Automated matrix-element re-weighting in effective field theories

Subtitle

# Abstract

TOC

# Introduction

## Reweighting

## Applications

### Sherpa

### FeynRules

# SherpaWeight

## Developing SherpaWeight

## Using SherpaME

## Using SherpaWeight

# 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 [1] and [2].

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 can be defined to help characterize the coefficients. The following is the EFT extension to the SM for dimension-six and dimension-eight operators:

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

Where and are dimensionless coefficients to dimension-six operators and dimension-eight operators respectively.

EFT assumes that 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 .

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 , essentially just introduces an unnecessary complication, and can be absorbed into the coefficients themselves. This gives us:

|  |  |  |
| --- | --- | --- |
|  |  | (3.2) |

Where are the coefficients to the dimension-six operators . 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:

|  |  |  |
| --- | --- | --- |
|  |  | (3.3) |
|  |  | (3.4) |
|  |  | (3.5) |

Where is the Higgs doublet field and

|  |  |  |
| --- | --- | --- |
|  |  | (3.6) |
|  |  | (3.7) |
|  |  | (3.8) |

Where is the covariant derivative. and are respectively the field-strengths tensors of the and gauge-bosons. and are respectively the weak and U(1) Y coupling constants at the Z pole. are the Pauli-sigma matrixes, and is the Levi-Civita symbol.

and are defined by requiring:

|  |  |  |
| --- | --- | --- |
|  |  | (3.9) |

Note that is a 2×2 matrix, thus the use of trace in (3.3).

The coefficients corresponding to the three operators , and are defined as , , and respectively.

## 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 [3]. 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. Details on FeynRules can be found in [4] [5] [6].

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 and 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 and 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:

|  |  |  |
| --- | --- | --- |
|  |  | (3.10) |

This results in a change of sign in the derivation of and 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 indexes of the three 2×2 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).

### 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 | Model independent files containing various UFO model methods. |
| function\_library.py |
| 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 |  |
| 4 | , |
| 5 | ,  , |
| 6 |  |

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.

### 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.

## 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:

|  |  |  |
| --- | --- | --- |
|  |  | (3.11) |

Where 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:

|  |  |  |
| --- | --- | --- |
|  |  | (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.

## Validating EFT model using WZ production

# Reweighting

## Comparison of reweighted EFT to SM for WZ-production

## Notes on reweighting using ROOT

### Scaling luminosity

### Reweighting histograms

# Optimal Observables

# Chosen observables for parton-level WZ-production

In the following sections 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 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, PT(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 cWWW, cW, and cB 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 | PT(Z) |  | GeV/c | 54.6 | 42.5 | 41.7 | 0 to 750 | 0 / 8.2 |
| 2 | M(WZ) = |  | GeV/c2 | 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 |  |  | GeV2 | −1340 | 5660 | 3210 | (−6 to 1) ×104 | 6.7 / 0.2 |
| 5 |  |  | GeV4 | 8.13×108 | 4.90×1010 | 8.35×107 | (0 to 14) ×1011 | 0 / 9.5 |
| 6 |  |  | GeV2 | −5540 | 26700 | 7280 | (−12 to 1) ×105 | 7.6 / 0 |
| 7 |  |  | GeV4 | 3.93×108 | 1.86×1010 | 8.08×107 | (0 to 53) ×1010 | 0 / 9.8 |
| 8 |  |  | GeV2 | 184 | 574 | 418 | (−1 to 5) ×103 | 0.2 / 8.2 |
| 9 |  |  | GeV4 | 6.78×105 | 2.76×106 | 4.00×105 | (0 to 13) ×107 | 0 / 9.1 |
| **Mean optimal observables vs. invariant mass** | | | | | | | | |
| 10 | vs | GeV2 | GeV | Same as M(WZ) | | | | |
| 11 | vs | GeV4 | GeV |
| 12 | vs | GeV2 | GeV |
| 13 | vs | GeV4 | GeV |
| 14 | vs | GeV2 | GeV |
| 15 | vs | GeV4 | GeV |
| **Mean optimal observables divided by powers of invariant mass vs. invariant mass** | | | | | | | | |
| 16 | vs |  | GeV | Same as M(WZ) | | | | |
| 17 | vs |  | GeV |
| 18 | vs |  | GeV |
| 19 | vs |  | GeV |
| 20 | vs |  | GeV |
| 21 | vs |  | GeV |

1. For all events including those outside the selected range.
2. SM data sample at 10 fb–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 χ2, 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 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.

## 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 χ2, 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.

### 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:

|  |  |  |
| --- | --- | --- |
|  |  | (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:

|  |  |  |
| --- | --- | --- |
|  |  | (6.2) |

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

|  |  |  |
| --- | --- | --- |
|  |  | (6.3) |

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

|  |  |  |
| --- | --- | --- |
|  |  | (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 [REF].

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:

|  |  |  |
| --- | --- | --- |
|  |  | (6.5) |

Where is the mean events per bin, is the variance in the events per bin, are the number of events in bin , and is the total number of bins.

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

|  |  |  |
| --- | --- | --- |
|  |  | (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 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.

#### 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. See Figure [REF] for an example minimization curve.

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.

### 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 | PT(Z) | 952 | 1700 | 900 | 750 |
| 2 | M(WZ) = | 709 | 1510 | 1750 | 1500 |
| 3 | y(Z) | 139 | 153 | 175 | 200 |
| **Optimal observables** | | | | | |
| 4 |  | 394 | 1220 | 2000 | 2000 |
| 5 |  | 16300 | 1.68×107 | > 1E6 | 10000 |
| 6 |  | 6110 | 39200 | 17500 | 10000 |
| 7 |  | 16200 | 6.53×106 | > 1E6 | 10000 |
| 8 |  | 338 | 812 | 1250 | 1000 |
| 9 |  | 6190 | 74900 | 325000 | 10000 |
| **Mean optimal observables vs. invariant mass** | | | | | |
| 10 | vs | Same as M(WZ) | | | |
| 11 | vs |
| 12 | vs |
| 13 | vs |
| 14 | vs |
| 15 | vs |
| **Mean optimal observables divided by powers of invariant mass vs. invariant mass** | | | | | |
| 16 | vs | Same as M(WZ) | | | |
| 17 | vs |
| 18 | vs |
| 19 | vs |
| 20 | vs |
| 21 | vs |

## 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.

# 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 on the parameters should be zero for all 3 parameters cWWW, cW, and cB.

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 4.2.1. 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 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 , as there is no error for an empty bin to calculate the denominator of the value.

## 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.

Table 7.1: Binned fit results for cWWW

| **Observable** | | **Fit only cWWW** | | | **Fit All** | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Value**  **[10−6 GeV2]** | **95% CL**  **[10−6 GeV2]** |  | **Value**  **[10−6 GeV2]** | **95% CL**  **[10−6 GeV2]** |  |
| **Phase-space observables** | | | | | | | |
| 1 | PT(Z) | −0.26 | 0.64  (−0.43,+0.96) | 0.26 | 0.051 | 1.2  (−0.85,+0.77) | 0.26 |
| 2 | M(WZ) = | −1.80 | 0.24  (−0.23,+0.24) | 0.41 | −1.0 | 2.0  (−0.93,+2.9) | 0.41 |
| 3 | y(Z) | −0.98 | 1.5  (−1.1,+4.6) | 0.6 | 0.45 | 5.7  (−4,+3.4) | 0.6 |
| **Optimal observables** | | | | | | | |
| 4 |  | −0.31 | 0.55  (−0.44,+0.9) | 0.36 | 0.43 | 1.6  (−1.7,+0.91) | 0.36 |
| 5 |  | −0.27 | 0.54  (−0.4,+0.88) | 0.023 | 0.21 | 1.6  (−1.1,+0.75) | 0.023 |
| 6 |  | −0.30 | 0.50  (−0.39,+0.91) | 0.081 | −0.43 | 1.5  (−0.65,+1.5) | 0.081 |
| 7 |  | −0.26 | 0.55  (−0.41,+0.85) | 0.024 | 0.13 | 1.5  (−1,+0.8) | 0.024 |
| 8 |  | −0.28 | 0.56  (−0.45,+0.88) | 0.39 | 0.30 | 2.0  (−1.6,+1) | 0.39 |
| 9 |  | −0.27 | 0.56  (−0.41,+0.96) | 0.062 | 0.38 | 1.2  (−1.4,+0.65) | 0.062 |
| **Mean optimal observables vs. invariant mass** | | | | | | | |
| 10 | vs | 0.011 | 0.018 | 7.5 | −0.00070 | 0.023 | 3.1 |
| 11 | vs | 0.0100 | 0.0015 | 780 | −0.00080 | 0.002 | 210 |
| 12 | vs | 0.0097 | 0.0029 | 210 | −0.00077 | 0.0039 | 60 |
| 13 | vs | 0.0098 | 0.0024 | 310 | −0.00078 | 0.0032 | 87 |
| 14 | vs | 0.012 | 0.032 | 3.2 | −0.00045 | 0.038 | 1.7 |
| 15 | vs | 0.012 | 0.029 | 3.8 | −0.00054 | 0.034 | 2 |
| **Mean optimal observables divided by powers of invariant mass vs. invariant mass** | | | | | | | |
| 16 | vs | 0.011 | 0.018 | 7.5 | −0.00070 | 0.023 | 3.1 |
| 17 | vs | 0.0100 | 0.0015 | 780 | −0.00080 | 0.002 | 210 |
| 18 | vs | 0.0097 | 0.0029 | 210 | −0.00077 | 0.0038 | 60 |
| 19 | vs | 0.0098 | 0.0024 | 310 | −0.00078 | 0.0032 | 87 |
| 20 | vs | 0.012 | 0.032 | 3.2 | −0.00045 | 0.038 | 1.7 |
| 21 | vs | 0.012 | 0.029 | 3.8 | −0.00054 | 0.034 | 2 |

Table 7.2: Binned fit results for cW

| **Observable** | | **Fit only cW** | | | **Fit All** | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Value**  **[10−6 GeV2]** | **95% CL**  **[10−6 GeV2]** |  | **Value**  **[10−6 GeV2]** | **95% CL**  **[10−6 GeV2]** |  |
| **Phase-space observables** | | | | | | | |
| 1 | PT(Z) | −0.11 | 0.26 (−0.25,+0.26) | 0.26 | −0.16 | 0.40 (−0.37,+0.68) | 0.26 |
| 2 | M(WZ) = | −2.00 | 0.33 (−0.32,+0.34) | 0.41 | −1.9 | 1.9 (−0.78,+2.1) | 0.41 |
| 3 | y(Z) | −0.41 | 0.74 (−0.71,+0.78) | 0.60 | −2.0 | 4.6  (−1.8,+9.3) | 0.60 |
| **Optimal observables** | | | | | | | |
| 4 |  | −0.11 | 0.22 | 0.36 | 0.011 | 0.83 (−0.5,+1.1) | 0.36 |
| 5 |  | −0.094 | 0.22 (−0.22,+0.23) | 0.023 | −0.011 | 0.60 (−0.39,+0.84) | 0.023 |
| 6 |  | −0.088 | 0.20 | 0.081 | 0.084 | 1.0 (−0.48,+0.93) | 0.081 |
| 7 |  | −0.093 | 0.22 | 0.024 | −0.044 | 0.44 (−0.36,+0.74) | 0.024 |
| 8 |  | −0.095 | 0.22 | 0.39 | −0.011 | 0.70 (−0.46,+1) | 0.39 |
| 9 |  | −0.083 | 0.22 (−0.21,+0.22) | 0.062 | 0.015 | 0.71 (−0.43,+0.82) | 0.062 |
| **Mean optimal observables vs. invariant mass** | | | | | | | |
| 10 | vs | 0.870 | 0.035 | 3.1 | 0.870 | 0.036 | 3.1 |
| 11 | vs | 0.8600 | 0.0032 | 210 | 0.8500 | 0.0033 | 210 |
| 12 | vs | 0.8400 | 0.0061 | 60 | 0.8400 | 0.0062 | 60 |
| 13 | vs | 0.8500 | 0.0051 | 87 | 0.8400 | 0.0051 | 87 |
| 14 | vs | 0.850 | 0.059 | 1.7 | 0.860 | 0.060 | 1.7 |
| 15 | vs | 0.860 | 0.053 | 2.0 | 0.870 | 0.054 | 2.0 |
| **Mean optimal observables divided by powers of invariant mass vs. invariant mass** | | | | | | | |
| 16 | vs | 0.870 | 0.035 | 3.1 | 0.870 | 0.036 | 3.1 |
| 17 | vs | 0.8600 | 0.0032 | 210 | 0.8500 | 0.0033 | 210 |
| 18 | vs | 0.8400 | 0.0061 | 60 | 0.8400 | 0.0062 | 60 |
| 19 | vs | 0.8500 | 0.0051 | 87 | 0.8400 | 0.0051 | 87 |
| 20 | vs | 0.850 | 0.059 | 1.7 | 0.860 | 0.060 | 1.7 |
| 21 | vs | 0.860 | 0.053 | 2.0 | 0.870 | 0.054 | 2.0 |

Table 7.3: Binned fit results for cB

| **Observable** | | **Fit only cW** | | | **Fit All** | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Value**  **[10−6 GeV2]** | **95% CL**  **[10−6 GeV2]** |  | **Value**  **[10−6 GeV2]** | **95% CL**  **[10−6 GeV2]** |  |
| **Phase-space observables** | | | | | | | |
| 1 | PT(Z) | 2.5 | 11  (−11,+10) | 0.26 | −2.3 | 20  (−17,+26) | 0.26 |
| 2 | M(WZ) = | 54 | 19  (−21,+17) | 0.39 | −51 | 55  (−41,+66) | 0.41 |
| 3 | y(Z) | 12 | 22  (−24,+20) | 0.6 | −61 | 230  (−190,+200) | 0.6 |
| **Optimal observables** | | | | | | | |
| 4 |  | 3.8 | 7.7  (−7.8,+7.6) | 0.36 | 7.5 | 35  (−22,+33) | 0.36 |
| 5 |  | 4.2 | 9.5  (−9.7,+9.2) | 0.023 | 5.6 | 31  (−20,+31) | 0.023 |
| 6 |  | 3.2 | 7.8  (−7.9,+7.7) | 0.081 | 0.91 | 17  (−14,+21) | 0.081 |
| 7 |  | 3.8 | 8.9  (−9.1,+8.7) | 0.024 | 3.4 | 25  (−17,+27) | 0.024 |
| 8 |  | 3.4 | 7.7  (−7.8,+7.6) | 0.39 | 5.6 | 34  (−20,+34) | 0.39 |
| 9 |  | 2.9 | 8.8  (−9,+8.6) | 0.062 | 6.1 | 30  (−21,+28) | 0.062 |
| **Mean optimal observables vs. invariant mass** | | | | | | | |
| 10 | vs | −12.0 | 2.0 | 7.2 | −0.45 | 2.5 | 3.1 |
| 11 | vs | −12.00 | 0.17 | 750 | −1.80 | 0.24 | 210 |
| 12 | vs | −12.00 | 0.33 | 200 | −1.80 | 0.45 | 60 |
| 13 | vs | −12.00 | 0.27 | 300 | −1.80 | 0.37 | 87 |
| 14 | vs | −11.0 | 3.2 | 3.1 | 2.1 | 3.8 | 1.7 |
| 15 | vs | −12.0 | 2.9 | 3.7 | 1.4 | 3.5 | 2 |
| **Mean optimal observables divided by powers of invariant mass vs. invariant mass** | | | | | | | |
| 16 | vs | −12.0 | 2.0 | 7.3 | −0.45 | 2.5 | 3.1 |
| 17 | vs | −12.00 | 0.17 | 750 | −1.80 | 0.24 | 210 |
| 18 | vs | −12.00 | 0.33 | 200 | −1.80 | 0.45 | 60 |
| 19 | vs | −12.00 | 0.27 | 300 | −1.80 | 0.37 | 87 |
| 20 | vs | −11.0 | 3.2 | 3.1 | 2.1 | 3.8 | 1.7 |
| 21 | vs | −12.0 | 2.9 | 3.7 | 1.4 | 3.5 | 2 |

## Unbinned fit results

Table 7.4: Unbinned fit results for cWWW

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Observable** | | **Fit only cWWW** | | **Fit All** | |
| **Value [10−6 GeV2]** | **95% CL [10−6 GeV2]** | **Value [10−6 GeV2]** | **95% CL [10−6 GeV2]** |
| **Phase-space observables** | | | | | |
| 1 | PT(Z) | −0.70 | 0.30 | −0.036 | 0.51 |
| 2 | M(WZ) = | −5.5 | 0.14 | −0.52 | 0.66 |
| 3 | y(Z) | −10 | 0.0089 | 10 | 1.7 |
| **Optimal observables** | | | | | |
| 4 |  | 6.3 | 0.16 | −5.0 | 0.37 |
| 5 |  | −0.0014 | 0.54 | −0.0017 | 0.78 |
| 6 |  | 0.77 | 0.36 | 1.1 | 0.37 |
| 7 |  | 0.0014 | 0.52 | −0.0012 | 0.76 |
| 8 |  | 6.1 | 0.16 | −4.0 | 0.42 |
| 9 |  | 0.34 | 0.53 | 0.55 | 1.1 |

Table 7.5: Unbinned fit results for cW

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Observable** | | **Fit only cW** | | **Fit All** | |
| **Value [10−6 GeV2]** | **95% CL [10−6 GeV2]** | **Value [10−6 GeV2]** | **95% CL [10−6 GeV2]** |
| **Phase-space observables** | | | | | |
| 1 | PT(Z) | −0.53 | 0.22 | −0.67 | 0.31 |
| 2 | M(WZ) = | −7.9 | 0.22 | −3.3 | 0.64 |
| 3 | y(Z) | −10 | 0.014 | −10 | 0.45 |
| **Optimal observables** | | | | | |
| 4 |  | 10 | 0.0098 | 1.0 | 0.67 |
| 5 |  | 0.043 | 0.23 | 0.12 | 0.39 |
| 6 |  | 0.035 | 0.22 | 0.81 | 0.85 |
| 7 |  | 0.055 | 0.23 | 0.14 | 0.40 |
| 8 |  | 10 | 0.011 | 0.85 | 0.67 |
| 9 |  | 0.13 | 0.23 | 0.28 | 1.1 |

Table 7.6: Unbinned fit results for cB

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Observable** | | **Fit only cB** | | **Fit All** | |
| **Value [10−6 GeV2]** | **95% CL [10−6 GeV2]** | **Value [10−6 GeV2]** | **95% CL [10−6 GeV2]** |
| **Phase-space observables** | | | | | |
| 1 | PT(Z) | 14 | 10 | −10 | 16 |
| 2 | M(WZ) = | 430 | 9.7 | 340 | 21 |
| 3 | y(Z) | −1000 | 1.6 | −870 | 34 |
| **Optimal observables** | | | | | |
| 4 |  | 81 | 7.6 | −420 | 20 |
| 5 |  | 0.092 | 9.4 | 3.9 | 18 |
| 6 |  | −3.9 | 8.8 | 27 | 27 |
| 7 |  | −0.055 | 9.3 | 4.5 | 18 |
| 8 |  | 65 | 8.8 | −450 | 15 |
| 9 |  | 81 | 7.6 | −420 | 20 |

## Analysis of fit results

TODO:

* Binned mean optimal observables had the best overall resolution, i.e. lowest CL.
* Mean optimal observables and mean optimal observables divided by powers of invariant mass gave the same results, to two significant digits. Neither is better than the other. [Double check this is not a bug].
* Unbinned results are inconsistent. Sometimes CL is low, sometimes high. Often fit value is much greater than CL, meaning that the true value of zero is very unlikely. Could this be due to only fitting shape? The results for binned were worse for 100K bins (those shown) than 10K bins. Shouldn’t higher resolution be better.
* Ran fits as described, as well as with empty-bins merged with non-empty bins. The difference on the fit results was very little, only changing the second significant digit of the CL on a small handful of observables. Decided to leave zero-bin-merging out of the final results.

## 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.

TODO: add more details

Observables

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Observable** | **Type** | **Bin Range** | **Bins** | **Fit Bins a** | **Underflow**  **Count b** | **Overflow**  **Count b** |
| **PT(Z)** | count | 0 to 750 GeV/c | 150 | 150 | 0 | 8.2 |
| **M(WZ) =** | count | 0 to 3000 GeV/c2 | 150 | 142 | 0 | 6.3 |
| **y(Z)** | count | −5 to +5 | 100 | 98 | 0 | 0 |
| **O1(cWWW)** | count | −60000 to 7000 | 1000 | 1000 | 6.7 | 3.4 |
| **O2(cWWW)** | count | 0 to 2×1012 | 1000 | 1000 | 0 | 5.9 |
| **O1(cW)** | count | (−200 to 2)×104 | 1000 | 997 | 2.7 | 0 |
| **O2(cW)** | count | 0 to 6×1011 | 1000 | 1000 | 0 | 7.6 |
| **O1(cB)** | count | −800 to 5000 | 1000 | 996 | 1.5 | 8.2 |
| **O2(cB)** | count | 0 to 2×108 | 1000 | 1000 | 0 | 3.7 |
| **O1(cWWW) vs** | mean | 0 to 3000 GeV | 100 | 76 | 0 | 6.3 |
| **O2(cWWW) vs** | mean | 0 to 3000 GeV | 100 | 76 | 0 | 6.3 |
| **O1(cW) vs** | mean | 0 to 3000 GeV | 100 | 76 | 0 | 6.3 |
| **O2(cW) vs** | mean | 0 to 3000 GeV | 100 | 76 | 0 | 6.3 |
| **O1(cB) vs** | mean | 0 to 3000 GeV | 100 | 76 | 0 | 6.3 |
| **O2(cB) vs** | mean | 0 to 3000 GeV | 100 | 76 | 0 | 6.3 |

1. Fit skips bins with a zero error value. For mean observables, fit also skips bins with less than 10 effective events in either the SM or EFT sample prior to any luminosity scaling.
2. SM sample at 10 fb−1.

cW

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Observable** | **Fit cW** | | | **Fit All** | | |
| **Value**  **[10−6 GeV2]** | **Error**  **[10−6 GeV2]** |  | **Value**  **[10−6 GeV2]** | **Error**  **[10−6 GeV2]** |  |
| **PT(Z)** | −0.11 | 0.13 | 0.27 | −0.16 | 0.20 | 0.27 |
| **M(WZ)** | −1.1 | 0.24 | 0.46 | −1.42 | 0.37 | 0.47 |
| **y(Z)** | −0.41 | 0.37 | 0.61 | −1.6 | 3.2 | 0.62 |
| **O1(cWWW)** | −0.10 | 0.11 | 0.36 | 0.020 | 0.39 | 0.36 |
| **O2(cWWW)** | −0.10 | 0.12 | 0.059 | 0.18 | 0.50 | 0.057 |
| **O1(cW)** | −0.069 | 0.095 | 0.099 | 0.0081 | 0.24 | 0.099 |
| **O2(cW)** | −0.12 | 0.11 | 0.077 | 0.23 | 0.50 | 0.074 |
| **O1(cB)** | −0.095 | 0.11 | 0.40 | −0.0091 | 0.36 | 0.40 |
| **O2(cB)** | −0.086 | 0.10 | 0.087 | −0.012 | 0.25 | 0.087 |
| **O1(cWWW) vs** | 0.44 | 0.17 | 0.063 | 0.44 | 0.17 | 0.065 |
| **O2(cWWW) vs** | 0.457 | 0.041 | 0.85 | 0.456 | 0.041 | 0.87 |
| **O1(cW) vs** | 0.456 | 0.046 | 0.67 | 0.455 | 0.047 | 0.69 |
| **O2(cW) vs** | 0.456 | 0.045 | 0.72 | 0.455 | 0.045 | 0.74 |
| **O1(cB) vs** | 0.43 | 0.27 | 0.034 | 0.43 | 0.28 | 0.035 |
| **O2(cB) vs** | 0.43 | 0.25 | 0.036 | 0.43 | 0.25 | 0.037 |

cB

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Observable** | **Fit cB** | | | **Fit All** | | |
| **Value**  **[10−6 GeV2]** | **Error**  **[10−6 GeV2]** |  | **Value**  **[10−6 GeV2]** | **Error**  **[10−6 GeV2]** |  |
| **PT(Z)** | 2.4 | 5.3 | 0.26 | −2.3 | 10 | 0.27 |
| **M(WZ)** | 23 | 12 | 0.47 | −27 | 29 | 0.47 |
| **y(Z)** | 12 | 11 | 0.61 | −39 | 140 | 0.62 |
| **O1(cWWW)** | 3.8 | 3.8 | 0.36 | 7.3 | 17 | 0.36 |
| **O2(cWWW)** | 7.0 | 6.4 | 0.058 | 17 | 22 | 0.057 |
| **O1(cW)** | 3.4 | 3.9 | 0.099 | 5.0 | 13 | 0.099 |
| **O2(cW)** | 7.3 | 5.5 | 0.075 | 19 | 20 | 0.074 |
| **O1(cB)** | 3.4 | 3.8 | 0.40 | 5.7 | 17 | 0.40 |
| **O2(cB)** | 4.1 | 4.4 | 0.086 | 5.1 | 13 | 0.087 |
| **O1(cWWW) vs** | −6.2 | 12 | 0.19 | −0.97 | 17 | 0.065 |
| **O2(cWWW) vs** | −6.3 | 2.9 | 3.2 | −0.94 | 4 | 0.87 |
| **O1(cW) vs** | −6.3 | 3.3 | 2.5 | −0.94 | 4.5 | 0.69 |
| **O2(cW) vs** | −6.3 | 3.1 | 2.7 | −0.94 | 4.3 | 0.74 |
| **O1(cB) vs** | −6.2 | 20 | 0.079 | −0.91 | 27 | 0.035 |
| **O2(cB) vs** | −6.2 | 18 | 0.090 | −0.92 | 25 | 0.037 |

# Fitting EFT to SM for simulated measurement of WZ-production

# Conclusion

# Bibliography

[1] 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).

[2] 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).

[3] 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).

[4] “FeynRules”, 2015. <http://feynrules.irmp.ucl.ac.be/>.

[5] 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).

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