ZPrimeCombine Interface to the Higgs Combine Tool for the $Z' \to \ell\ell$ search

User Manual

Jan-Frederik Schulte

Version 2.0.0, 2018/17/07

Contents

1 Introduction

The ZPrimeCombine package, to be found on Gitlab, provides an interface between the experimental results of the $Z' \to \ell\ell$ analysis, as well as the non-resonant interpretation in Contact Interactions, and the Higgs Combine Tool. This tool, referred to simply as "combine" going further, is in turn an interface to the underlying statistical tools provided by RooStats. This document aims to summarize the functionality of the the tool and give instructions how to use it to derive limits and significances for the analysis.

2 Setup

The current implementation in the package is based on version `v7.0.10` of combine. CMSSW_8_1_0 is used to set the environment, but this is only to ensure a consistent version of ROOT, Combine does not rely on CMSSW itself. Combine is installed using the following commands

```
export SCRAM_ARCH=slc6_amd64_gcc530
cmsrel CMSSW_8_1_0
cd CMSSW_8_1_0/src
cmsenv
git clone https://github.com/cms-analysis/HiggsAnalysis-CombinedLimit.git HiggsAnalysis/CombinedLimit
cd HiggsAnalysis/CombinedLimit
cd $CMSSW_BASE/src/HiggsAnalysis/CombinedLimit
git fetch origin
git checkout v7.0.10
scramv1 b clean; scramv1 b # always make a clean build
```

Detailed documentation of combine can be found on this Twiki page.

To finish the setup, just clone the ZPrimeCombine repository from the link given above. Note that this is only possible for CMS members subscribed to the Z' e-group. This documentation refers to version 1.0 of the framework, which can be checked out using `git checkout v1.0`. At the moment of writing this, all developments are merged into the master branch. This might change in the future, so be aware that you might have to check out a different branch to find the functionality you need.

3 Usage

The central entry point to the framework is the script `runInterpretation.py`. It steers both the creation of the inputs given to combine as well as the execution of it, either locally or via batch/grid jobs. Let's have a look at its functionality:

```
Steering tool for Zprime -> ll analysis interpretation in combine

optional arguments:
  -h, --help            show this help message and exit
  -r, --redo            recreate datacards and workspaces for this
```

```
                        configuration
  -w, --write           create datacards and workspaces for this configuration
  -b, --binned          use binned dataset
  -s, --submit          submit jobs to cluster
  --signif              run significance instead of limits
  --LEE                 run significance on BG only toys to estimate LEE
  --frequentist         use frequentist CLs limits
  --hybrid              use frequenstist-bayesian hybrid methods
  --plc                 use PLC for signifcance calculation
  -e, --expected        expected limits
  -i, --inject          inject signal
  --recreateToys        recreate the toy dataset for this configuration
  --crab                submit to crab
  -c CONFIG, --config CONFIG
                        name of the congiguration to use
  -t TAG, --tag TAG     tag to label output
  --workDir WORKDIR     tells batch jobs where to put the datacards. Not for
                        human use!
  -n NTOYSEXP, --nToysExp NTOYSEXP
                        number of expected limits to be calculated in one job,
                        overwrites then expToys option from the config file.
                        Used for CONDOR jobs at LPC so far
  -m MASS, --mass MASS  mass point
  -L LAMBDA, --Lambda LAMBDA
                        Lambda values
  --CI                  calculate CI limits
  --usePhysicsModel     use PhysicsModel to set limtsi of Lamda
  --singlebin           use single bin counting for CI. The mass parameter now
                        designates the lower mass threshold
  --Lower               calculate lower limits
  --spin2               calculate limits for spin2 resonances
  --bias                perform bias study
```

In the following the use of these options for different purposes is described. The most important parameter is `-c`, which tells the framework, which configuration file to use for steering. It is the only argument which is mandatory to give, and the name of the configuration will be used to tag all inputs and results. The configuration files themselves are discussed in the next section. Another universal option is the `-t` option which can used to tag the in- and output of the tool.

3.1 Input creation

The first task of the framework is to create the datacards and workspaces used as inputs for combine. To provide the framework with the necessary information, the user has to provide two different types of inputs. All experimental information is located in the `input/` directory. Here, for each channel of the analysis, there is one `channelConfig_channelName.py` file. For example, for the barrel-barrel category in the dimuon channel for the 2016 result (EXO-16-047), it looks like the example shown below. Important things are commented throughout.

```
import ROOT,sys
ROOT.gROOT.SetBatch(True)
ROOT.gErrorIgnoreLevel = 1
from ROOT import *
from muonResolution import getResolution as getRes #external source for ↩
    parametrization of dimuon resolution
nBkg = -1

dataFile = "input/dimuon_Mordion2017_BB.txt"
def addBkgUncertPrior(ws,label,channel,uncert):

        beta_bkg = RooRealVar('beta_%s_%s'%(label,channel),'beta_%s_%s'%(label,↩
            channel),0,-5,5)
        getattr(ws,'import')(beta_bkg,ROOT.RooCmdArg())
        uncert = 1. + uncert
        bkg_kappa = RooRealVar('%s_%s_kappa'%(label,channel),'%s_%s_kappa'%(label↩
            ,channel),uncert)
        bkg_kappa.setConstant()
        getattr(ws,'import')(bkg_kappa,ROOT.RooCmdArg())
        ws.factory("PowFunc::%s_%s_nuis(%s_%s_kappa, beta_%s_%s)"%(label,channel,↩
            label,channel,label,channel))
        ws.factory("prod::%s_%s_forUse(%s_%s, %s_%s_nuis)"%(label,channel,label,↩
            channel,label,channel))
#functionality to add a uncertainty to a background fit parameter

def provideSignalScaling(mass,spin2=False):
        nz   =  53134                       #From Alexander (80X Moriond ReReco)
        nsig_scale = 1376.0208367514358        # prescale/eff_z (167.73694/0.1219)↩
            -->derives the lumi
        eff = signalEff(mass,spin2)
        result = (nsig_scale*nz*eff)

        return result
#provides the scaling of the signal cross section to the Z peak so that we can ↩
    set limits on the cross section ratio

def signalEff(mass,spin2=False):


        if spin2:
                eff_a = 1.020382
                eff_b = -1166.881533
                eff_c = 1468.989496
                eff_d = 0.000044
                return eff_a + eff_b / (mass + eff_c) - mass*eff_d
        else:
                if mass <= 600:
                        a = 2.129
                        b = 0.1268
                        c = 119.2
                        d = 22.35
                        e = -2.386
                        f = -0.03619
                        from math import exp
                        return a - b * exp( -(mass - c) / d ) + e * mass**f
                else:

                        eff_a     =   2.891
                        eff_b     =  -2.291e+04
                        eff_c     =  8294.
                        eff_d     =  0.0001247

                        return  eff_a + eff_b / (mass + eff_c) - mass*eff_d
#mass depending signal efficiency for the resonant search

def signalEffUncert(mass):
        if mass <= 600:
                a = 2.129
                b = 0.1268
                c = 119.2
                d = 22.38
                e = -2.386
                f = -0.03623
                from math import exp
                eff_default =  a - b * exp( -(mass - c) / d ) + e * mass**f
        else:
                eff_a     =   2.891
                eff_b     =  -2.291e+04
                eff_c     =  8294.
                eff_d     =  0.0001247


                eff_default = eff_a + eff_b / (mass + eff_c) - mass*eff_d
        if mass <= 600:
                a = 2.13
                b = 0.1269
                c = 119.2
                d = 22.42
                e = -2.384
                f = -0.03596
                from math import exp
                eff_syst =  a - b * exp( -(mass - c) / d ) + e * mass**f
        else:

                eff_a     =  2.849
                eff_b     = -2.221e+04
                eff_c     =  8166.
                eff_d     =  0.0001258
                eff_syst = eff_a + eff_b / (mass + eff_c) - mass*eff_d


        effDown = eff_default/eff_syst

        return [1./effDown,1.0]
#one-sided signal efficiency uncertainty from high momentum efficiency loss

def provideUncertainties(mass):

        result = {}

        result["sigEff"] = signalEffUncert(mass)
        result["massScale"] = 0.01
        result["bkgUncert"] = 1.4
        result["res"] = 0.15
        result["bkgParams"] = {"bkg_a":0.0008870490833826137,"bkg_b"↩
            :0.0735080058224163,"bkg_c":0.020865265760197774,"bkg_d"↩
            :0.13546622914957615,"bkg_e":0.0011148272017837235, "bkg_a2"↩
            :0.0028587764436821044,"bkg_b2":0.008506113769271665,"bkg_c2"↩
            :0.019418985270049097,"bkg_e2":0.0015616866215512754}
        return result
# provides all the systematic uncertainties for the resonant analysis
def provideUncertaintiesCI(mass):

        result = {}

        result["trig"] = 1.003
        result["zPeak"] = 1.05
        result["xSecOther"] = 1.07
        result["jets"] = 1.5
        result["lumi"] = 1.025
        result["stats"] = 0.0 ##dummy values
        result["massScale"] = 0.0 ##dummy values
        result["res"] = 0.0 ## dummy values
        result["pdf"] = 0.0 ## dummy values
        result["ID"] = 0.0 ## dummy values
        result["PU"] = 0.0 ## dummy values

        return result
# similar to above, but this time for the non-resonant analysis. A value of 0 ↩
    indicates that these uncertainties are mass-dependent and will be provided as↩
     external histograms

def getResolution(mass):
        result = {}
        params = getRes(mass)
        result['alphaL'] = params['alphaL']['BB']
        result['alphaR'] = params['alphaR']['BB']
        result['res'] = params['sigma']['BB']
        result['scale'] = params['scale']['BB']

        return result
# repackages the mass dependent resolution into the format used in the limit tool

def loadBackgroundShape(ws,useShapeUncert=False):

        bkg_a = RooRealVar('bkg_a_dimuon_Moriond2017_BB','↩
            bkg_a_dimuon_Moriond2017_BB', 33.82)
        bkg_b = RooRealVar('bkg_b_dimuon_Moriond2017_BB','↩
            bkg_b_dimuon_Moriond2017_BB',-0.0001374)
        bkg_c = RooRealVar('bkg_c_dimuon_Moriond2017_BB','↩
            bkg_c_dimuon_Moriond2017_BB',-1.618e-07)
        bkg_d = RooRealVar('bkg_d_dimuon_Moriond2017_BB','↩
            bkg_d_dimuon_Moriond2017_BB', 3.657E-12)
        bkg_e = RooRealVar('bkg_e_dimuon_Moriond2017_BB','↩
            bkg_e_dimuon_Moriond2017_BB',-4.485)
        bkg_a2 = RooRealVar('bkg_a2_dimuon_Moriond2017_BB','↩
            bkg_a2_dimuon_Moriond2017_BB', 17.49)
        bkg_b2 = RooRealVar('bkg_b2_dimuon_Moriond2017_BB','↩
            bkg_b2_dimuon_Moriond2017_BB',-0.01881)
        bkg_c2 = RooRealVar('bkg_c2_dimuon_Moriond2017_BB','↩
            bkg_c2_dimuon_Moriond2017_BB', 1.222e-05)
        bkg_e2 = RooRealVar('bkg_e2_dimuon_Moriond2017_BB','↩
            bkg_e2_dimuon_Moriond2017_BB',-0.8486)
        bkg_a.setConstant()
        bkg_b.setConstant()
        bkg_c.setConstant()
        bkg_d.setConstant()
        bkg_e.setConstant()
        bkg_a2.setConstant()
        bkg_b2.setConstant()
        bkg_c2.setConstant()
        bkg_e2.setConstant()
        getattr(ws,'import')(bkg_a,ROOT.RooCmdArg())
        getattr(ws,'import')(bkg_b,ROOT.RooCmdArg())
        getattr(ws,'import')(bkg_c,ROOT.RooCmdArg())
        getattr(ws,'import')(bkg_d,ROOT.RooCmdArg())
        getattr(ws,'import')(bkg_e,ROOT.RooCmdArg())
        getattr(ws,'import')(bkg_a2,ROOT.RooCmdArg())
        getattr(ws,'import')(bkg_b2,ROOT.RooCmdArg())
        getattr(ws,'import')(bkg_c2,ROOT.RooCmdArg())
        getattr(ws,'import')(bkg_e2,ROOT.RooCmdArg())

        # background systematics
        bkg_syst_a = RooRealVar('bkg_syst_a','bkg_syst_a',1.0)
        bkg_syst_b = RooRealVar('bkg_syst_b','bkg_syst_b',0.0)
        #bkg_syst_b = RooRealVar('bkg_syst_b','bkg_syst_b',-0.00016666666666)
        bkg_syst_a.setConstant()
        bkg_syst_b.setConstant()
        getattr(ws,'import')(bkg_syst_a,ROOT.RooCmdArg())
        getattr(ws,'import')(bkg_syst_b,ROOT.RooCmdArg())

        # background shape
        if useShapeUncert:
                bkgParamsUncert = provideUncertainties(1000)["bkgParams"]
                for uncert in bkgParamsUncert:
                        addBkgUncertPrior(ws,uncert,"dimuon_Moriond2017_BB",↩
                                bkgParamsUncert[uncert] )

                ws.factory("ZPrimeMuonBkgPdf2::bkgpdf_dimuon_Moriond2017_BB(↩
                    mass_dimuon_Moriond2017_BB, ↩
                    bkg_a_dimuon_Moriond2017_BB_forUse, ↩
                    bkg_b_dimuon_Moriond2017_BB_forUse, ↩
                    bkg_c_dimuon_Moriond2017_BB_forUse,↩
                    bkg_d_dimuon_Moriond2017_BB_forUse,↩
                    bkg_e_dimuon_Moriond2017_BB_forUse,↩
                    bkg_a2_dimuon_Moriond2017_BB_forUse, ↩
                    bkg_b2_dimuon_Moriond2017_BB_forUse, ↩
                    bkg_c2_dimuon_Moriond2017_BB_forUse,↩
                    bkg_e2_dimuon_Moriond2017_BB_forUse,bkg_syst_a,bkg_syst_b)")
                ws.factory("ZPrimeMuonBkgPdf2::bkgpdf_fullRange(massFullRange, ↩
                    bkg_a_dimuon_Moriond2017_BB_forUse, ↩
                    bkg_b_dimuon_Moriond2017_BB_forUse, ↩
                    bkg_c_dimuon_Moriond2017_BB_forUse,↩
                    bkg_d_dimuon_Moriond2017_BB_forUse,↩
                    bkg_e_dimuon_Moriond2017_BB_forUse, ↩
                    bkg_a2_dimuon_Moriond2017_BB_forUse, ↩
                    bkg_b2_dimuon_Moriond2017_BB_forUse, ↩
                    bkg_c2_dimuon_Moriond2017_BB_forUse,↩
                    bkg_e2_dimuon_Moriond2017_BB,bkg_syst_a,bkg_syst_b)")

        else:
                ws.factory("ZPrimeMuonBkgPdf2::bkgpdf_dimuon_Moriond2017_BB(↩
                    mass_dimuon_Moriond2017_BB, bkg_a_dimuon_Moriond2017_BB, ↩
                    bkg_b_dimuon_Moriond2017_BB, bkg_c_dimuon_Moriond2017_BB,↩
                    bkg_d_dimuon_Moriond2017_BB,bkg_e_dimuon_Moriond2017_BB,↩
                    bkg_a2_dimuon_Moriond2017_BB, bkg_b2_dimuon_Moriond2017_BB, ↩
                    bkg_c2_dimuon_Moriond2017_BB,bkg_e2_dimuon_Moriond2017_BB,↩
                    bkg_syst_a,bkg_syst_b)")
                ws.factory("ZPrimeMuonBkgPdf2::bkgpdf_fullRange(massFullRange, ↩
                    bkg_a_dimuon_Moriond2017_BB, bkg_b_dimuon_Moriond2017_BB, ↩
                    bkg_c_dimuon_Moriond2017_BB,bkg_d_dimuon_Moriond2017_BB,↩
                    bkg_e_dimuon_Moriond2017_BB, bkg_a2_dimuon_Moriond2017_BB, ↩
                    bkg_b2_dimuon_Moriond2017_BB, bkg_c2_dimuon_Moriond2017_BB,↩
                    bkg_e2_dimuon_Moriond2017_BB,bkg_syst_a,bkg_syst_b)")
        return ws

# provides to background shapes, one in the mass window for the tested resonance ↩
    mass, the other for the full mass range. log-normal priors can be added to ↩
    the shape parameters if desired
```

For each channel of the analysis (i.e. for each subcategory of the dielectron and dimuon channels), one such config has to be provided. The other input to the tool is located in the cfgs/ directory. Here, the scanConguration_ConfigName.py files contain all information needed to steer the actual interpretation, setting the channels to be considered, the mass range to be scanned, and similar features. Given here is the example for the combination of all four subcategories for the 2016 result.

```
leptons = "elmu" # dilepton combination, can also be elel or mumu
systematics = ["sigEff","bkgUncert","massScale",'res',"bkgParams"] # list of ↩
    systematic uncertainties to be considered
correlate = False #should uncertainties be treated as correlated between channels↩
    ?
masses = [[5,200,1000], [10,1000,2000], [20,2000,5500]] #mass scan parameters for↩
     observed limits/p-Value scans
massesExp = [[100,200,600,500,4,500000], [100,600,1000,250,8,500000], ↩
    [250,1000,2000,100,20,50000], [250,2000,5600,100,20,500000]] #mass scan ↩
    parameters for expected limits
```

```
libraries = ["ZPrimeMuonBkgPdf2_cxx.so","ZPrimeEleBkgPdf3_cxx.so","PowFunc_cxx.so↩
    ","RooCruijff_cxx.so"] #libraries to be added to the combine call
channels = ["dielectron_Moriond2017_EBEB","dielectron_Moriond2017_EBEE","↩
    dimuon_Moriond2017_BB","dimuon_Moriond2017_BE"] # list of channels to be ↩
    considered
#Markov Chain parameters
numInt = 500000
numToys = 6
exptToys = 1000

width = 0.006 #signal width (here 0.6%)
submitTo = "FNAL" #computing resources used for batch jobs. Right now Purdue and ↩
    the LPC Condor cluster are supported
LPCUsername = "jschulte" # username at LPC, necessary to run CONDOR jobs there

binWidth = 10 #bin width for binned limits
CB = True # use non-Gaussian signal resolution shape. Does not necessarily have ↩
    to CB anymore
signalInjection = {"mass":750,"width":0.1000,"nEvents":0,"CB":True}#parameters ↩
    for toy generation for MC studies
```

Using this input, the framework will create first the datacards for the single channels and afterwards combined datacards. For local running, this can be triggered by running with the `-w` or `-r` options. In the first case, the datacards are produced and the program is exited without performing any statistical procedures. In the latter case, the datacards are reproduced on the fly before performing statistical interpretations. If a local batch system is used, the input will be created inside the individual jobs to increase performance. When tasks are submitted to CRAB, the input is created locally.

3.2 Running statistical procedures

If not called with the `-w` option (which will only write datacards, see above), the default behaviour of `runInterpretation.py` is to calculate observed limits using the Bayesian approach. For this, the mass binning and the configuration of the algorithm given in the scan configuration is given. There are numerous command line options to modify the statistical methods used

- `-e` switches the limit calculation to expected limits
- `--signif` switches to calculation of p-Values using an approximate formula. For full calculation with the ProfileLikelihoodCalculator, use this option in conjuction with `--plc`
- `--frequentist` uses frequentist calculations for limits or p-Values
- `--hybrid` uses Frequentist-Bayesian Hybrid method for the p-Values

Apart from these fundamental options, there are several further modifications that can be made

3.2.1 Binned limits

The `--binned` option triggers the use of binned instead of unbinned datasets. For this purpose, binned templates are generated from the background and signal PDFs. The binning is hardcoded within the `createInputs.py` script. The advantage of this approach is a large improvement in speed, the disadvantage is a very long time needed to generate the templates in the first place.

3.2.2 Single mass/Λ points

To run a single mass (or Λ in case of CI) point instead of the full scan, the option `-m mass` (`-L Lambda`) can be used.

3.2.3 Spin-2 limits

If run with the `--spin2` option, the signal efficiency for spin-2 resonances will be used

3.2.4 Bias study

If run with the `--bias` option, a bias study will be performed. Two sets of toy datasets are generated based on the datacards, with signal strength $\mu = 0$ and $\mu = 1$. These datasets are then fit with the background + signal model and the fitted $\hat{\mu}$ is recorded. These fit results can then be used to determine if there are biases in the modelling, as we expected the average $< \hat{\mu} >$ to be 0 and 1, respectively.

3.2.5 Signal injection and Look Elsewhere Effect

For performance studies, pseudo-data can be generated in which the statistical interpretation is then performed. When run with the `--inject` option, pseudo background and signal events are generated according to the respective PDFs. The background is normalized to the yield observed in data in each channel, but can be scaled to a desired luminosity. The signal parameters used for the injection are taken from the scan configuration. The signal events are distributed between the sub-channels according to the signal efficiencies. The default behaviour is that a toy dataset for a given configuration is not over-written if the program is rerun, so that the same toy dataset can be processed with the same configuration. The option `--recreateToys` can be used to force the dataset to be overwritten.

To account for the look elsewhere effect, the `--LEE` option can be used. Many background only datasets will be generated and p-Value scans will be performed. The tool `readPValueToys.py` can be used to harvest the large number of resulting result cards.

3.2.6 Contact Interaction limits

So far most options discussed were mostly focused on the statistical analysis for the resonant analysis. To switch the program to perform the analysis for the CI signal, the option `--CI` can be used. This will by default run the multibin shape analysis for constructive interference. With the option `--singlebin`, it can be switched to single bin counting about a certain threshold. The threshold has to be chosen with the `-m` option.

Setting limits not on the signal cross section but on the CI scale Λ will be possible with the `--usePhysicsModel` option, combined with the `--Lower` option to convert the limits from upper into lower limits (not supported in combine for MarkovChainMC calculation). This is still under development at this stage.

3.2.7 Job submission

As the calculations used for the statistical interpretations, parallelization is unavoidable. The framework supports two options for it, submission to local batch systems and CRAB. The `-s` option triggers submission to batch system. At the moment, only the Purdue system is supported. However, the job configurations can be easily used for any qsub system and should be adaptable to others system as well.

Less specific and giving access to much more computing resources is submission via crab. At the moment, only expected and observed Bayesian limits are supported. On the upside, submission is very easy, just run the tool with the `--crab` option. A valid GRID certificate is required.

3.3 Output processing

The output of the combine tool are root files which contain the resulting limit or p-Value as entries in a ROOT tree. The script `createLimitCard.py` is available to convert these files into simple ascii files. This tool takes a variety of arguments, very similar to the main `runInterpretation.py` script:

```
optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        configuration name (default: )
  --input INPUT         folder with input root files (default: )
  -t TAG, --tag TAG     tag (default: )
  --exp                 write expected limits (default: False)
  --signif              write pValues (default: False)
  --injected            injected (default: False)
  --binned              binned (default: False)
  --frequentist         use results from frequentist limits (default: False)
  --hybrid              use results from hybrid significance calculations
                        (default: False)
  --merge               merge expected limits first (default: False)
  --CI                  is CI (default: False)
```

Basically you have to match up the configuration to the one used to create the output. Then you have the choice of either providing the location of the output to be processed with the `--input` option or leave the tool to figure it out for itself. In the latter case, if will take the newest result produced on a local batch system matching the configuration.

Results produced via CRAB have to be downloaded from the respective resource, with the script `harvestCRABResults.py`, which will download and properly rename/merge the files so they can be used with the `createLimitCard.py` script

```
optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        configuration name (default: )
  -t TAG, --tag TAG     tag (default: )
  -u USER, --user USER  name of the user running the script (default: )
  --obs                 renamae obeserved limits (default: False)
  --merge               merge expected limits (default: False)
```

The plot scripst `makeLimitPlot.py`, `makeLimitPlotWidths.py`, `makeLimitPlotCI.py`, `makeRLimitPlotCI.py`, and `makePValuePlot.py` can be used to create plots from the ascii files previously created.