely, and that are close to the suggested values. Considering the broad minimization plateau of the cross-validation function, this should not change the fit result significantly. The selected number of bins is shown in Table 6.2.

For the mean-value observables using profile histograms, the number of bins is selected to match the observable the mean is plotted against. In our case this is invariant mass, which is the same as M(WZ).

Table 6.2: Observables for parton-level WZ-production, optimal bins by different methods, and selected bins.

Observable		Optimal Bins			Selected Bins
		Scott's	Freedman-Diaconis	Cross-Validation	
Phase-space observables					
1	$P_T(Z)$	952	1700	900	750
2	M(WZ) = $\sqrt{s}$	709	1510	1750	1500
3	y(Z)	139	153	175	200
Optimal observables					
4	$O_1(c_{WWW})$	394	1220	2000	2000
5	$O_2(c_{WWW})$	16300	1.68×10^7	> 1E6	10000
6	$O_1(c_W)$	6110	39200	17500	10000
7	$O_2(c_W)$	16200	6.53×10^6	> 1E6	10000
8	$O_1(c_B)$	338	812	1250	1000
9	$O_2(c_B)$	6190	74900	325000	10000
Mean optimal observables vs. invariant mass					
10	$\langle O_1(c_{WWW})\rangle$ vs $\sqrt{s}$	Same as M(WZ)			
11	$\langle O_2(c_{WWW})\rangle$ vs $\sqrt{s}$				
12	$\langle O_1(c_W)\rangle$ vs $\sqrt{s}$				
13	$\langle O_2(c_W)\rangle$ vs $\sqrt{s}$				
14	$\langle O_1(c_B)\rangle$ vs $\sqrt{s}$				
15	$\langle O_2(c_B)\rangle$ vs $\sqrt{s}$				
Mean optimal observables divided by powers of invariant mass vs. invariant mass					
16	$\langle O_1(c_{WWW})/s\rangle$ vs $\sqrt{s}$	Same as M(WZ)			
17	$\langle O_2(c_{WWW})/s^2\rangle$ vs $\sqrt{s}$				
18	$\langle O_1(c_W)/s\rangle$ vs $\sqrt{s}$				
19	$\langle O_2(c_W)/s^2\rangle$ vs $\sqrt{s}$				
20	$\langle O_1(c_B)/s\rangle$ vs $\sqrt{s}$				
21	$\langle O_2(c_B)/s^2\rangle$ vs $\sqrt{s}$				

6.3 Binning for unbinned fits

One way to minimize the effects of binning on fit results is to do unbinned fits. In unbinned fits, only the model events are binned, and the individual likelihood for all data events is calculated and the sum maximized. We also performed unbinned fits on event counted observables in the following sections. The advantage of unbinned fits is that one no longer needs to bin the fit model with the same binning used for the fit data, one can use much higher resolution binning, improving the precision of the result.

For our unbinned fits, we have used 100K bins for the 1M model events for each observable.

7 Fitting EFT to SM for parton-level WZ-production

Now that we have the reweighting tools to construct the observable plots for any defined set of EFT model parameters, we can now measure the fit characteristics of reweighting EFT to SM data at the parton-level. Specifically, we can measure how well reweighted EFT data fits the SM data for each selected observable, using a fit algorithm to determine the EFT model parameters that fit best. We are primarily interested in the error to the parameters, as the true value is zero for all three parameters c_{WWW}, c_W, and c_B.

For the fits we used the same parton-level WZ-production pseudo-data generated by Sherpa used previously. The SM events are used as the fit-data, and the EFT events are reweighted and used as the fit-model. The SM and EFT data samples both have 1M events and their total cross-sections are 18.554 ± 0.016 pb and 42.805 ± 0.038 pb respectively. The data samples are scaled to a luminosity of 10 fb^{-1}, and Asimov bin errors are used as discussed in section 4.2. Note that the fit will only use the errors on the fit-data, that is the SM data.

We used ROOT's fit methods to perform all fits, and describe the details in section 7.3. We fit all objectives detailed in section 6, fitting each to each EFT parameter individually, as well as fitting to all 3 parameters simultaneously.

We performed two kinds of fits: binned and unbinned. Binned fits use the same binning for both the fit data and model, and are the traditional form of fit. For unbinned fits, we only bin the model, and maximize the log-likelihood of all 1M data events.

The choice of binning for binned and unbinned fits is described in detail in sections 6.2 and 6.3. Unbinned fits should be less affected by the choice of binning. Unfortunately, unbinned fits are far slower as all 1M fit-data events must be iterated through for each parameter selection during the fit process, as opposed to iterating through typically less than 1000 bins. Unbinned fits can only be done on event-counted observables.

For binned fits we used two different fit objectives: maximize log-likelihood (or rather minimize negative log-likelihood) for event-counted observables, and minimize χ^2 for mean-observables.

Note that the likelihood function for binned fits uses Poisson statistics to calculate the likelihood. However, the likelihood function for unbinned fits requires the fit-model histogram to be normalized to a constant value, and does not take the relative number of events into consideration.

It should be noted that in all cases fit data corresponding to empty bins in the fit model are skipped by the fit process, as it is impossible to reweight an empty bin. Empty bins in the fit data are also skipped when minimizing χ^2, as there is no error for an empty bin to calculate the denominator of the χ^2 value.

7.1 Binned fit results

The binned fit results are shown in Table 7.1, Table 7.2 and Table 7.3 for each of the EFT model parameters. One can immediately see that the lowest confidence-limits are acquired for mean optimal observables.

An unexpected result, is that the fits for mean optimal observables and mean optimal observables divided by invariant mass are virtually identical. At first a programming error was suspected, but the results are indeed correct. This means that one gains nothing by dividing the mean optimal observable by invariant mass.

The confidence-limits for fitting all parameters simultaneously are always higher than for fitting a single parameter at a time, which is not unexpected. This is probably only the case, as we used very good values (zero) for the parameters that were fixed. Thus for determining confidence-limits on a parameter, it is best to perform a single fit, and ensure the other parameters are fixed to good values. One could perform a multi-parameter fit to find the good values, before performing individual fits on each parameter.

Finally, several of the fit-values are outside their 95% confidence-limits. As the true value of all our parameters is zero, this is rather peculiar. It was thought that it could be due to the fit getting stuck in a local minimum. However, examination of the fit objective function around the minimum indicates that these are true global minima. More likely the results indicate some systematic error that is larger than the statistical error.

In terms of execution time, all the binned fits for all three EFT model parameters take a little over 10 seconds to execute on a fast Mac Book Pro (2.8 GHz i7), after the histogram data is read from the cached files they were saved to. The entire execution, including cache loading takes less than 15 seconds. This really demonstrates the power of reweighting.

Table 7.1: Binned fit results for c_{WWW}

Fit results for binned fitting of reweighted EFT to SM for c_{WWW} are listed, both for fitting only c_{WWW} and for fitting all parameters simultaneously. Values in parenthesis for 95% confidence-limits, are the asymmetric confidence-limits acquired by enabling MINOS errors for the fit. Fit-values in bold are larger than the 95% confidence limit.

Observable		**Fit only c_{WWW}**			**Fit All**		
		Value [10^{-6} GeV^2]	**95% CL [10^{-6} GeV^2]**	$\frac{\chi^2}{ndf}$	**Value [10^{-6} GeV^2]**	**95% CL [10^{-6} GeV^2]**	$\frac{\chi^2}{ndf}$
Phase-space observables							
1	$P_T(Z)$	−0.24	0.66 (−0.44,+0.94)	0.26	0.041	1.1 (−0.82,+0.76)	0.26
2	M(WZ) = $\sqrt{s}$	**−1.81**	0.24 (−0.23,+0.24)	0.41	−0.88	2.3 (−1,+2.8)	0.42
3	y(Z)	−0.91	1.5 (−1.1,+4.5)	0.6	0.41	5.7 (−4,+3.4)	0.61
Optimal observables							
4	$O_1(c_{WWW})$	−0.31	0.55 (−0.43,+0.89)	0.35	0.42	1.7 (−1.7,+0.91)	0.35
5	$O_2(c_{WWW})$	−0.26	0.54 (−0.4,+0.85)	0.023	0.12	1.5 (−1,+0.8)	0.023
6	$O_1(c_W)$	−0.30	0.50 (−0.38,+0.91)	0.082	−0.48	1.2 (−0.61,+1.6)	0.082
7	$O_2(c_W)$	−0.27	0.53 (−0.4,+0.83)	0.024	0.22	1.5 (−1.1,+0.72)	0.024
8	$O_1(c_B)$	−0.27	0.56 (−0.44,+0.85)	0.39	0.25	2.1 (−1.5,+1)	0.39
9	$O_2(c_B)$	−0.28	0.55 (−0.4,+0.95)	0.059	0.32	1.3 (−1.3,+0.68)	0.059
Mean optimal observables vs. invariant mass							
10	$\langle O_1(c_{WWW})\rangle$ vs $\sqrt{s}$	0.0093	0.011	15	−0.00078	0.014	6.4
11	$\langle O_2(c_{WWW})\rangle$ vs $\sqrt{s}$	**0.00719**	0.00020	29000	**−0.00065**	0.00028	8500
12	$\langle O_1(c_W)\rangle$ vs $\sqrt{s}$	**0.00770**	0.00099	1300	−0.00068	0.0013	390
13	$\langle O_2(c_W)\rangle$ vs $\sqrt{s}$	**0.00764**	0.00068	2800	−0.00067	0.00092	820
14	$\langle O_1(c_B)\rangle$ vs $\sqrt{s}$	**0.0097**	0.020	6.5	−0.00079	0.023	3.5
15	$\langle O_2(c_B)\rangle$ vs $\sqrt{s}$	0.0096	0.017	8.8	−0.00082	0.020	4.5
Mean optimal observables divided by powers of invariant mass vs. invariant mass							
16	$\langle O_1(c_{WWW})/s\rangle$ vs $\sqrt{s}$	0.0093	0.011	15	−0.00078	0.014	6.4
17	$\langle O_2(c_{WWW})/s^2\rangle$ vs $\sqrt{s}$	**0.00719**	0.00020	29000	**−0.00065**	0.00028	8500
18	$\langle O_1(c_W)/s\rangle$ vs $\sqrt{s}$	**0.00770**	0.00099	1300	−0.00068	0.0013	390
19	$\langle O_2(c_W)/s^2\rangle$ vs $\sqrt{s}$	**0.00764**	0.00068	2800	−0.00067	0.00092	820
20	$\langle O_1(c_B)/s\rangle$ vs $\sqrt{s}$	**0.0097**	0.020	6.5	−0.00079	0.023	3.5
21	$\langle O_2(c_B)/s^2\rangle$ vs $\sqrt{s}$	0.0096	0.017	8.8	−0.00082	0.020	4.5

Table 7.2: Binned fit results for c_W

Fit results for binned fitting of reweighted EFT to SM for c_W are listed, both for fitting only c_W and for fitting all parameters simultaneously. Values in parenthesis for 95% confidence-limits, are the asymmetric confidence-limits acquired by enabling MINOS errors for the fit. Fit-values in bold are larger than the 95% confidence limit.

Observable		Fit only c_W			Fit All		
		Value [10^{-6} GeV^2]	**95% CL [10^{-6} GeV^2]**	$\frac{\chi^2}{ndf}$	**Value [10^{-6} GeV^2]**	**95% CL [10^{-6} GeV^2]**	$\frac{\chi^2}{ndf}$
Phase-space observables							
1	$P_T(Z)$	−0.11	0.25 (−0.25,+0.26)	0.26	−0.16	0.39 (−0.36,+0.64)	0.26
2	M(WZ) = $\sqrt{s}$	**−2.00**	0.32 (−0.32,+0.33)	0.42	**−2.00**	1.9 (−0.66,+2.2)	0.42
3	y(Z)	−0.37	0.73 (−0.7,+0.77)	0.6	−2.20	4.1 (−1.6,+9.4)	0.61
Optimal observables							
4	$O_1(c_{WWW})$	−0.11	0.21	0.35	0.0061	0.81 (−0.49,+1)	0.35
5	$O_2(c_{WWW})$	−0.095	0.22	0.023	−0.044	0.44 (−0.36,+0.77)	0.023
6	$O_1(c_W)$	−0.086	0.20	0.082	0.11	0.95 (−0.5,+0.83)	0.082
7	$O_2(c_W)$	−0.089	0.22 (−0.21,+0.22)	0.024	−0.00026	0.59 (−0.38,+0.78)	0.024
8	$O_1(c_B)$	−0.091	0.21	0.39	−0.021	0.62 (−0.44,+1)	0.39
9	$O_2(c_B)$	−0.086	0.21	0.059	−0.022	0.67 (−0.39,+0.79)	0.059
Mean optimal observables vs. invariant mass							
10	$\langle O_1(c_{WWW})\rangle$ vs $\sqrt{s}$	**0.798**	0.022	6.4	**0.795**	0.023	6.4
11	$\langle O_2(c_{WWW})\rangle$ vs $\sqrt{s}$	**0.72301**	0.00044	8500	**0.71985**	0.00045	8500
12	$\langle O_1(c_W)\rangle$ vs $\sqrt{s}$	**0.7478**	0.0021	390	**0.7444**	0.0022	390
13	$\langle O_2(c_W)\rangle$ vs $\sqrt{s}$	**0.7439**	0.0015	820	**0.7405**	0.0015	820
14	$\langle O_1(c_B)\rangle$ vs $\sqrt{s}$	**0.791**	0.037	3.5	**0.790**	0.037	3.5
15	$\langle O_2(c_B)\rangle$ vs $\sqrt{s}$	**0.796**	0.031	4.5	**0.794**	0.032	4.5
Mean optimal observables divided by powers of invariant mass vs. invariant mass							
16	$\langle O_1(c_{WWW})/s\rangle$ vs $\sqrt{s}$	**0.798**	0.022	6.4	**0.794**	0.023	6.4
17	$\langle O_2(c_{WWW})/s^2\rangle$ vs $\sqrt{s}$	**0.72300**	0.00044	8500	**0.71985**	0.00045	8500
18	$\langle O_1(c_W)/s\rangle$ vs $\sqrt{s}$	**0.7478**	0.0021	390	**0.7443**	0.0022	390
19	$\langle O_2(c_W)/s^2\rangle$ vs $\sqrt{s}$	**0.7439**	0.0015	820	**0.7405**	0.0015	820
20	$\langle O_1(c_B)/s\rangle$ vs $\sqrt{s}$	**0.790**	0.037	3.5	**0.790**	0.037	3.5
21	$\langle O_2(c_B)/s^2\rangle$ vs $\sqrt{s}$	**0.796**	0.031	4.5	**0.794**	0.032	4.5

Table 7.3: Binned fit results for c_B

Fit results for binned fitting of reweighted EFT to SM for c_B are listed, both for fitting only c_B and for fitting all parameters simultaneously. Values in parenthesis for 95% confidence-limits, are the asymmetric confidence-limits acquired by enabling MINOS errors for the fit. Fit-values in bold are larger than the 95% confidence limit.

Observable		Fit only c_W			Fit All		
		Value [10^{-6} GeV^2]	**95% CL [10^{-6} GeV^2]**	$\frac{\chi^2}{ndf}$	**Value [10^{-6} GeV^2]**	**95% CL [10^{-6} GeV^2]**	$\frac{\chi^2}{ndf}$
Phase-space observables							
1	$P_T(Z)$	2.3	10 (−11,+10)	0.26	−2.6	20 (−17,+25)	0.26
2	$M(WZ) = \sqrt{s}$	**52**	19 (−21,+17)	0.4	**−56**	52 (−38,+63)	0.42
3	$y(Z)$	10	21 (−23,+20)	0.6	−68	220 (−180,+200)	0.61
Optimal observables							
4	$O_1(c_{WWW})$	3.7	7.5 (−7.6,+7.4)	0.35	7.2	35 (−22,+33)	0.35
5	$O_2(c_{WWW})$	4.1	9.3 (−9.5,+9.1)	0.023	3.6	26 (−18,+29)	0.023
6	$O_1(c_W)$	3.1	7.6 (−7.8,+7.5)	0.082	0.95	18 (−14,+21)	0.082
7	$O_2(c_W)$	3.9	8.7 (−8.9,+8.5)	0.024	5.7	29 (−18,+28)	0.024
8	$O_1(c_B)$	3.3	7.5 (−7.6,+7.4)	0.39	4.7	33 (−19,+33)	0.39
9	$O_2(c_B)$	2.9	8.6 (−8.8,+8.5)	0.059	4.6	30 (−20,+28)	0.059
Mean optimal observables vs. invariant mass							
10	$\langle O_1(c_{WWW})\rangle$ vs $\sqrt{s}$	**−11.2**	1.3	15	−1.3	1.6	6.4
11	$\langle O_2(c_{WWW})\rangle$ vs $\sqrt{s}$	**−9.9604**	0.025	29000	**−1.561**	0.034	8500
12	$\langle O_1(c_W)\rangle$ vs $\sqrt{s}$	**−10.35**	0.12	1300	**−1.65**	0.16	390
13	$\langle O_2(c_W)\rangle$ vs $\sqrt{s}$	**−10.286**	0.083	2700	**−1.63**	0.11	820
14	$\langle O_1(c_B)\rangle$ vs $\sqrt{s}$	**−10.9**	2.2	6.3	−0.26	2.6	3.5
15	$\langle O_2(c_B)\rangle$ vs $\sqrt{s}$	**−11.0**	1.8	8.6	−0.71	2.2	4.5
Mean optimal observables divided by powers of invariant mass vs. invariant mass							
16	$\langle O_1(c_{WWW})/s\rangle$ vs $\sqrt{s}$	**−11.2**	1.3	15	−1.3	1.6	6.4
17	$\langle O_2(c_{WWW})/s^2\rangle$ vs $\sqrt{s}$	**−9.9603**	0.025	28000	**−1.561**	0.034	8500
18	$\langle O_1(c_W)/s\rangle$ vs $\sqrt{s}$	**−10.35**	0.12	1300	**−1.65**	0.16	390
19	$\langle O_2(c_W)/s^2\rangle$ vs $\sqrt{s}$	**−10.286**	0.083	2700	**−1.63**	0.11	820
20	$\langle O_1(c_B)/s\rangle$ vs $\sqrt{s}$	**−10.9**	2.2	6.3	−0.27	2.6	3.5
21	$\langle O_2(c_B)/s^2\rangle$ vs $\sqrt{s}$	**−11.0**	1.8	8.6	−0.71	2.2	4.5

7.2 Unbinned fit results

The unbinned fit results are shown in Table 7.4, Table 7.5 and Table 7.6 for each of the EFT model parameters. Here it is rather unclear if any observable is better than another at producing low confidence-limits. As in the previous section for unbinned fits, he confidence-limits for fitting all parameters simultaneously are always higher than for fitting a single parameter at a time.

The execution time of the unbinned fits is significantly slower than that for binned fits. Four binned fits were performed for each observable, one for each model parameter, and one for all. For the 9 observables, that is 36 unbinned fits. All unbinned fits together took about 15 minutes to complete on a fast Mac Book Pro.

Here, many more of the fit-values for unbinned fits are larger than their 95% confidence limits. As unbinned fits take significantly longer to execute, scans of the objective function were not performed, so it is difficult to know currently if these were local minima. Strangely, the number ±10 is a unbinned fit favorite. We suspect that this might be further indication of a local minimum. This needs to be examined further.

All-in-all, the unbinned fits performed the same or worse than the binned fits, on the same observables. When execution time is factored in, the binned fits clearly have no advantage, in our specific use-case.

Table 7.4: Unbinned fit results for c_{WWW}

Fit results for unbinned fitting of reweighted EFT to SM for c_{WWW} are listed, both for fitting only c_{WWW} and for fitting all parameters simultaneously. Fit-values in bold are larger than the 95% confidence-limit.

Observable		Fit only c_{WWW}		Fit All	
		Value [10^{-6} GeV^2]	95% CL [10^{-6} GeV^2]	Value [10^{-6} GeV^2]	95% CL [10^{-6} GeV^2]
Phase-space observables					
1	P_T(Z)	−0.69	0.30	−0.043	0.50
2	M(WZ) = $\sqrt{s}$	**−5.49**	0.14	−0.51	0.65
3	y(Z)	**−10.0000**	0.0085	**−10.00**	0.15
Optimal observables					
4	$O_1(c_{WWW})$	**6.23**	0.16	**−4.89**	0.37
5	$O_2(c_{WWW})$	−0.0057	0.52	−0.0026	0.74
6	$O_1(c_W)$	**0.84**	0.33	**1.23**	0.32
7	$O_2(c_W)$	−0.0026	0.50	−0.0012	0.74
8	$O_1(c_B)$	**6.05**	0.16	**−4.05**	0.41
9	$O_2(c_B)$	0.33	0.52	0.54	0.75

Table 7.5: Unbinned fit results for c_W

Fit results for unbinned fitting of reweighted EFT to SM for c_W are listed, both for fitting only c_W and for fitting all parameters simultaneously. Fit-values in bold are larger than the 95% confidence limit.

Observable		Fit only c_W		Fit All	
		Value [10^{-6} GeV^2]	95% CL [10^{-6} GeV^2]	Value [10^{-6} GeV^2]	95% CL [10^{-6} GeV^2]
Phase-space observables					
1	P_T(Z)	**−0.52**	0.22	**−0.64**	0.30
2	M(WZ) = $\sqrt{s}$	**−7.91**	0.22	**−3.28**	0.60
3	y(Z)	**−10.000**	0.013	**−10.00**	0.18
Optimal observables					
4	$O_1(c_{WWW})$	**10.0000**	0.0097	**0.87**	0.67
5	$O_2(c_{WWW})$	0.041	0.22	0.12	0.39
6	$O_1(c_W)$	0.012	0.22	**0.97**	0.83
7	$O_2(c_W)$	0.052	0.22	0.14	0.39
8	$O_1(c_B)$	**10.000**	0.011	**0.77**	0.65
9	$O_2(c_B)$	0.13	0.23	0.27	0.71

Table 7.6: Unbinned fit results for c_B

Fit results for unbinned fitting of reweighted EFT to SM for c_B are listed, both for fitting only c_B and for fitting all parameters simultaneously. Fit-values in bold are larger than the 95% confidence limit.

Observable		Fit only c_B		Fit All	
		Value [10^{-6} GeV^2]	**95% CL [10^{-6} GeV^2]**	**Value [10^{-6} GeV^2]**	**95% CL [10^{-6} GeV^2]**
Phase-space observables					
1	$P_T(Z)$	**14**	10	−8.9	16
2	M(WZ) = $\sqrt{s}$	**424.0**	9.5	**339**	20
3	y(Z)	**−1000.0**	1.5	**−1000.0**	3.4
Optimal observables					
4	$O_1(c_{WWW})$	**−474.3**	8.9	**−424**	18
5	$O_2(c_{WWW})$	0.21	9.2	4	18
6	$O_1(c_W)$	−3.7	8.6	**32**	25
7	$O_2(c_W)$	0.12	9.2	4.7	18
8	$O_1(c_B)$	**−466.4**	8.9	**−444**	15
9	$O_2(c_B)$	−5.6	9.6	6.7	28

7.3 Notes on using ROOT to fit data

In general, we have used as much as possible ROOT’s default fit configuration. It was necessary to increase the tolerance setting from its default of 0.1 to 25 for unbinned fits, in order for some of the unbinned fits to converge, possibly due to the increased scale of the log-likelihood measurements from summing 1M likelihoods as opposed to 100 to 10000.

8 Conclusion

This project has primarily been a programming project, wherein a long-term software tool called *SherpaWeight* was developed for automating the reweighting task of calculating reweight-coefficients and embedding them with their events. SherpaWeight is simple and easy to use, particularly to those familiar with Sherpa run cards. The tool currently supports HepMC, gzipped HepMC and Sherpa-ROOT event files, and more an be easily added.

Additional libraries of methods and classes were developed beyond SherpaWeight to perform the actual reweighting with the reweight-coefficients, and to be able to reweight a model-dataset while fitting it to a pseudo-dataset. This library of routines was tested thoroughly.

We used an effective field theory model of dimension-six operators with three model parameters to test the functionality of SherpaWeight and the reweighting library. We developed this model with FeynRules, and did extensive testing to validate it was properly implemented.

We examined optimal observables and showed that they can acquired easily from the reweight-coefficients for the events. So not only can one use SherpaWeight to enable reweighting, it can also be used to calculate optimal observables for a dataset.

We performed a large fit analysis of 21 different observables, including 3 different sets of optimal observables, and determined that mean optimal observables do indeed give the lowest statistical error compared to the other observables we tested. We also found that it was unnecessary to divide the optimal observable for an event by the event's invariant mass in order to "factor-out" any energy dependence the optimal observable may have – the fit results were unchanged.

We performed both binned and unbinned fits, and determined that for our datasets and observables, that unbinned fits did not give us any noticeable advantage, and took nearly 100 times longer to perform.

It was very clear when performing binned fits, that the time taken to reweight one's model data iteratively for a fit is many orders of magnitude less than the time to generate event data and run it through a detector simulator.

This project has clearly shown the advantages and ease of using reweighting.

8.1 Further work

More work can yet be done.

One could perform a 2D-fit of the first and second-order optimal observables to ascertain if this would improve confidence limits further.

Fitting performance can be further tested with pseudo-data that has been run through a detector simulator, to ascertain what consequences it has on confidence levels.

9 References

[1] Almalioglu Yasin, Salzburger Andreas, and Ritsch Elmar, "Performance Improvements for the ATLAS Detector Simulation Framework," (2013) 10.5281/zenodo.7558.

[2] “Sherpa”, 2014-2015. https://sherpa.hepforge.org/trac/wiki.

[3] “Homebrew – The missing package manager for OS X”, 2015. http://brew.sh/.

[4] David C. Hall, “homebrew-hep”, 2014-2015. http://davidchall.github.io/homebrew-hep/.

[5] “FeynRules”, 2014-2015. http://feynrules.irmp.ucl.ac.be/.

[6] Adam Alloul, Neil D. Christensen, Céline Degrande, Claude Duhr, and Benjamin Fuks, "FeynRules 2.0 - A complete toolbox for tree-level phenomenology," Comput.Phys.Commun. **185**, 2250-2300 (2014).

[7] Neil D. Christensen and Claude Duhr, "FeynRules - Feynman rules made easy," Comput.Phys.Commun. **180**, 1614-1641 (2009).

[8] Celine Degrande, Claude Duhr, Benjamin Fuks, David Grellscheid, Olivier Mattelaer, and Thomas Reiter, "UFO - The Universal FeynRules Output," Comput.Phys.Commun. **183**, 1201-1214 (2012).

[9] “HepMC Event Record”, 2015. http://lcgapp.cern.ch/project/simu/HepMC/.

[10] “ROOT a data analysis framework”, 2014-2015. https://root.cern.ch/.

[11] Celine Degrande, Nicolas Greiner, Wolfgang Kilian, Olivier Mattelaer, Harrison Mebane, Tim Stelzer, Scott Willenbrock, and Cen Zhang, "Effective Field Theory: A Modern Approach to Anomalous Couplings," Annals Phys. **335**, 21-32 (2013).

[12] K. Hagiwara, S. Ishihara, R. Szalapski, and D. Zeppenfeld, "Low energy effects of new interactions in the electroweak boson sector," Physical Review D **48** (5), 2182-2203 (1993).

[13] Markus Diehl, Otto Nachtmann, and Felix Nagel, "Triple gauge couplings in polarized e- e+ ---> W- W+ and their measurement using optimal observables," Eur.Phys.J. **C27**, 375-397 (2003).

[14] Otto Nachtmann and Felix Nagel, "Optimal observables and phase-space ambiguities," Eur.Phys.J. **C40**, 497-503 (2005).

[15] M. Dobbs and L. Lefebvre, "Prospects for Probing the Three Gauge-boson Couplings in W + Photon Production at the LHC," (2002).

[16] M. Diehl and O. Nachtmann, "Optimal observables for the measurement of three gauge boson couplings in e+ e- ---> W+ W-," Z.Phys. **C62**, 397-412 (1994).

[17] M. Diehl and O. Nachtmann, "Anomalous three gauge couplings in e+ e- ---> W+ W- and 'optimal' strategies for their measurement," Eur.Phys.J. **C1**, 177-190 (1998).

10 Source code

Source code for all the applications and libraries developed for this project are publicly available for download from *GitHub* at https://github.com/christopher-jacobsen. Relevant repositories are prefixed with Physics-Thesis.