The Compact Muon Solenoid Experiment

CMS Draft Note

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

2015/04/18
Head Id: 92718
Archive Id: 285132P
Archive Date: 2011/12/22
Archive Tag: trunk

A Guide to Monte Carlo Production

C. Contreras-Campana, E. Contreras-Campana, P. Thomassen, M. Walker, P. Zywicki
Rutgers University

Abstract

The purpose of this note is to document the accumulated knowledge that Rutgers has acquired over the years for the production of Monte Carlo (MC) events. We will cover ISAJET, which generate SLHA files, leading-order (LO) MC event generators like PYTHIA and MADGRAPH, validation tools, the CMS software (CMSSW), PROSPINO2, which is a next-to-leading order (NLO) cross section calculator, and the recommended program for producing Feynman diagrams.

This box is only visible in draft mode. Please make sure the values below make sense.

PDFAuthor:	C. Contreras-Campana, E. Contreras-Campana, P. Thomassen, M. Walker, P. Zywicki
PDFTitle:	CMS Paper Template 2006 LaTeX/PdfLaTeX version
PDFSubject:	CMS
PDFKeywords:	CMS, physics, software, computing, MC production

Please also verify that the abstract does not use any user defined symbols

Contents

1 Overview

2 ISAJET

2.1 Introduction

ISAJET 7.83 is a Monte Carlo program which simulates pp, $p\bar{p}$, and e^+e^- interactions at high energies [1]. It is based on perturbative QCD plus phenomenological models for parton and beam jet fragmentation. The manual describes the physics and explains how to use the program.

ISAJET is written in FORTRAN 77 (with a few common extensions) and is distributed using the Patchy code management system developed at CERN. The Patchy source file isajet.car can be be unpacked and compiled on any supported Unix system by editing the Makefile and selecting the appropriate options. Compiling ISAJET on any other computer with ANSI Fortran 77 and Patchy, including any for which CERNlib is supported, should be straightforward.

This should produce the following executables: isajet.x, isasusy.x and isasugra.x. You can run make clean to get rid of the temporary files.

2.2 Producing SLHA files with ISAJET

No information about ISAJET will be included at this time.

2.3 Producing SLHA files with ISASUSY

We give here an example of how to run ISASUSY interactively.

```
[shell prompt]$ ./isasusy.x
 ENTER output file name (in single quotes)
'HadronicRPV_squark1000_gluino1000.dat' <enter>
 ENTER SUSY Les Houches Accord filename [/ for none]:
HadronicRPV_squark1000_gluino1000_temp.slha <enter>
 ENTER Isawig (Herwig interface) filename [/ for none]:
HadronicRPV_squark1000_gluino1000.hwg <enter>
 ENTER M(TP)
172 <enter>
 ENTER M(GLSS), MU, M(A), TAN(BETA)
1000,3000,4000,3 <enter>
 ENTER M(Q1), M(DR), M(UR), M(L1), M(ER)
5000,1000,1000,4000,300 <enter>
 ENTER M(Q3), M(BR), M(TR), M(L3), M(LR), A_T, A_B, A_L
5000,1000,1000,4000,300,1000,9000,9000 <enter>
 ENTER OPTIONAL 2ND GEN MASSES (/ FOR DEFAULT):
 ENTER M(Q2), M(SR), M(CR), M(L2), M(MR)
/ <enter>
 ENTER OPTIONAL GAUGINO MASSES M1, M2 (/ FOR DEFAULT):
500,150 <enter>
 ENTER OPTIONAL GRAVITINO MASS (/ FOR DEFAULT):
/ <enter>
```

The SLHA file has to be corrected with using an awk script.

```
[shell prompt]$ awk -f SLHAprocessing.awk
HadronicRPV_squark1000_gluino1000_temp.slha
```

```
> HadronicRPV_squark1000_gluino1000.slha
[shell prompt]$ rm HadronicRPV_squark1000_gluino1000_temp.slha
```

We have included an example in the appendix on how to run ISASUSY in a shell script (See Listing 9). The shell script runs with the following command,

```
[shell prompt]$ ./HadronicRPV.sh "HadronicRPV_squark1000_gluino1000" 1000 1000
```

One item to point out is that we no longer produce SLHA files for the HadronicRPV model using ISASUSY and then read it into PYTHIA in order to generate LHE files. We now produce SLHA and LHE files directly using PYTHIA. The output SLHA file is not used for anything in the MC production workflow other than for debugging and validation purposes. The above example was given to illustrate how to run ISASUSY.

2.4 Producing SLHA files with ISASUGRA

We give here an example of how to run ISASUGRA interactively.

```
[shell prompt]$ ./isasugra.x
 ENTER output filename in single quotes:
'mSUGRA_mZero275_mHalf150_AZero0_TanBeta3_SignMu1_mTop172.dat' <enter>
 ENTER SUSY Les Houches Accord filename [/ for none]:
mSUGRA_mZero275_mHalf150_AZero0_TanBeta3_SignMu1_mTop172.slha <enter>
 ENTER Isawig (Herwig interface) filename [/ for none]:
mSUGRA_mZero275_mHalf150_AZero0_TanBeta3_SignMu1_mTop172.hwg <enter>
 ENTER 1 for mSUGRA:
 ENTER 2 for mGMSB:
 ENTER 3 for non-universal SUGRA:
 ENTER 4 for SUGRA with truly unified gauge couplings:
 ENTER 5 for non-minimal GMSB:
 ENTER 6 for SUGRA+right-handed neutrino:
 ENTER 7 for minimal anomaly-mediated SUSY breaking:
 ENTER 8 for non-minimal AMSB:
 ENTER 9 for mixed moduli-AMSB:
 ENTER 10 for Hypercharged-AMSB:
1 <enter>
 ENTER M_0, M_(1/2), A_0, tan(beta), sgn(mu), M_t:
275,150,0,3,1,172 <enter>
```

The decay table should be appended to the end of the SLHA file, which has the mass spectra only at this point, in order for it to more closely resemble the conventional form of an SLHA file (i.e. the file should have both the mass spectra and decay tables in it).

```
[shell prompt]$ cat ISALHD.out | awk '{if(NR >= 15) print}' >>
mSUGRA_mZero275_mHalf150_AZero0_TanBeta3_SignMu1_mTop172.slha
```

We have included an example in the appendix on how to run ISASUGRA in a shell script (See Listing 10). The shell script runs with the following command,

```
[shell prompt]$ ./mSUGRA.sh
"mSUGRA_mZero275_mHalf150_AZero0_TanBeta3_SignMu1_mTop172" 275 150 0 3 1 172
```

One item to point out is that ISASUGRA produces the decay tables in a file call ISALHD.out. Therefore running multiple jobs in parallel to generate SLHA files will cause the jobs to overwrite each others ISALHD.out file. This can be avoided by writing a shell script that takes in a list of jobs to run sequentially. We recommend that you DO NOT run jobs in parallel since jobs that produce SLHA files run rather quickly. But if you really do want to run them in parallel then you will have to create temporary directories for each one of your jobs and copy over the ISASUGRA program to every directory. Afterwards you will have to run the executable from within these temporary directories.

3 PYTHIA

3.1 Introduction

PYTHIA is a computer simulation program for particle collisions at very high energies in particle accelerators. PYTHIA was originally written in FORTRAN 77, until the release of PYTHIA 8.1 which was rewritten in C++. Both the Fortran and C++ versions are being maintained because not all components were merged into the 8.1 version. However, the latest version already includes new features not available in the Fortran release. Torbjorn Sjostrand, Stefan Ask, Richard Corke, Stephen Mrenna, Stefan Prestel, and Peter Skands are the main contributors to PYTHIA [2].

The PYTHIA program can be used to generate high-energy-physics 'events', i.e. sets of outgoing particles produced in the interactions between two in-coming particles. The objective is to provide as accurate as possible a representation of event properties in a wide range of reactions, within and beyond the Standard Model, with emphasis on those where strong interactions play a role, directly or indirectly, and therefore multihadronic final states are produced. The physics is then not understood well enough to give an exact description; instead the program has to be based on a combination of analytical results and various QCD-based models. This physics input is summarized here, for areas such as hard subprocesses, initial- and final-state parton showers, underlying events and beam remnants, fragmentation and decays, and much more. Furthermore, extensive information is provided on all program elements: subroutines and functions, switches and parameters, and particle and process data. This should allow the user to tailor the generation task to the topics of interest. The code and further information may be found on the PYTHIA web page (`http://www.thep.lu.se/~torbjorn/Pythia.html`).

3.2 Using a PYTHIA card as input

We will review various PYTHIA examples to discuss different aspects of the configuration file.

We will take a look at the below PYTHIA configuration file which to produce an SLHA and LHE file.

Listing 1: Example LeptonicRPV.f. Pythia configuration file which produces an SLHA file and an LHE file.

```
C...A simple skeleton program, illustrating a typical Pythia run:
C...LeptonicRPV production at CMS LHC.
C...Toy task: compare multiplicity distribution with matrix elements.
C...and with parton showers (using same fragmentation parameters).

```

```
C-----------------------------------------------------------------------

C...Preamble: declarations.

C...All real arithmetic in double precision.
      IMPLICIT DOUBLE PRECISION(A-H, O-Z)
C...Three Pythia functions return integers, so need declaring.
      INTEGER PYK,PYCHGE,PYCOMP

C...EXTERNAL statement links PYDATA on most machines.
      EXTERNAL PYDATA

C...Commonblocks.
C...The event record.
      COMMON/PYJETS/N,NPAD,K(4000,5),P(4000,5),V(4000,5)
C...Parameters.
      COMMON/PYDAT1/MSTU(200),PARU(200),MSTJ(200),PARJ(200)
C...Particle properties + some flavour parameters.
      COMMON/PYDAT2/KCHG(500,4),PMAS(500,4),PARF(2000),VCKM(4,4)
C...Decay information.
      COMMON/PYDAT3/MDCY(500,3),MDME(8000,2),BRAT(8000),KFDP(8000,5)
C...Selection of hard scattering subprocesses.
      COMMON/PYSUBS/MSEL,MSELPD,MSUB(500),KFIN(2,-40:40),CKIN(200)
C...Parameters.
      COMMON/PYPARS/MSTP(200),PARP(200),MSTI(200),PARI(200)
C...Supersymmetry parameters.
      COMMON/PYMSSM/IMSS(0:99),RMSS(0:99)
C...R-parity-violating couplings in supersymmetry.
      COMMON/PYMSRV/RVLAM(3,3,3), RVLAMP(3,3,3), RVLAMB(3,3,3)
C...Random Seed.
      COMMON/PYDATR/MRPY(6),RRPY(100)

C-----------------------------------------------------------------------

C...First section: initialization.
      LOGICAL debug
      INTEGER randomseed, numevnt
      REAL sqrtSinGeV, gluinomass, squarkmass
      CHARACTER coupling*200, slhaoutput*200, txtoutput*200, lheoutput*200

      debug = .TRUE.

C...Reading in the names for all output files.
      READ(*,*) randomseed, numevnt, sqrtSinGeV, gluinomass, squarkmass, couplin

      MRPY(1) = randomseed    ! sets the random seed pythia will use

      IF (debug .EQV. .TRUE.) THEN
```

```
          WRITE(*,*) randomseed, numevnt, sqrtSinGeV, gluinomass, squarkmass
          WRITE(*,*) trim(coupling)
          WRITE(*,*) trim(slhaoutput)
          WRITE(*,*) trim(txtoutput)
          WRITE(*,*) trim(lheoutput)
          WRITE(*,*) trim(lheoutput)//'.init'
          WRITE(*,*) trim(lheoutput)//'.evnt'
        END IF

C...Final SLHA file with spectrum and decay table.
        OPEN(UNIT=9,FILE=trim(slhaoutput)//'.spectrum.slha',STATUS='unknown')
        OPEN(UNIT=10,FILE=trim(slhaoutput)//'.decay.slha',STATUS='unknown')

C...Pythia log output.
        MSTU(11) = 11
        OPEN(UNIT=11,FILE=trim(txtoutput),STATUS='unknown')

C...Temporary files for initialization/event output.
        MSTP(161) = 12
        OPEN(UNIT=12,FILE=trim(lheoutput)//'.init',STATUS='unknown')
        MSTP(162) = 13
        OPEN(UNIT=13,FILE=trim(lheoutput)//'.evnt',STATUS='unknown')

C...Final Les Houches Event file, obtained by combining above two.
        MSTP(163) = 14
        OPEN(UNIT=14,FILE=trim(lheoutput),STATUS='unknown')

C...Main parameters of run: c.m. energy and number of events.
        ECM = sqrtSinGeV
        NEV = numevnt

C...Select LeptonicRPV production processes.
        MSEL = 39                 ! turns on all MSSM processes except Higgs produ
        IMSS( 1) = 1              ! generic SUSY scenario
        IMSS( 3) = 1              ! gluino is pole mass
        IMSS(23) = 9              ! write out spectrum table to SLHA file
        IMSS(24) = 10             ! write out decay table to SLHA file
        IMSS(51) = 3              ! RPV LLE on with user specified couplings
        IMSS(52) = 0              ! RPV LQD off
        IMSS(53) = 0              ! RPV UDD off
        IF (trim(coupling) .EQ. 'LLE122') THEN
          RVLAM(1,2,2) = 0.05     ! LLE coupling
        ELSE IF (trim(coupling) .EQ. 'LLE123') THEN
          RVLAM(1,2,3) = 0.05     ! LLE coupling
        ELSE IF (trim(coupling) .EQ. 'LLE233') THEN
          RVLAM(2,3,3) = 0.05     ! LLE coupling
        END IF
        RMSS( 1) = 300.0          ! bino
```

```
      RMSS( 2) = 3000.0          ! wino
      RMSS( 3) = gluinomass      ! gluino
      RMSS( 4) = 3000.0          ! mu
      RMSS( 5) = 3.0             ! tan beta
      RMSS( 8) = squarkmass      ! left squark (1st-2nd generation)
      RMSS( 9) = squarkmass      ! right down squark (1st-2nd generation)
      RMSS(10) = squarkmass      ! left squark (3rd generation)
      RMSS(11) = squarkmass      ! right down squark (3rd generation)
      RMSS(12) = squarkmass      ! right up squark (3rd generation)
      RMSS( 6) = 3000.0          ! left slepton (1st-2nd generation)
      RMSS( 7) = 3000.0          ! right slepton (1st-2nd generation)
      RMSS(13) = 3000.0          ! left slepton (3rd generation)
      RMSS(14) = 3000.0          ! right slepton (3rd generation)
      RMSS(15) = 4800.0          ! bottom trilinear
      RMSS(16) = 533.3           ! top trilinear
      RMSS(17) = 4800.0          ! tau trilinear
      RMSS(18) = 0.0             ! Higgs mixing angle alpha
      RMSS(19) = 3000.0          ! pseudo-scalar Higgs mass

C...Initialize PYTHIA for LHC.
      CALL PYINIT('CMS','p','p',ECM)

C-----------------------------------------------------------------------

C...Second section: event loop.

C...Begin event loop.
      DO 100 IEV = 1, NEV
        CALL PYUPEV
  100 CONTINUE

C-----------------------------------------------------------------------

C...Third section: produce output and end.

C...Cross section table and partial decay widths.
      CALL PYSTAT(1)
      CALL PYSTAT(2)
      CALL PYUPIN

C...Produce final Les Houches Event File.
      CALL PYLHEF

      CLOSE(10)
      CLOSE(11)
      CLOSE(14)
      END
```

Use the following command to compile the PYTHIA configuration file.

```
[shell prompt]$ gfortran -ffixed-line-length-none test/LeptonicRPV.f
test/pythia-6.4.26.o -o test/LeptonicRPV.out <enter>
```

Use the following command to run the executable file.

```
[shell prompt]$ ./test/LeptonicRPV.out <enter>
1 5000 8.0D3 1100 1200 "LLE123"
"slha/LeptonicRPV_LLE123_gluino1100_squark1200"
"pythialogs/LeptonicRPV_LLE123_gluino1100_squark1200.txt"
"lhe/LeptonicRPV_LLE123_gluino1100_squark1200.lhe" <enter>
```

Additional examples of PYTHIA configuration files.

Listing 2: Example SemiLeptonicRPV.f. Pythia configuration file which produces an SLHA file and an LHE file.

```
      IMSS(51) = 0              ! RPV LLE off
      IMSS(52) = 3              ! RPV LQD on with user specified couplings
      IMSS(53) = 0              ! RPV UDD off
      IF (trim(coupling) .EQ. 'LQD231') THEN
        RVLAMP(2,3,1) = 0.005   ! LQD coupling
      ELSE IF (trim(coupling) .EQ. 'LQD233') THEN
        RVLAMP(2,3,3) = 0.005   ! LQD coupling
      END IF
      RMSS( 1) = 700.0          ! bino
```

Listing 3: Example HadronicRPV.f. Pythia configuration file which produces an SLHA file and an LHE file.

```
      IMSS(51) = 0              ! RPV LLE off
      IMSS(52) = 0              ! RPV LQD off
      IMSS(53) = 3              ! RPV UDD on with user specified couplings
      IF (trim(coupling) .EQ. 'UDD112') THEN
        RVLAMB(1,1,2) = 0.005   ! UDD coupling
      END IF
```

3.3 Using an SHLA file as input

We will take a look at the below PYTHIA configuration file that reads in an SLHA and outputs an LHE file.

Listing 4: Example coNLSP.f. Pythia configuration file which reads in an SLHA file and produces an LHE file.

```
C...Read SLHA file with mass spectrum and decay table.
      OPEN(UNIT=10,FILE=trim(slhainput),STATUS='unknown')

C...Pythia log output.
      MSTU(11) = 11
      OPEN(UNIT=11,FILE=trim(txtoutput),STATUS='unknown')

C...Temporary files for initialization/event output.
```

```
      MSTP(161) = 12
      OPEN(UNIT=12,FILE=trim(lheoutput)//'.init',STATUS='unknown')
      MSTP(162) = 13
      OPEN(UNIT=13,FILE=trim(lheoutput)//'.evnt',STATUS='unknown')

C...Final Les Houches Event file, obtained by combining above two.
      MSTP(163) = 14
      OPEN(UNIT=14,FILE=trim(lheoutput),STATUS='unknown')

C...Main parameters of run: c.m. energy and number of events.
      ECM = sqrtSinGeV
      NEV = numevnt

C...Select coNLSP production processes.
      MSEL = 39                    ! turns on all MSSM processes except Higgs produ
      IMSS( 1) = 11                ! generic SUSY scenario from a SUSY Les Houches
      IMSS(11) = 1                 ! turns on gauge mediation
      IMSS(21) = 10                ! read in spectrum table from SLHA file
      IMSS(22) = 10                ! read in decay table from SLHA file
      RMSS(21) = gravitinomass     ! gravitino mass in units of eV
```

Additional examples of PYTHIA configuration files.

Listing 5: Example mSUGRA.f. Pythia configuration file which reads in an SLHA file and produces an LHE file.

```
C...Select mSUGRA production processes.
      MSEL = 39                    ! turns on all MSSM processes except Higgs produ
      IMSS( 1) = 11                ! generic SUSY scenario from a SUSY Les Houches
      IMSS(21) = 10                ! read in spectrum table from SLHA file
      IMSS(22) = 10                ! read in decay table from SLHA file
```

Listing 6: Example mSUGRA_LRPV.f. Pythia configuration file which reads in an SLHA file and produces an LHE file.

```
C...Select mSUGRA with R-Parity violation production processes.
      MSEL = 39                    ! turns on all MSSM processes except Higgs produ
      IMSS( 1) = 11                ! generic SUSY scenario from a SUSY Les Houches
      IMSS(21) = 10                ! read in spectrum table from SLHA file
      IMSS(22) = 10                ! read in decay table from SLHA file
      IMSS(51) = 3                 ! RPV LLE on with user specified couplings
      IMSS(52) = 0                 ! RPV LQD off
      IMSS(53) = 0                 ! RPV UDD off
      IF (trim(coupling) .EQ. 'LLE122') THEN
        RVLAM(1,2,2) = 0.005       ! LLE coupling
      ELSE IF (trim(coupling) .EQ. 'LLE123') THEN
        RVLAM(1,2,3) = 0.005       ! LLE coupling
      ELSE IF (trim(coupling) .EQ. 'LLE233') THEN
        RVLAM(2,3,3) = 0.005       ! LLE coupling
      END IF
```

4 MADGRAPH with aMC@NLO

4.1 Introduction

MADGRAPH5 is the new version of the MADGRAPH matrix element generator, written in the Python programming language [3]. It implements a number of new, efficient algorithms that provide improved performance and functionality in all aspects of the program. It features a new user interface, several new output formats including C++ process libraries for Pythia 8, and full compatibility with FeynRules for new physics models implementation, allowing for event generation for any model that can be written in the form of a Lagrangian. MADGRAPH5 builds on the same philosophy as the previous versions, and its design allows it to be used as a collaborative platform where theoretical, phenomenological and simulation projects can be developed and then distributed to the high-energy community. We illustrate its capabilities through a few simple phenomenological examples.

MADGRAPH5_aMC@NLO is a framework that aims at providing all the elements necessary for SM and BSM phenomenology, such as the computations of cross sections, the generation of hard events and their matching with event generators, and the use of a variety of tools relevant to event manipulation and analysis. Processes can be simulated to LO accuracy for any user-defined Lagrangian, and the NLO accuracy in the case of QCD corrections to SM processes. Matrix elements at the tree- and one-loop-level can also be obtained.

MADGRAPH5_aMC@NLO is the new version of both MadGraph5 and aMC@NLO that unifies the LO and NLO lines of development of automated tools within the MadGraph family. It therefore supersedes all the MadGraph5 1.5.x versions and all the beta versions of aMC@NLO.

4.2 Setting up MADGRAPH

Download MADGRAPH5 from the MADGRAPH website (`http://madgraph.hep.uiuc.edu/`). To unzip and untar the file use the below command.

```
[shell prompt]$ tar -xzvf MG5_aMC_v2.1.1.tar.gz
```

Optional configurations for the MADGRAPH program to disable it from checking if version is the most up-to-date release and to prevent auto-opening of web browser to display Feynman diagrams. These options are especially useful when submitting batch jobs using condor.

```
[shell prompt]$ emacs -nw MG5_aMC_v2_1_1/input/mg5_configuration.txt
change "#auto_update = 7" to "auto_update = 0"
change "#automatic_html_opening = True" to "automatic_html_opening = False"
```

We are now ready to use MADGRAPH, but first some general information regarding which configuration files are most important for running the program. The two major files are the proc_card_mg5.dat file in the main MADGRAPH directory and the run_card.dat file in the “Template/Cards” directory.

The proc_card_mg5.dat file contains the physics process you wish to generated, as an example,

```
import model_v4 StopHiggsino_stop200_chargino150

# Define multiparticle labels
define p = u u~ c c~ d d~ s s~ b b~ g
```

```
define j = p
define l+ = e+ mu+ ta+
define l- = e- mu- ta-
define vl = ve vm vt
define vl~ = ve~ vm~ vt~

# Specify process(es) to run
generate p p > t1 t1~ @1

# Output processes to MadEvent directory
output StopHiggsino_stop200_chargino150
```

This file is very model dependent and is responsible for the creation of the Feynman diagrams. While the run_card.dat file contains the number of events you wish to generate, the center-of-mass energy of the experiment, the type of collision you wish to have (i.e. pp collisions), and a number of other generator level options. As can be seen this file is for the most part model independent and is responsible for the creation of the LHE file. The only time it is model dependent is when you wish to generate a process that has a specific generator level selection (i.e. changing the generator lepton p_{T} threshold, etc...). Below are the most important lines of the configuration file you should monitor,

```
 10000     = nevents ! Number of unweighted events requested
 314159265 = iseed   ! rnd seed (0=assigned automatically=default))
     1     = lpp1    ! beam 1 type
     1     = lpp2    ! beam 2 type
   6500    = ebeam1  ! beam 1 total energy in GeV
   6500    = ebeam2  ! beam 2 total energy in GeV
```

To test out your local version of the software you may use the example proc_card.dat file provided by MadGraph.

```
[shell prompt]$ cd MG5_aMC_v2_1_1
[shell prompt]$ ./bin/mg5_aMC proc_card.dat
[shell prompt]$ cd PROC_sm_0
[shell prompt]$ ./bin/generate_events -f PROC_sm_0 >& PROC_sm_0.log &
```

As a note when you execute "generate_events" a copy of the "Template" directory will be made with the name specified in the proc_card.dat file. This is where your Feynman diagrams will be stored. If everything worked out correctly then an LHE file should have been produced.

```
[shell prompt]$ cd Events/PROC_sm_0
[shell prompt]$ gunzip unweighted_events.lhe.gz
```

Please review the LHE file to make sure the correct physics process was generated at the desired center-of-mass energy for the collision.

4.3 Using an SHLA file as input

We give below an example of how to generate an LHE file from an SLHA file using MadGraph.

```
[shell prompt]$ cd MG5_aMC_v2_1_1/models
[shell prompt]$ cp -r mssm_v4 StopHiggsino_stop200_chargino150
[shell prompt]$ cd -
[shell prompt]$ cp slha/StopHiggsino_stop200_chargino150.slha
MG5_aMC_v2_1_1/models/StopHiggsino_stop200_chargino150/param_card.dat
[shell prompt]$ cd MG5_aMC_v2_1_1
[shell prompt]$ ./bin/mg5_aMC
../test/StopHiggsino_stop200_chargino150_proc_card.dat
```

The above steps will have generated the Feynman diagrams in the StopHiggsino_stop200_chargino150 directory. You may use the command "firefox index.html" to view the diagrams. We now proceed onto generating the simulated events, which will be stored in an LHE file. But before we go on verify that the "StopHiggsino_stop200_chargino150/Events/param_card.dat" file is the same one as "models/StopHiggsino_stop200_chargino150/param_card.dat" using the "diff" command. Older versions of MadGraph did not guarantee this and so one had to manually move it into the "Cards" directory. The reason to have param_card.dat file in the "Cards" directory is so the information contained in the SLHA file will be inserted into header section of the LHE file. Having such information in the LHE file can help resolve any problems in the MC generation.

Now onto the event generation and creation of the LHE file. Follow the below steps,

```
[shell prompt]$ cp ../test/StopHiggsino_stop200_chargino150_run_card.dat
StopHiggsino_stop200_chargino150/Cards/run_card.dat
[shell prompt]$ cd StopHiggsino_stop200_chargino150
[shell prompt]$ ./bin/generate_events -f StopHiggsino_stop200_chargino150
--nb_core=1
[shell prompt]$ cd Events/StopHiggsino_stop200_chargino150
[shell prompt]$ gunzip unweighted_events.lhe.gz
```

We have now generated an LHE file with top squark pair production.

5 BRIDGE

5.1 Introduction

The BRIDGE (Branching Ratio Inquiry/Decay Generated Events) program is designed to operate with arbitrary models defined within matrix element generators, so that one can simulate events with small final-state multiplicities, decay them with BRIDGE, and then pass them to showering and hadronization programs. BRI can automatically calculate widths of two and three body decays. DGE can decay unstable particles in any Les Houches formatted event file. DGE is useful for the generation of event files with long decay chains, replacing large matrix elements by small matrix elements followed by sequences of decays. BRIDGE is currently designed to work with the MadGraph/MadEvent programs for implementing and simulating new physics models. In particular, it can operate with the MadGraph implementation of the MSSM. In this manual we describe how to use BRIDGE, and present a number of sample results to demonstrate its accuracy.

5.2 Setting up BRIDGE

Download BRIDGE from the BRIDGE website (`http://www.lepp.cornell.edu/Research/TPP/BridgeSoftware.html`). To install, you should place the “BRIDGE” directory you get from the tar file under your MadGraph directory, at the same level as “Template”. You should then be able to run “make”. The makefile assumes that you have the library “libdhelas3.a” in the directory “HELAS/lib” under your MadGraph directory. If you do not, you can edit the makefile under “BRIDGE/source” to point to the right location for the HELAS library (BRIDGE does not rely on MadGraph, and can be used in principle with a standalone HELAS library. You just need to edit the makefile.)

In general, MADGRAPH does not need to be compiled, but we will need to compile the HELAS library. The BRIDGE program will depend on the HELAS library when it is compiled. First, we have to changed the default compiler gfortran to g77 so that the HELAS library can be compatible with the BRIDGE program.

```
[shell prompt]$ cd MG5_aMC_v2_1_1/HELAS
[shell prompt]$ emacs -nw Makefile
Add "FC = g77" after the line "LIBDIR  = ./lib/"
[shell prompt]$ make
[shell prompt]$ ln -s /usr/lib64/libg2c.so.0 lib/libg2c.so
(for hexcms.rutgers.edu and lxplus.cern.ch)
Note: libg2c.so is missing at cmslpc-sl6.fnal.gov
(So you may need to compile BRIDGE on hexcms
or lxplus and then transfer it to cmslpc-sl6)
```

Please follow the instructions below to allow BRIDGE to function within the MADGRAPH directory. We now unzip and untar the BRIDGE file,

```
[shell prompt]$ tar -xzvf BRIDGEv2.25.tar.gz
[shell prompt]$ mv BRIDGE MG5_aMC_v2_1_1/.
[shell prompt]$ cd MG5_aMC_v2_1_1/BRIDGE
[shell prompt]$ emacs -nw source/makefile
change "-ldhelas3" to "-ldhelas3 -lg2c"
[shell prompt]$ make
```

After BRIDGE has compiled properly we need to set the HELAS library back to its original state or else MADGRAPH will stop working properly.

```
[shell prompt]$ cd MG5_aMC_v2_1_1/HELAS
[shell prompt]$ emacs -nw Makefile
Remove "FC = g77"
[shell prompt]$ make clean
[shell prompt]$ make
```

We are now ready to use BRIDGE.

5.3 Using BRIDGE with MADGRAPH

No information about BRIDGE with MADGRAPH will be included at this time.

5.4 Using SUSY BRIDGE with MadGraph

We briefly here discuss how the MSSM is designed to be used with BRIDGE. We provide two executables runBRIsusy.exe and runDGEsusy.exe that are specifically designed for the MSSM. We note however that this in no way means it is necessary to generate new versions of the BRI and DGE executables for a generic model. As discussed in Section 2 any new model implemented in the framework of the Madgraph usrmod can be accommodated regardless of complexity. However, since the MSSM is well defined in its couplings and there exists a standard SLHA interface to spectrum calculators runBRIsusy.exe and runDGEsusy.exe were designed to specifically work with this format. In the future the SUSY versions of runDGE and runBRI may be reincorporated into the non-SUSY runDGE and runBRI, but for now we will explain the existing interface.

As alluded to, the only main difference between the SUSY and non-SUSY versions of BRIDGE is the input format. As discussed in Section 2 the model is defined by four files, particles.dat, interactions.dat, couplings check.txt, and paramcard.dat. The use of the couplings and param card files are what defines the numerical values of the masses and couplings in a generic usrmod file. However, in the context of the MSSM there is a specific format for defining the model parameters and the couplings of the model are well defined. For this reason instead of having the user only interface couplings through theirnumerical values as in the usrmod version of input, the couplings are defined separately in a file SUSYpara.cpp and read directly from paramcard.dat through the SLHA read routines in SLHArw.cpp. The coupling definitions found in SUSYpara.cpp are based upon those written originally for the SMadgraph project, that have since been incorporated into Madgraph v4. If one wanted to modify the format of the MSSM couplings beyond the original assumptions implemented in SMadgraph, the files SUSYpara.cpp and SLHArw.cpp are all that are necessary to be modified.

The actual parameters used from the SLHA formatted input file are those found in the blocks corresponding to mixing matrices for the various supersymmetric particles, masses, SM inputs, A terms, Yukawa couplings and Higgs parameters. Additionally if available BLOCK GAUGE is used to define the SM gauge couplings evolved to the scale specified by the spectrum calculator. BRIDGE does not run the SM couplings so BLOCK GAUGE is used to define couplings at a higher scale if available, if not the default values at mZ are used.

BRIDGE does not make use of the decay tables found inside SLHA files but calculates them internally when runBRIsusy.exe is executed.

```
[shell prompt]$ ./runBRIsusy.exe
../models/StopHiggsino_stop200_chargino150/particles.dat
../models/StopHiggsino_stop200_chargino150/param_card.dat
../models/StopHiggsino_stop200_chargino150/interactions.dat
../models/StopHiggsino_stop200_chargino150/
../models/StopHiggsino_stop200_chargino150/StopHiggsino_stop200_chargino150
blist t1 t1~ n3 x1+ x1- n2 elist 314159265 50000 5 Y
```

We modify the branching ratios by hand using the following series of shell commands so that only decays to a higgsino and a Z boson, or a higgsino and a Higgs boson are possible.

```
[shell prompt]$ cd ../models/StopHiggsino_stop200_chargino150
[shell prompt]$ cat n2_decays.table
```

```
| awk '{if (NF == 3 && $1 == "n1" && $2 == "z")
{printf "%s %s %0.6f\n", $1,$2, 0.500000}
else if (NF == 3 && $1 == "n1" && $2 == "h1")
{printf "%s %s %0.6f\n", $1,$2, 0.500000}
else if (NF == 3 && $1 != "#")
{printf "%s %s %0.6f\n", $1,$2, 0.000000}
else if (NF == 4 && $1 != "#")
{printf "%s %s %s %0.6f\n", $1,$2, $3, 0.000000}
else if ($1 == "#") {print}}' > n2_decays.table_temp
[shell prompt]$ mv n2_decays.table_temp n2_decays.table
[shell prompt]$ cd -
```

We copy a list of susy particles we would like SUSY BRIDGE to decay store in the test directory

```
[shell prompt]$ cp ../test/full_decays_offshell.txt
../test/full_decays_onshell.txt
models/StopHiggsino_stop200_chargino150/.
```

Note: The "full_decays_offshell.txt" and "full_decays_onshell.txt" files were compiled from the list of *.grid* files in the models directory, which had the decays with the largest branchings.

Since $|m_{\tilde{\chi}_1^\pm} - m_{\tilde{\chi}_1^0}| < 175\,\text{GeV}$ then we use the following,

```
[shell prompt]$ cd BRIDGE
[shell prompt]$ ./runDGEsusy.exe
../models/StopHiggsino_stop200_chargino150/particles.dat
../models/StopHiggsino_stop200_chargino150/param_card.dat
../models/StopHiggsino_stop200_chargino150/interactions.dat
../StopHiggsino_stop200_chargino150/Events/StopHiggsino_stop200_chargino150/unweighted_events.lhe
../StopHiggsino_stop200_chargino150/Events/StopHiggsino_stop200_chargino150/unweighted_events_hh.lhe
../models/StopHiggsino_stop200_chargino150/ 314159265 3
../models/StopHiggsino_stop200_chargino150/full_decays_offshell.txt
```

Or else if $|m_{\tilde{\chi}_1^\pm} - m_{\tilde{\chi}_1^0}| > 175\,\text{GeV}$ then we use the following,

```
[shell prompt]$ ./runDGEsusy.exe
../models/StopHiggsino_stop200_chargino150/particles.dat
../models/StopHiggsino_stop200_chargino150/param_card.dat
../models/StopHiggsino_stop200_chargino150/interactions.dat
../StopHiggsino_stop200_chargino150/Events/StopHiggsino_stop200_chargino150/unweighted_events.lhe
../StopHiggsino_stop200_chargino150/Events/StopHiggsino_stop200_chargino150/unweighted_events_hh.lhe
../models/StopHiggsino_stop200_chargino150/ 314159265 3
../models/StopHiggsino_stop200_chargino150/full_decays_onshell.txt
```

Note: When BRIDGE processes an LHE file, which has been subjected to the jet matching procedure, it will strip the jet matching information that is given on an event-by-event basis. It is recommended that you keep the original LHE file in order to recover this information. There are python scripts that can take in the original LHE file, the BRIDGE LHE file, and output a corrected LHE file with the missing jet matching information recovered.

6 MC production validation

6.1 Introducion

It is recommended that 90% of MC production should be devoted to MC validation and 10% to MC production. Because MC production is a time consuming process it is worth the extra time to validate the LHE files as much as possible in order to avoid having to rerun the MC production several times. In the following sections we cover possible ways to validate the physics process that were generated in the LHE file.

6.2 Using the lheReader program

It is recommended that once an LHE file has been produced a series of validations steps should be taken in order to guarantee that the physical processes in it make sense. For this we have a program which takes in an LHE file and produces a TTree.

```
[shell prompt]$ lheReader "myfile.lhe" "myoutput.root" 1 1 1 false
```

where lheReader is the name of the binary, ”myfile.lhe” and ”myoutput.root” are the input and output files, 1 1 1 are the run number, first event number, and lumi number, respectively. A few useful commands are given below.

Plotting the p_{T} distribution of all outgoing and decaying particles.

```
[shell prompt]$ LHETree->Draw("pt","state != -1 && state != 2");
```

Plotting the leading muon p_{T} distribution.

```
[shell prompt]$ LHETree->Draw("Max$(pt)","abs(pdgID) == 13 &&
state != -1 && state != 2");
```

Plotting the $E_{\mathrm{T}}^{\mathrm{miss}}$ distribution produced by neutrinos.

```
[shell prompt]$ LHETree->Draw("Sum$(pt)","(abs(pdgID) == 12 ||
abs(pdgID) == 14 || abs(pdgID) == 16) && state != -1 && state != 2");
```

Plotting the p_{T} distribution for particles with a Z boson for its parent particle.

```
[shell prompt]$ LHETree->Draw("pt","pdgID[mother1-1] == 23 && state==1");
```

6.3 Using the aodsimReader program

It is recommended that once an AODSIM file has been produced from an LHE file that particle decay chains be investigated. For this end we suggest using the aodsimReader program that prints out such decay tables for any given event.

```
[shell prompt]$ aodsimReader inputFilename="Seesaw_M-220_aodsim.root"
maximumEvents=-1 run=1 lumi=1 event=2 maximumEventsToPrint=-1
```

6.4 Using a Cut Flow Analysis to validate MC production

A very helpful analysis one can perform on an MC sample that you may have generated is a Cut Flow Analysis. The main idea of a cut flow analysis is to investigate how the number of generated events changes as they are processed from one step to another. In principle one should be able to follow the number of generated events at the LHE file level all the way to the datacard level. The time consuming aspect of performing a cut flow analysis is tracking down all the event yields in the various steps since there is not a uniform place to look for these values and information. It may involve looking at aodsim log files when multilepton event filtering is involved, looking at the number of entries in an ntuple, integrating histogram to estimate event yields, and calculating how many signal events should land in the different channels within a datacard. You would like to compare the number of events you expect to see at a certain stage to the number of event you are actually seeing in the file. This sort of comparison should be done at each stage: LHE, AODSIM, Ntuple, Histogram, and datacard. If values match at each level then you can achieve a high level of confidence that the MC sample has been produced correctly.

6.5 MC production checklist

Below is a checklist of items to investigate during MC production to avoid making time consuming mistakes or catching mistakes during the workflow of the production.

1. Verify that the correct center-of-mass energy is being set.
2. Review that particles have the correct mass values using the lheReader program.
3. Review that particles have the correct relative branching ratios between different decay modes using the lheReader program.
4. Investigate different kinematic distributions of new physics particles like η and ϕ to make sure they look like they are suppose to.
5. Review that particles are decaying correctly with both the lheRearder and aodsimReader program.
6. Use a Cut Flow Analysis to verify that the correct number of events are being generated at each production step (i.e. at LHE level, at AODSIM level, at Ntuple/Histogram level, and at datacard level, etc...)

7 CMS software and Event Data Model

7.1 Introduction

We use the CMS software (CMSSW) to generate collision events from LHE files. The overall collection of software, referred to as the CMSSW, is built around a Framework, an Event Data Model (EDM), and Services needed by the simulation, calibration and alignment, and reconstruction modules that process event data so that physicists can perform analysis. The primary goal of the Framework and EDM is to facilitate the development and deployment of reconstruction and analysis software.

The CMSSW event processing model consists of one executable, called cmsRun, and many plug-in modules which are managed by the Framework. All the code needed in the event processing (calibration, reconstruction algorithms, etc.) is contained in the modules. The same executable is used for both detector and Monte Carlo data.

The CMSSW executable, cmsRun, is configured at run time by the user's job-specific configuration file. This file tells cmsRun

1. Which data to use
2. Which modules to execute
3. Which parameter settings to use for each module
4. What is the order or the executions of modules, called path
5. How the events are filtered within each path, and
6. How the paths are connected to the output files

Unlike the previous event processing frameworks, cmsRun is extremely lightweight: only the required modules are dynamically loaded at the beginning of the job.

The CMS EDM is centered around the concept of an Event. An Event is a C++ object container for all RAW and reconstructed data related to a particular collision. During processing, data are passed from one module to the next via the Event, and are accessed only through the Event. All objects in the Event may be individually or collectively stored in ROOT files, and are thus directly browsable in ROOT. This allows tests to be run on individual modules in isolation. Auxiliary information needed to process an Event is called Event Setup, and is accessed via the EventSetup.

You will find more infomation on the CMSSW Framework in The CMS Offline WorkBook (`https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBook`).

We recommend that you use the following shell script to setup your CMSSW environment if you have already check out CMSSW release.

Listing 7: Example cms.sh. Setups up CMSSW environment.

```
#!/bin/csh

# Shown for c shell
setenv WORKING_DIRECTORY $PWD

setenv SCRAM_ARCH slc5_amd64_gcc481
setenv VO_CMS_SW_DIR /cms/base/cmssoft
setenv COIN_FULL_INDIRECT_RENDERING 1

# To setup the default cmssw release and enable SRM-Client Tools
source $VO_CMS_SW_DIR/cmsset_default.csh

setenv MYREL CMSSW_7_0_0
setenv MYPROJECT private
setenv MYBASE ${MYPROJECT}/${MYREL}

```

```
# The following command for eval is equivalent to cmsenv
cd ~${USER}/${MYBASE}/src
eval `scramv1 runtime -csh`
cd $WORKING_DIRECTORY

setenv PATH ${PATH}:/usr/local/bin/cms-git-tools
```

7.2 Using the cmsDriver to generate a Hadronizer file

```
cmsDriver.py Hadronizer_MgmMatchTuneZ2star_8TeV_cff.py --filetype=LHE --filein=f
--step=GEN,FASTSIM,HLT:GRun --pileup=2012_Summer_inTimeOnly --geometry=DB
--beamspot=Realistic8TeVCollision --conditions=auto:startup --eventcontent=AODSI
--datatier=GEN-SIM-DIGI-AODSIM --number=5000 --mc --no_exec
--python_filename=Hadronizer_TuneZ2star_8TeV_cfi_py_LHE_GEN_FASTSIM_HLT_PU_AODSI
```

To run the Hadronizer file use the following command,

```
[shell prompt]$ cmsRun Hadronizer_TuneZ2star_8TeV_cfi_py_LHE_GEN_FASTSIM_HLT_PU_AODSIM_52X.py
```

7.3 Using an SLHA file directly with CMSSW

Even though an SLHA file can be read in directly into CMSSW framework using the PYTHIA interface this is not really recommended because useful physics validations steps are skipped so that one has to wait for generation, simulation, and reconstruction to finish before anything can be verified. If indeed there was something incorrect in the SLHA file hours of MC production would have been lost since the samples would have to be resubmitted with the updated corrections. Therefore it is recommended that LHE files be produced from the SLHA file in order to validate that the correct physics is being generated since LHE files are quicker to produce than AODSIM (or miniAODSIM).

Below is an example given for completeness of how to use an CMSSW configuration file that can read in an SLHA file directly with the PYTHIA interface.

7.4 Using an EDFilter with CMSSW

Below are instructions how how to produce an EDFilter to use in a CMSSW Hadronizer file.

```
[shell prompt]$ cd CMSSW_7_0_0/src
[shell prompt]$ cmsenv
[shell prompt]$ mkedfltr RutgersGenFilter
```

And here is an example file of a Hadronizer file which is designed to make use of an EDFilter (See Listing 11 for an example of an EDFilter code).

```
# Auto generated configuration file
# using:
# Revision: 1.372.2.14
# Source: /local/reps/CMSSW/CMSSW/Configuration/PyReleaseValidation/python/Confi
# with command line options: Configuration/GenProduction/Hadronizer_MgmMatchTun
import sys
import FWCore.ParameterSet.Config as cms
```

```
process = cms.Process('HLT')

# Import of standard configurations
process.load('Configuration.StandardSequences.Services_cff')
process.load('SimGeneral.HepPDTESSource.pythiapdt_cfi')
process.load('FWCore.MessageService.MessageLogger_cfi')
process.load('FastSimulation.Configuration.EventContent_cff')
process.load('FastSimulation.Configuration.FamosSequences_cff')
process.load('FastSimulation.PileUpProducer.PileUpSimulator_2012_Summer_inTimeO
process.load('FastSimulation.Configuration.Geometries_START_cff')
process.load('Configuration.StandardSequences.MagneticField_38T_cff')
process.load('Configuration.StandardSequences.Generator_cff')
process.load('GeneratorInterface.Core.genFilterSummary_cff')
process.load('IOMC.EventVertexGenerators.VtxSmearedParameters_cfi')
process.load('HLTrigger.Configuration.HLT_GRun_Famos_cff')
process.load('Configuration.StandardSequences.FrontierConditions_GlobalTag_cff')

TAG = sys.argv[2]
NAMEONE = sys.argv[3]
MASSONE = sys.argv[4]
NAMETWO = sys.argv[5]
MASSTWO = sys.argv[6]
NUMBER = sys.argv[7]
NAMETHREE = sys.argv[8]
JOB = sys.argv[9]
EVENTS = sys.argv[10]
STOREDIR = sys.argv[11]

MAXEVENTS = 1000

INPUTFILENAME = 'file:'+STOREDIR+'/lhe/'+TAG+'_'+NAMEONE+MASSONE+'_'+NAMETWO+MAS
OUTPUTFILENAME = STOREDIR+'/aodsim/'+TAG+'_'+NAMEONE+MASSONE+'_'+NAMETWO+MASSTWO

process.maxEvents = cms.untracked.PSet(
    input = cms.untracked.int32(MAXEVENTS)
)

# Input source
process.source = cms.Source("LHESource",
    fileNames = cms.untracked.vstring(
        INPUTFILENAME
    ),
    firstRun   = cms.untracked.uint32(int(NUMBER)),
    firstEvent = cms.untracked.uint32(int(EVENTS)),
    skipEvents = cms.untracked.uint32(int(EVENTS)-1)
)
```

```
process.options = cms.untracked.PSet(

)

# Production info
process.configurationMetadata = cms.untracked.PSet(
    version = cms.untracked.string('$Revision: 1.372.2.14 $'),
    annotation = cms.untracked.string('Configuration/GenProduction/Hadronizer_Mg
    name = cms.untracked.string('PyReleaseValidation')
)

# Output definition
process.AODSIMoutput = cms.OutputModule("PoolOutputModule",
    eventAutoFlushCompressedSize = cms.untracked.int32(15728640),
    outputCommands = process.AODSIMEventContent.outputCommands,
    fileName = cms.untracked.string(
        OUTPUTFILENAME
    ),
    dataset = cms.untracked.PSet(
        filterName = cms.untracked.string(''),
        dataTier = cms.untracked.string('GEN-SIM-DIGI-AODSIM')
    ),
    SelectEvents = cms.untracked.PSet(
        SelectEvents = cms.vstring('mypath')
    )
)

# Additional output definition

# Other statements
process.famosSimHits.SimulateCalorimetry = True
process.famosSimHits.SimulateTracking = True
process.simulation = cms.Sequence(process.simulationWithFamos)
process.HLTEndSequence = cms.Sequence(process.reconstructionWithFamos)
process.Realistic8TeVCollisionVtxSmearingParameters.type = cms.string("BetaFunc"
process.famosSimHits.VertexGenerator = process.Realistic8TeVCollisionVtxSmearing
process.famosPileUp.VertexGenerator = process.Realistic8TeVCollisionVtxSmearingP

# Customise the HLT menu for running on MC
from HLTrigger.Configuration.customizeHLTforMC import customizeHLTforMC
process = customizeHLTforMC(process)

process.GlobalTag.globaltag = 'START52_V10::All'

process.generator = cms.EDFilter("Pythia6HadronizerFilter",
    pythiaPylistVerbosity = cms.untracked.int32(1),
    pythiaHepMCVerbosity = cms.untracked.bool(True),
    filterEfficiency = cms.untracked.double(1.0),
```

```
    crossSection = cms.untracked.double(-1),
    comEnergy = cms.double(8000.0),
    maxEventsToPrint = cms.untracked.int32(0),
    PythiaParameters = cms.PSet(
        pythiaUESettings = cms.vstring(
            'MSTU(21) = 1        ! Check on possible errors during program execut
            'MSTJ(22) = 2        ! Decay those unstable particles',
            'PARJ(71) = 10.      ! for which ctau  10 mm',
            'MSTP(33) = 0        ! no K factors in hard cross sections',
            'MSTP(2)  = 1        ! which order running alphaS',
            'MSTP(51) = 10042    ! structure function chosen (external PDF CTEQ6L
            'MSTP(52) = 2        ! work with LHAPDF',
            'PARP(82) = 1.921    ! pt cutoff for multiparton interactions',
            'PARP(89) = 1800.    ! sqrts for which PARP82 is set',
            'PARP(90) = 0.227    ! Multiple interactions: rescaling power',
            'MSTP(95) = 6        ! CR (color reconnection parameters)',
            'PARP(77) = 1.016    ! CR',
            'PARP(78) = 0.538    ! CR',
            'PARP(80) = 0.1      ! Prob. colored parton from BBR',
            'PARP(83) = 0.356    ! Multiple interactions: matter distribution par
            'PARP(84) = 0.651    ! Multiple interactions: matter distribution par
            'PARP(62) = 1.025    ! ISR cutoff',
            'MSTP(91) = 1        ! Gaussian primordial kT',
            'PARP(93) = 10.0     ! primordial kT-max',
            'MSTP(81) = 21       ! multiple parton interactions 1 is Pythia defau
            'MSTP(82) = 4        ! Defines the multi-parton model'),
        processParameters = cms.vstring(
            'MSEL = 0            ! User defined processes',
            'PMAS(5,1) = 4.8     ! b quark mass',
            'PMAS(6,1) = 172.5   ! t quark mass',
            'MSTJ(1)  = 1        ! Fragmentation/hadronization on or off',
            'MSTP(61) = 1        ! Parton showering on or off'),
        parameterSets = cms.vstring(
            'pythiaUESettings',
            'processParameters')
    )
)

process.multileptonFilter = cms.EDFilter('MultiLeptonFilter',
    src = cms.InputTag("genParticles"),
    edfilterOn = cms.bool(True),
    multlepFilterOn = cms.bool(True),
    mixModeFilterOn = cms.bool(False),
    nLepton = cms.int32(2),
    debug = cms.bool(False)
)

process.myfilter = cms.Sequence(process.multileptonFilter)
```

```
process.mypath = cms.Path(process.myfilter*process.reconstructionWithFamos)

# Path and EndPath definitions
process.generation_step = cms.Path(process.pgen_genonly)
process.reconstruction = cms.Path(process.reconstructionWithFamos)
process.genfiltersummary_step = cms.EndPath(process.genFilterSummary)
process.AODSIMoutput_step = cms.EndPath(process.AODSIMoutput)

# Schedule definition
#process.schedule = cms.Schedule(process.generation_step,process.genfiltersummar
#process.schedule.extend(process.HLTSchedule)
#process.schedule.extend([process.reconstruction,process.AODSIMoutput_step])

# filter all path with the production filter sequence
for path in process.paths:
      getattr(process,path)._seq = process.generator * process.genParticles *
```

8 Jet matching

8.1 Introduction

The aim of any parton-jets matching procedure is mainly to avoid overlapping between phase-space descriptions given by matrix-element generators and showering/hadronization softwares in multi-jets process simulation. The motivation for using both at the same time is the following:

1. The Parton Shower (PS) Monte Carlo programs such as Pythia and Herwig describe parton radiation as successive parton emissions using Markov chain techniques based on Sudakov form factors. This description is formally correct only in the limit of soft and collinear emissions, but has been shown to give a good description of much data also relatively far away from this limit. However, for the production of hard and widely separated QCD radiation jets, this description breaks down due to the lack of subleading terms and interference. For that case, it is necessary to use the full tree-level amplitudes for the heavy particle production plus additional hard partons.

2. The Matrix Element (ME) description diverges as partons become soft or collinear, while the parton shower description breaks down when partons become hard and widely separated.

In MadEvent, three versions of matching are implemented:

1. MLM matching with cone jets (as in AlpGen)

2. MLM matching with kt jets (where there are two options for Pythia treatment, the normal MLM procedure or the "Shower kT" scheme)

3. CKKW matching with Pythia P_T shower Sudakov form factors (this option is under development)

The matching scheme (CKKW or MLM) is chosen by the setting of the parameter ickkw in the run_card.dat (ickkw=0 for no matching, 1 for MLM matching and 2 for CKKW matching). The use of cone jets or kt jets is decided by whether the parameter xqcut (specifying the minimum kt jet measure between jets, i.e. gluons or quarks (except top quarks) which are connected in Feynman diagrams) in the run_card.dat is 0 or not. If xqcut=0, cone jets are used, while if xqcut > 0, kt jet matching is assumed. In this case, ptj and drjj should be set to zero. Note: For most processes, the generation speed can be improved by setting ptj and mjj to xqcut, which is done automatically if the flag auto_ptj_mjj is set to T. If some jets should not be restricted this way (as in single top or vector boson fusion (VBF) production, where some jets are not radiated from QCD), auto_ptj_mjj should be set to F.

If ickkw>0, MadEvent will cluster each event to find its corresponding ”parton shower history”. This clustering is done according to the Durham kt algorithm, allowing only clusterings corresponding to Feynman diagrams for the process in question (thereby avoiding e.g. clustering of two gluons to a Z). For each clustered QCD vertex, the scale of alpha_s is set to be the kt jet measure value in that vertex. This corresponds to reweighting each alpha_s to the value it would get in a corresponding parton shower. The clustering value for each final-state parton is printed as a comment for each event in the output LHE event file.

If ickkw=2, MadEvent will also apply a Sudakov suppression factor for each internal parton line, with starting and ending scales corresponding to the scales in the surrounding vertices. Please note that this option is still under development.

The MadEvent parameters affecting the matching are the following:

1. ickkw: 0 for no matching, 1 for MLM matching and 2 for CKKW matching
2. xqcut: minimum jet measure (pT/kT) for QCD partons, if xqcut=0 use cone jet matching, if xqcut>0 use kt jet matching. This value should be related to the hard scale (e.g. mass of produced particle, HT cut, or similar) in the process, and set to ~(1/6-1/3 x hard scale). Please check that the differential jet rate plots (which are automatically generated if you have MadAnalysis and Root properly installed on your system) are smooth, and check that the cross section does not vary significantly when the xqcut is varied up and down.
3. ptj, ptb, drjj, drbb, drbj: For cone jet matching. Note that for kt jet matching, ptj and ptb should be set to xqcut while drjj, drbj and drbb should be set to 0.
4. fixed_ren_scale, fixed_fac_scale: (default F) If false, use the highest kt jet measure, or mT of the central produced particles, as factorization and renormalization scales for non-emission vertices (see below). If true, use the fixed scales as factorization and renormalization scale for non-emission vertices.
5. scalefact: (default 1) Factor to multiply the jet measure in the factorization scale and non-emission vertices
6. alpsfact: (default 1) Factor to multiply the jet measure in emission vertices
7. maxjetflavor: (default 4) Defines which partons are considered as ”j” and which are considered as ”b”. If matching is including b quarks, set to ”5”, while if b-quarks are not considered as partons in the proton, set to ”4”. This option is fully supported from MadGraph 5 v. 1.3.18 and Pythia/PGS package v. 2.1.10.

8. pdfwgt: (default F) Whether emission vertices should be reweighted by the relative PDF factors relating to the vertices. This is needed for a fully consistent description. Note that this option is fully implemented only in MadGraph 5 v. 1.3.18.

9. ktscheme, chcluster, highestmult: Experiment parameters, leave at default.

A comment on renormalization and factorization scales: Emission vertices are all QCD vertices where a gluon or light quark (including bottom) are emitted, except the vertex with the highest kt jet measure (e.g. the q-qbar-g vertex in top quark pair production by an s-channel gluon). Only for those vertices is alpha_s evaluated at the jet measure scale. All other vertices are considered to be non-emission vertices. The factorization scale (either the highest kt jet measure or the given fixed scale depending of the value of fixed_fac_scale) is also used as starting scale for the parton shower in the Pythia run. Note that for t-channel singlet exchange processes such as single top or VBF, the factorization scale is set to the pt of the scattered parton on each side of the event. For 4-flavor matching (where b quarks are considered as heavy particles and not as partons), the factorization scale is set to the geometric average of the highest pT_b and the central m_T scale.

When the event file is read in the Pythia package, the ickkw parameter is automatically read and matching is turned on, using the routine UPVETO. In this routine, which is called for each event after parton showering but before decays and hadronization, the event is clustered using the corresponding jet clustering scheme (cone jets or kt jets), and the event is rejected or accepted depending on whether the resulting jets correspond to final-state partons in the MadEvent event. For the highest jet multiplicity, extra jets are allowed if they are not harder than the softest MadEvent jet. From MadGraph 5 v. 1.3.18 and Pythia/PGS package v. 2.1.10, non-radiation jets such as the scattered jets in VBF are not included in the matching (but final state radiation from such particles is matched consistently), which allows for variation of the matching scale (xqcut/QCUT) in a consistent way also for such processes.

Either the virtuality-ordered showers (chosen by setting MSTP(81)<20) or the pT-ordered showers (MSTP(81)=20 or 21) can be used in the Pythia run. For the pT-ordered shower, there is an option to use the "shower kT" scheme. This scheme uses information from Pythia about the hardness of the first shower emission to reject events, which means that the same value can be used for QCUT and xqcut.

The Pythia parameters (given in the pythia_card.dat) relevant for matching are:

1. IEXCFILE: 1 for exclusive samples (not including the highest jet multiplicity), 0 for inclusive samples (including the highest jet multiplicity)

2. QCUT. For matching using the kt scheme, this is the jet measure cutoff used by Pythia. If not given, it will be set to max(xqcut+5,xqcut*1.2) (where xqcut is read from the MadEvent run_card.dat).

3. MINJETS: Minimum jet multiplicity included in the matching (default -1: lowest multiplicity in file)

4. MAXJETS: Maximum jet multiplicity included in the matching (default -1: highest multiplicity in file)

5. KTSCHE: The kt clustering scheme used by KTCLUS. Default 4313 for hadron collisions, 1 for e+e- collisions.

6. SHOWERKT=T: The ”shower kt scheme” is used. Only valid for pT-ordered showers.
7. EXCRES=PDG: Discard event with on-shell resonance PDG in event file. Repeat for additional resonances.

Please see `http://arxiv.org/abs/0706.2569`, especially sections 2.3 (for MLM matching with cone jet clustering) and 2.4 (for kt jet matching), and `http://arxiv.org/abs/0810.5350` (for shower kt), for further details. SHOWERKT (only usable with pt-ordered showers) means that Pythia determines whether to veto events based on the kt values of the hardest shower emission instead of performing jet clustering and comparing with the matrix element. This allows to set QCUT=xqcut, which allows using more of the ME events and therefore improves statistics.

Note that there are special processes, such as p p > t bj̃ + p p > t bj̃ j (with 4-flavor matching) which contains a mix of different processes with different highest jet multiplicity - in this case, t-channel single top (with leading order process p p > t bj̃), for which p p > t bj̃ j contains only one radiated jet, and s-channel single top (with leading order process p p > t b)̃ for which p p > t bj̃ j contains two radiated jets. In this case, Pythia can not automatically perform the highest multiplicity correctly, and the highest multiplicity (in this case 1 jet) has to be set explicitly in the pythia_card.dat file using MAXJETS=1 (`https://cp3.irmp.ucl.ac.be/projects/madgraph/wiki/IntroMatching` or `https://cp3.irmp.ucl.ac.be/projects/madgraph/wiki/Matching`).

Please always check the following after performing matching:

1. The cross section (at the end of the pythia.log) should match the cross section for the 0-jet sample within 20% or so
2. All jet-related distributions should be smooth. You can use MatchChecker to check the differential jet rate distributions for the matched jets. If ROOT, MadAnalysis and td are correctly installed, you will automatically get differential jet rate distributions on the pythia plot page on the generation result page.
3. As an additional check for large productions, vary the xqcut and QCUT scales around the starting value and ensure that post-matching cross sections and jet-related observables do not vary too much. Note that once you get outside of the range of validity, the results have no physical meaning.

8.2 Differential Jet Rate plot

Below are instructions on how to make Differential Jet Rate (DJR) plots which will help determine the best qcut value to use for a given xqcut value when hadronizing an LHE file with PYTHIA.

```
[shell prompt]$ cmsRun
Hadronizer_MgmMatchTuneZ2star_8TeV_madgraph_tauola_cff_py_GEN.py
inputFilename="WinoNLSP_chargino130_bino1_101_hw_www.lhe"
outputFilename="WinoNLSP_chargino130_bino1_101_hw_www.root"
maximumEvents=-1 qCut=23
[shell prompt]$ root -b -l -n 'plotDJR.C("events.tree",
```

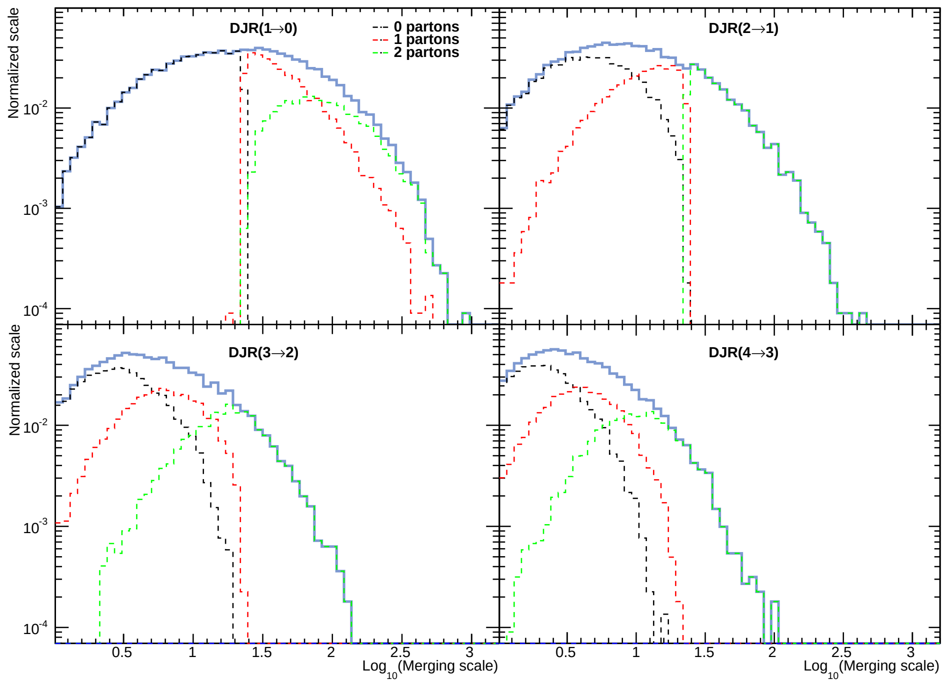

Figure 1: An example of a JDR plot for the Wino NLSP scenario for a top squark mass of 200 GeV and chargino mass of 150 GeV for xqcut = 15 and qcut = 23.

```
"WinoNLSP_chargino130_bino1_101_hw_www_DJR.pdf",
"WinoNLSP_chargino130_bino1_101_hw_www.root")'
```

9 PROSPINO

9.1 Introduction

PROSPINO2 is a computer program which computes next-to-leading order cross sections for the production of supersymmetric particles at hadron colliders [4]. The processes currently included are squark, gluino, stop, neutralino/chargino, and slepton pair production. They have also included the associated production of squarks with gluinos and of neutralinos/charginos with gluinos or squarks and most recently leptoquark pair production.

The physics results are usually published, so you can cross check the results you get. As far as referencing goes, they will not publish a complete Prospino2 manual, because there are physics papers available for all processes. Instead, we would like you to reference the published papers for the respective processes.

There has been a Fortran77 version of Prospino availabe for several years. The increased number of processes and the more complex set of input parameters have made it more convenient to migrate to Fortran90. The new code should run with any F90 compiler - for example the free gfortran, which even runs on your Macs.

Prospino2 can easily be linked to C++ programs, reads the Les Houches SUSY spectrum files and includes a set of easily accessible interfaces.

For the third generation PROSPINO2.1. allows you to treat the sbottom and stop channels differently. In this version there are also switches to compute the combined renormalization/factorization scale variation and allow for non-mass-genenerate light-flavor squarks (`http://www.thphys.uni-heidelberg.de/~plehn/index.php?show=prospino&visible=tools`).

9.2 Calculating cross sections

You will have to download PROSPINO and untar it.

```
[shell prompt]$ tar -xzvf on_the_web_9_23_12.tar.gz
```

In order to calculate a cross section for you signal you will have to create a softlink to your SLHA file in on_the_web_9_23_12 and modify the configuration file.

```
[shell prompt]$ ln -sf ln -sf
../Pythia/slha/coNLSP_chargino1400_gluino1700_delta0.slha
prospino.in.les_houches
```

We now configure the PROSPINO program to calculate gluino-gluino production cross section for the slepton co-NLSP scenario. Below is an example of the configuration file.

Listing 8: Example of prospino_main.f90. PROSPINO configuration file which will calculate NLO cross sections.

```
program main
  use xx_kinds
  use xx_prospino_subroutine
  implicit none

  integer                          :: inlo,isq_ng_in,icoll_in,i_error_in,ipa
  logical                          :: lfinal
  character(len=2)                 :: final_state_in

!----------------------------------------------------------------------------
  inlo = 1          ! specify LO only[0] or complete NLO (slower)[1]
!
!                   ! results: LO     - leading order, degenerate squarks
!
!                   !          NLO    - NLO, degenerate squarks
!
!                   !          LO_ms  - leading order, free squark masses
!
!                   !          NLO_ms - NLO, free squark masses
!
!                   ! all numerical errors (hopefully) better than 1%
!
!                   ! follow Vergas iteration on screen to check
!
!----------------------------------------------------------------------------
```

```
!----------------------------------------------------------------------
  isq_ng_in = 0     ! specify degenerate [0] or free [1] squark masses
!
                    ! [0] means Prospino2.0 with average squark masses
!
                    ! [0] invalidates isquark_in switch
!
!----------------------------------------------------------------------

!----------------------------------------------------------------------
  icoll_in = 3      ! collider : tevatron[0], lhc14[1], lhc7[2], lhc8[3]
!
!----------------------------------------------------------------------

!----------------------------------------------------------------------
  i_error_in = 1    ! with central scale [0] or scale variation [1]
!
!----------------------------------------------------------------------

!----------------------------------------------------------------------
  final_state_in = 'gg'
!
!
!
!                   ng     neutralino/chargino + gluino
!
!                   ns     neutralino/chargino + squark
!
!                   nn     neutralino/chargino pair combinations
!
!                   ll     slepton pair combinations
!
!                   sb     squark-antisquark
!
!                   ss     squark-squark
!
!                   tb     stop-antistop
!
!                   bb     sbottom-antisbottom
!
!                   gg     gluino pair
!
!                   sg     squark + gluino
!
!                   lq     leptoquark pairs (using stop1 mass)
!
!                   le     leptoquark plus lepton (using stop1 mass)
!
```

```
!                    hh      charged Higgs pairs (private code only!)
!
!                    ht      charged Higgs with top (private code only!)
!
!
!
!  squark and antisquark added, but taking into account different sb or ss
!
!-------------------------------------------------------------------------------

!-------------------------------------------------------------------------------
   ipart1_in = 1
!
   ipart2_in = 1
!
!
!
!  final_state_in = ng,ns,nn
!
!  ipart1_in   = 1,2,3,4  neutralinos
!
!                5,6      positive charge charginos
!
!                7,8      negative charge charginos
!
!  ipart2_in the same
!
!      chargino+ and chargino- different processes
!
!
!
!  final_state_in = ll
!
!  ipart1_in   = 0          sel,sel + ser,ser  (first generation)
!
!                1          sel,sel
!
!                2          ser,ser
!
!                3          snel,snel
!
!                4          sel+,snl
!
!                5          sel-,snl
!
!                6          stau1,stau1
!
```

```
!                     7           stau2 ,stau2
!
!                     8           stau1 ,stau2
!
!                     9           sntau ,sntau
!
!                    10           stau1+,sntau
!
!                    11           stau1-,sntau
!
!                    12           stau2+,sntau
!
!                    13           stau2-,sntau
!
!                    14          H+,H-  in  Drell-Yan  channel
!
!
!
!   final_state_in  =  tb  and  bb
!
!   ipart1_in    = 1           stop1/sbottom1  pairs
!
!                     2           stop2/sbottom2  pairs
!
!
!
!   note:  otherwise  ipart1_in ,ipart2_in  have  to  set  to  one  if  not  used
!
!
!
!--------------------------------------------------------------------------------

!--------------------------------------------------------------------------------
  isquark1_in  =  0
!
  isquark2_in  =  0
!
!
!
!   for LO with  light-squark  flavor  in  the  final  state
!
!   isquark1_in       =   -5,-4,-3,-2,-1,+1,+2,+3,+4,+5
!
!                           (bL  cL  sL  dL  uL  uR  dR  sR  cR  bR)  in  CteQ  ordering
!
!   isquark1_in       = 0  sum  over  light-flavor  squarks  throughout
!
!                              (the  squark  mass  in  the  data  files  is  then  averaged) !
```

```
!
!
!   flavors in initial state: only light-flavor partons, no bottoms
!
!                               bottom partons only for Higgs channels
!
!
!
!   flavors in final state: light-flavor quarks summed over five flavors
!
!
!
!----------------------------------------------------------------------
```

We use inlo=1 to calculate the NLO cross section. Since the squark masses are degenerate in the slepton co-NLSP scenario we use the isq_ng_in = 0 option. We are interested in physics at the LHC with $\sqrt{s} = 8$ TeV so we use icoll_in = 3. We are also interested in seeing how the NLO cross section varies with different factorization scales (μ_f) in order to estimate a theory uncertainty though this has not been the case in the past. We can calculate the cross section with $0.5\mu_{\mathrm{F}}$ (scale down), μ_{F} (central), and $2\mu_{\mathrm{F}}$ (scale up) by setting i_error_in = 1. To calculate the gluino-gluino cross section we set final_state_in = 'gg'. Since we are not calculating anything with neutralinos/charginos, sleptons, top or bottom squarks we set ipart1_in = 1 and ipart2_in = 1. Again, we are not doing anything with squarks we can set isquark1_in = 0 and isquark2_in = 0.

We can now compile and run the program.

```
[shell prompt]$ make
[shell prompt]$ ./prospino_2.run >&
coNLSP_gluino1700_chargino1400_gg.log &
```

The central value for the gluino-gluino cross sections is in prospino.dat file with a value of 0.454E-04 pb.

We now try to calculate the squark-gluino and squark-squark cross section. To do this we open up prospino_main.f90 and make the following modifications final_state_in = 'sg' and we keep ipart1_in = 1 and ipart2_in = 1 the same. We keep isquark1_in = 0 and isquark2_in = 0 since we are interested in all possible flavors of squarks. Similarly, for squark-squark production we use final_state_in = 'ss'. The model in which ipart1_in = 1 and ipart2_in = 1 where ever changed was for the natural higgsino NLSP scenario which had top squark pair production and neutralino-neutralino pair production. We never ran into a scenario in which isquark1_in = 0 and isquark2_in = 0 were set to anything other than zero.

10 Drawing Feynman diagrams with feynMF/feynMP

10.1 Introduction

feynMF (Feyman Metafont) and feyMP (Feyman Metapost) are packages made by Thorsten Ohl to draw Feynman diagrams in LaTeXenvironment [5]. You can download bundled feynmf.zip from `http://www.ctan.org/tex-archive/macros/latex/contrib/feynmf`.

11 Summary

A brief guide has been written that spans a wide variety of MC production tools. In the interest of brevity and relevance we have decided against a more comprehensive guide. The softwares documented in this note were the ones used most often during these last few years by the Rutgers multilepton group for CMS analyses at the LHC during Run I.

12 References

References

[1] H. Baer, F. E. Paige, S. D. Protopopescu, and X. Tata, "ISAJET 7.48: A Monte Carlo event generator for pp, $p\bar{p}$, and e^+e^- reactions", arXiv:hep-ph/0001086.

[2] T. Sjostrand, S. Mrenna, and P. Z. Skands, "PYTHIA 6.4 Physics and Manual", *JHEP* **0605** (2006) 026, doi:10.1088/1126-6708/2006/05/026, arXiv:hep-ph/0603175.

[3] F. Maltoni and T. Stelzer, "MadEvent: Automatic event generation with MADGRAPH", *JHEP* **02** (2003) 027, doi:10.1088/1126-6708/2003/02/027, arXiv:hep-ph/0208156v1.

[4] W. Beenakker, R. Hopker, and M. Spira, "PROSPINO: A program for the production of supersymmetric particles in next-to-leading order QCD", 1996, arXiv:hep-ph/9611232v1.

[5] T. Ohl, "Drawing Feynman diagrams with Latex and Metafont", *Comput. Phys. Commun.* **90** (1995) 340–354, doi:10.1016/0010-4655(95)90137-S, arXiv:hep-ph/9505351.

A Appendix

Example of a shell script that produces an SLHA file using ISASUSY.

Listing 9: Example HadronicRPV.sh. Shell script that produces an SLHA file using the ISASUSY program.

```
#!/bin/sh

FILENAME=${1}

DATFILE="${FILENAME}.dat"
SLHAFILE="${FILENAME}_temp.slha"
HERWIGFILE="${FILENAME}.hwg"

# Top mass pole
MTP="172"

# Gluino mass, Up-type quark mass, Pseudo-Scalar Higgs, and tangent beta
MGLSS=${3}
MU="3000"
MA="4000"
```

```
TANBETA="3"

MQ1="5000"
MDR=${2}
MUR=${2}
ML1="4000"
MER="300"

MQ3="5000"
MBR=${2}
MTR=${2}
ML3="4000"
MLR="300"
A_T="1000"
A_B="9000"
A_L="9000"

MQ2="/"
MSR=""
MCR=""
ML2=""
MMR=""

# Gaugino masses
M1="500"
M2="150"

# Gravitino mass
GRAVITINOMASS="/"

PARAM1="${MTP}"
PARAM2="${MGLSS},${MU},${MA},${TANBETA}"
PARAM3="${MQ1},${MDR},${MUR},${ML1},${MER}"
PARAM4="${MQ3},${MBR},${MTR},${ML3},${MLR},${A_T},${A_B},${A_L}"
PARAM5="${MQ2}${MSR}${MCR}${ML2}${MMR}"
PARAM6="${M1},${M2}"
PARAM7="${GRAVITINOMASS}"

# Creating SLHA files
./isasusy.x << EOF

'${DATFILE}'
${SLHAFILE}
${HERWIGFILE}
${PARAM1}
${PARAM2}
${PARAM3}
${PARAM4}
```

```
${PARAM5}
${PARAM6}
${PARAM7}

EOF

rm $DATFILE
rm $HERWIGFILE
```

Example of a shell script that produces an SLHA file using ISASUGRA.

Listing 10: Example mSUGRA.sh. Shell script that produces an SLHA file using the ISASUGRA program.

```
#!/bin/sh

FILENAME=${1}

DATFILE="${FILENAME}.dat"
SLHAFILE="${FILENAME}.slha"
HERWIGFILE="${FILENAME}.hwg"

# Squark mass, Gluino, A0, and tangent beta
MZERO=$2          # Mass of all scalar fermions at the Grand Unified Scale
MHALF=$3          # Mass of all gauginos at the Grand Unified Scale
A0=$4
TANBETA=$5
SIGNMU=$6

# Top mass pole
MTP=$7

PARAM1="1"
PARAM2="${MZERO},${MHALF},${A0},${TANBETA},${SIGNMU},${MTP}"

# Creating SLHA files
./isasugra.x << EOF

'${DATFILE}'
${SLHAFILE}
${HERWIGFILE}
${PARAM1}
${PARAM2}
EOF

rm $DATFILE
rm $HERWIGFILE

cat ISALHD.out | awk '{if(NR >= 15) print}' >> ${FILENAME}.slha
rm ISALHD.out
```

Example of an EDFilter code.

Listing 11: Example MultiLeptonFilter.cc.

```
// -*- C++ -*-
//
// Package:    MultiLeptonFilter
// Class:      MultiLeptonFilter
//
/**\class MultiLeptonFilter MultiLeptonFilter.cc RutgersGenFilter/MultiLeptonFil

 Description: [one line class summary]

 Implementation:
     [Notes on implementation]
*/
//
// Original Author:  Emmanuel Contreras-Campana
//         Created:  Wed Apr 25 17:18:58 EDT 2012
// $Id: MultiLeptonFilter.cc,v 1.2 2013/04/07 16:42:59 ecampana Exp $
//
//

// system include files
#include <memory>
#include <iomanip>
#include <iostream>

// user include files
#include "FWCore/Framework/interface/Frameworkfwd.h"
#include "FWCore/Framework/interface/EDFilter.h"
#include "FWCore/Framework/interface/Event.h"
#include "FWCore/Framework/interface/ESHandle.h"
#include "FWCore/Framework/interface/MakerMacros.h"
#include "FWCore/ParameterSet/interface/ParameterSet.h"
#include "SimGeneral/HepPDTRecord/interface/ParticleDataTable.h"
#include "DataFormats/HepMCCandidate/interface/GenParticle.h"
#include "DataFormats/Candidate/interface/Particle.h"


//
// class declaration
//
class MultiLeptonFilter : public edm::EDFilter
{
  public:
    explicit MultiLeptonFilter( const edm::ParameterSet & );
    ~MultiLeptonFilter();

    static void fillDescriptions( edm::ConfigurationDescriptions & );
```

```

  private:
    // ----------- member data ---------------------------
    virtual void beginJob();
    virtual bool filter( edm::Event &, const edm::EventSetup & );
    virtual void endJob();

    virtual bool beginRun( edm::Run &, edm::EventSetup const & );
    virtual bool endRun( edm::Run &, edm::EventSetup const & );
    virtual bool beginLuminosityBlock( edm::LuminosityBlock &, edm::EventSetup c
    virtual bool endLuminosityBlock( edm::LuminosityBlock &, edm::EventSetup con

    // ----------- member data ---------------------------
    edm::InputTag src_;

    bool m_edfilterOn;
    bool m_multlepFilterOn;
    bool m_mixModeFilterOn;

    int m_nLepton;

    bool m_debug;

    int nTotalEvents;
    int nEventsPassed;
    int nEventsPassedGt2Lep;
    int nEventsPassedGt3Lep;
    int nHZEventsPassed;

    int n0;
    int n1;
    int n2;
    int n3;
    int nGt4;
};

//
// constants, enums and typedefs
//

//
// static data member definitions
//

//
// constructors and destructor
//

```

```
MultiLeptonFilter::MultiLeptonFilter( const edm::ParameterSet & iPSet ):
  src_(iPSet.getParameter<edm::InputTag>("src")),
  m_edfilterOn(iPSet.getParameter<bool>("edfilterOn")),
  m_multlepFilterOn(iPSet.getParameter<bool>("multlepFilterOn")),
  m_mixModeFilterOn(iPSet.getParameter<bool>("mixModeFilterOn")),
  m_nLepton(iPSet.getParameter<int>("nLepton")),
  m_debug(iPSet.getParameter<bool>("debug"))
{
   // Now do what ever initialization is needed
   std::cout << "src_: " << src_ << std::endl;
   std::cout << "m_edfilterOn: " << m_edfilterOn << std::endl;
   std::cout << "m_multlepfilterOn: " << m_multlepFilterOn << std::endl;
   std::cout << "m_mixModeFilterOn: " << m_mixModeFilterOn << std::endl;
   std::cout << "m_nLepton: " << m_nLepton << std::endl;
   std::cout << "m_debug: " << m_debug << std::endl;

   nTotalEvents = 0;
   nEventsPassed = 0;
   nEventsPassedGt2Lep = 0;
   nEventsPassedGt3Lep = 0;
   nHZEventsPassed = 0;

   nGt4 = 0;
   n3 = 0;
   n2 = 0;
   n1 = 0;
   n0 = 0;
}

MultiLeptonFilter::~MultiLeptonFilter()
{
   // do anything here that needs to be done at desctruction time
   // (e.g. close files, deallocate resources etc.)
}

//
// member functions
//

// ------------ method called on each new Event ------------
bool MultiLeptonFilter::filter( edm::Event& iEvent, const edm::EventSetup& iSetu
{
  int nLepton = 0;
  int nLepton2 = 0;
  int nHiggs = 0;
  int nZboson = 0;
  int nHZ = 0;

```

```
bool pass = true;

// Gather information on the reco::GenParticle collection
edm::Handle<reco::GenParticleCollection> genParticles;
iEvent.getByLabel(src_, genParticles);

nTotalEvents++;

if (m_edfilterOn == true) {

  for (reco::GenParticleCollection::const_iterator iter = genParticles->begin(
       iter != genParticles->end(); ++iter) {

    // Verify whether particle comes from the decay of a Z boson
    try {
      if (iter != genParticles->begin() && iter != genParticles->begin()+1) {
        if ( abs( iter->mother()->pdgId() ) == 23 ) {
          if ( abs( iter->pdgId() ) == 11 || abs( iter->pdgId() ) == 13 || abs
            std::cout << "pdgID: " << iter->pdgId() << " status: " << iter->st
                      << " mother pdgID: " << abs( iter->mother()->pdgId() )
                      << " mother status: " << iter->mother()->status() << std

            // Throw exception if the particle that comes from the decay of a
            if ( iter->status() != 3 ) {
              throw 1;
            }
          }
        }
      }
    }

    // Catch exception that have been thrown
    catch (int e) {
      std::cout << "An exception occurred. Found lepton comming from Z boson w
    }

    // Search for status 3 particles
    if (iter->status() == 3) {
      // Search for electrons
      if ( abs( iter->pdgId() ) == 11 && iter->pt() > 5.0 ) {
        nLepton++;

        if ( abs( iter->pdgId() ) == 11 && iter->pt() > 10.0 ) {
          nLepton2++;
        }
      }

      // Search for muons
```

```
        else if ( abs( iter->pdgId() ) == 13 && iter->pt() > 5.0 ) {
          nLepton++;

          if ( abs( iter->pdgId() ) == 13 && iter->pt() > 10.0 ) {
            nLepton2++;
          }

        }

        // Search for taus
        else if ( abs( iter->pdgId() ) == 15 && iter->pt() > 5.0 ) {
          nLepton++;

          if ( abs( iter->pdgId() ) == 15 && iter->pt() > 10.0 ) {
            nLepton2++;
          }
        }
      } // iter->status() == 3

      // Search for Z bosons coming from neutralino
      if ( abs( iter->pdgId() ) == 23 && abs( iter->mother()->pdgId() ) == 10000
        nZboson++;
      }

      // Search for Higgs bosons coming from neutralino
      else if ( abs( iter->pdgId() ) == 25 && abs( iter->mother()->pdgId() ) ==
        nHiggs++;
      }
    } // end for-loop
  } // m_edfilterOn == true

  // Lepton multiplicity break down
  if (nLepton == 0) {
    n0++;
  }

  if (nLepton == 1) {
    n1++;
  }

  if (nLepton == 2) {
    n2++;
  }

  if (nLepton == 3) {
    n3++;
  }
```

```
  if (nLepton >= 4) {
    nGt4++;
  }

  // HZ multiplicity
  if (nHiggs == 1 && nZboson == 1) {
    nHZ++;
  }

  // Only multilepton filter is on
  if (nLepton < m_nLepton && m_multlepFilterOn && !m_mixModeFilterOn) {
    pass = false;

    if (m_debug == true) {
      std::cout << "Only multilepton filter is on!" << std::endl;
    }
  }

  // Only mix mode filter is on
  if (nHiggs != 1 && nZboson != 1 && m_mixModeFilterOn && !m_multlepFilterOn) {
    pass = false;

    if (m_debug == true) {
      std::cout << "Only mix mode filter is on!" << std::endl;
    }
  }

  // Both mulitlepton and mix mode filter is on
  if (nLepton < m_nLepton && m_multlepFilterOn && m_mixModeFilterOn) {
    pass = false;

    if (m_debug == true) {
      std::cout << "Both mulitlepton and mix mode filter is on!" << std::endl;
    }
  }

  else if (nHiggs != 1 && nZboson != 1 && m_multlepFilterOn && m_mixModeFilterOn
    pass = false;

    if (m_debug == true) {
      std::cout << "Both mulitlepton and mix mode filter is on!" << std::endl;
    }
  }

  if (pass) {
    nEventsPassed++;
  }
```

```
  if (nLepton >= 2) {
    nEventsPassedGt2Lep++;
  }

  if (nLepton2 >= 3) {
    nEventsPassedGt3Lep++;
  }

  if (nHZ > 0) {
    nHZEventsPassed++;
  }

  if (m_debug == true) {
    std::cout << "nLepton: " << nLepton << std::endl;
    std::cout << "nHiggs: " << nHiggs << std::endl;
    std::cout << "nZboson: " << nZboson << std::endl;

    if (pass == true) std::cout << "Event passed MultiLeptonFilter" << std::endl

    if (pass == false) std::cout << "Event failed MultiLeptonFilter" << std::end
  }

  //pass = false;
  return pass;
}

// ------------ method called once each job just before starting event loop
------------
void MultiLeptonFilter::beginJob()
{

}

// ------------ method called once each job just after ending the event loop
------------
void MultiLeptonFilter::endJob()
{
  std::cout << "Lepton multiplicity break down" << std::endl;
  std::cout << "nGt4: " << nGt4 << std::endl;
  std::cout << "n3: " << n3 << std::endl;
  std::cout << "n2: " << n2 << std::endl;
  std::cout << "n1: " << n1 << std::endl;
  std::cout << "n0: " << n0 << std::endl;
  std::cout << "nEventsPassedGt2Lep: " << nEventsPassedGt2Lep << std::endl;
  std::cout << "nEventsPassedGt3Lep: " << nEventsPassedGt3Lep << std::endl;
  std::cout << "nHZEventsPassed: " << nHZEventsPassed << std::endl;
  std::cout << "nEventsPassed: " << nEventsPassed << std::endl;
  std::cout << "nTotalEvents: " << nTotalEvents << std::endl;
```

```

  std::cout.setf(std::ios::fixed, std:: ios::floatfield);
  std::cout << "EDFilter MultiLepton efficiency: " << std::setprecision(6)
            << (double)nEventsPassed/(double)nTotalEvents << std::endl;
  std::cout.unsetf(std::ios::floatfield);
}

// ------------ method called when starting to processes a run ------------
bool MultiLeptonFilter::beginRun( edm::Run &iRun, edm::EventSetup const &iSetup
{
  return true;
}

// ------------ method called when ending the processing of a run ------------
bool MultiLeptonFilter::endRun( edm::Run &iRun, edm::EventSetup const &iSetup )
{
  return true;
}

// ------------ method called when starting to processes a luminosity block
------------
bool MultiLeptonFilter::beginLuminosityBlock( edm::LuminosityBlock &iLuminosityB
{
  return true;
}

// ------------ method called when ending the processing of a luminosity block
------------
bool MultiLeptonFilter::endLuminosityBlock( edm::LuminosityBlock &iLuminosityBlo
{
  return true;
}

// ------------ method fills 'descriptions' with the allowed parameters for the
------------
void MultiLeptonFilter::fillDescriptions( edm::ConfigurationDescriptions &descri
{
  // The following says we do not know what parameters are allowed so do no vali
  // Please change this to state exactly what you do use, even if it is no param

  edm::ParameterSetDescription desc;
  desc.setUnknown();
  descriptions.addDefault(desc);
}

// define this as a plug-in
DEFINE_FWK_MODULE(MultiLeptonFilter);
```