Dark Matter Searches at CMS at $\sqrt{s} = 13$ TeV

Searches for invisibly decaying Higgs bosons and semi-visible jets

By

Eshwen Bhal

School of Physics
University of Bristol

A lab book to keep track of research during my PhD.

Author details: Eshwen Bhal, University of Bristol, United Kingdom. **eshwen.bhal@bristol.ac.uk**, **eshwen.bhal@cern.ch**.

Supervisor details: Henning Flächer, University of Bristol, United Kingdom. **Henning.Flacher@cern.ch**, **Henning.Flaecher@bristol.ac.uk**.

23rd April 2020

Word count: number in words

Abstract

This document is an electronic copy of the lab book I'll be keeping during my PhD. It was initially intended both as a backup in case something happens to my physical lab book, and as a searchable document if I want to find a reference, etc., quickly. But now it has assumed the role of my primary lab book for various reasons. This contains everything I have worked on since the start of my PhD including all analysis, research, service work, and also code, commands and reminders of things I may forget. It is also date-logged (in part), fully searchable and hyper-referenced.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This is an electronic copy of my lab book that I will use for the duration of my PhD. It contains all notes, information and ideas pertaining to my PhD. Although it was originally designed to be both as a backup in case something happens to my physical lab book, and as a searchable document if I want to find a reference, etc., quickly, it has become my primary lab book because it is much easier to edit, include figures/files, and is easily accessible (as long as I can access my iCloud Drive). I start each new topic as a separate section, on a new page, and keep the document as closely-resembling as possible to my physical lab book (up to the point that I decided that this electronic version would be my primary lab book). The date, next to the title of each topic, is the date I *started* writing that section in my lab book. Any subsections which contain a date stamp in the title indicate that it was written well after the previous entries, but made sense to include it in that section. Most plots will probably be produced using ROOT, but for figures in presentations and publishable material (including my thesis) I may use some external software such as GraphPad Prism (as for my masters project) or Veusz.

1.1 Semantics and clarifications

To reference High Energy Physics (HEP) programs, I will use the `\textsc{}` command to display it in the "small caps" style, whereas normal GUI programs will be represented with the normal font. In-line code and commands with the Command Line Interface (CLI) will use the `\verb` command enclosed by the "!" symbol, so they are displayed in the "typewriter" font. If the command is too long for one line and runs into the margin of the page, use the `\texttt{}` command instead or enclose the code in an "lstlisting". If done within an "easylist", the command will be broken over two lines and

leave less vertical space compared to the spacing between each entry in the list. This way I can tell if the command is supposed to be entered as a single line or whether they are multiple commands. If the same applies for a URL, use `\sloppy\url{}`. To enclose something in quotes, I will just use " for both opening and closing quotations. To add an underscore in text, use `\_`, which is an alias I've created for the command `\texttt{\char'_}`. When including more than one command in a row, I combine them into an easylist or lstlisting rather than writing one command and then leaving a big line break between the next command. Combining them into a list tightens it up and distinguishes it. If I make my own command using `\newcommand`, when using it in text I must include a "\" afterward to force a space. For example, I have the command `\newcommand{\madgraph}{\textsc{MadGraph}}`. So I must write `\madgraph` in the text to force a space. I've created aliases for other commands and variables to make the typing more efficient. Some paragraphs are enclosed in a red-coloured box. These are potentially important things that apply to areas wider than the scope of the section/subsection it is written in. Another thing to note is the use of special characters of variables in section/subsection titles. These display correctly in the pdf, but in the bookmarks (list of contents when viewing in Adobe Reader, etc.) can't render them properly. But I can use the `\texorpdfstring{}{}` command to fix it. The first argument contains the special characters and the second is the plain text to display instead.

When including code, I also include the name of the file (with the version number in brackets) within the caption so that it's already declared if I reference it within the text. These files are stored in **./secX/**, the **X** referring to the section number in the lab book. The source code files that are included are exactly the same as the versions used to run their respective tasks, the only exception being comments. In some of these files, I've changed multi-line comments (which would otherwise exceed the 80-character guideline) to single-lined so that, when printed in the PDF, it looks better. When I include file names or directories in text (excluding captions), I will usually write them in bold font so that they look distinguished. An extra distinguishing feature is the use of colour boxes in conjunction with mini-pages. I normally use them to highlight text that is applicable to more general cases than the section it is included in, so I can find it more easily later on. For some reason, that environment doesn't like the `\verb` command, so I need to use `\texttt{}` instead.

For obvious reasons, I do not include work that I have crossed-out or erased in this electronic copy, but I will keep conceptual mistakes (e.g. if I refer to something that I later found to be incorrect) such that this work is representative of what I have done and reads as if I have been writing this as I progress with my PhD (which I shall, more-or-less, be doing). When referencing previous stuff in my physical lab book, I normally use "see page X". However, one page in my physical lab book wouldn't necessarily correspond to one page in this document, so, in this work, I'll give the appropriate object a label and then reference it with `\ref{}`.

Each section is given a number that matches its order in my lab book. Generally this would be sloppy practice but because this is an ongoing piece of work, I will not change the order of any of these sections because it wouldn't be representative of what a lab book is supposed to symbolise. Each of these sections is written in its own .tex file, with the file name including the section number as well as its title, and stored in **./modules/**. Then the path to each file is added to **labbook.tex** – which also handles the typesetting, packages, title and contents pages, and references – so it is included in the compilation. This way I can add and remove files during compilation for testing.

1.2 Terminal/bash shell commands

This list contains commands applicable to bash shells in UNIX systems. Unless stated otherwise, these commands are applicable to both macOS and Linux. This list will be updated regularly when I learn new terminal commands that I may otherwise forget.

- `cd <path>` – change directory relative to the current working directory. Can also use the absolute path.
- `ls` – list the files/folders in the current directory.
- `pwd` – print the working directory.
- `./` – point to the current directory.
- `../` – point to the directory above you.
- ∼ – point to the home directory.
- `<common path>*` – encompass objects that contain the path before the * character (also known as the "wildcard" character).
- `*<common ending>` – encompass all objects that end with <common ending>.
- Tab – auto-complete until an ambiguity arises.
- Tab → Tab – displays contents of directory/path before the command.
- `cp <file path> <destination path>` – copy and paste a file. Use `cp -r` (the `-r` stands for "recursive") to copy folders.
- `scp <local file path> <user>@<hostname>:<destination path>` – copy and paste a file to a remote server. Use `scp -r` to copy folders. Note that I must *not* be logged in to the server to copy the file. I can also copy a file from a remote server to my local machine, and between two remote servers.

- `ssh <user>@<hostname>` – to log in to a remote server. Note that I must be on the same network as the server, or be able to access it via a VPN.
- `mv <file/folder path> <destination path/new name>` – move a file/folder. It can also be used to rename a file/folder.
- `rm <file path>` – delete a file. Use `rm -rf` to delete a folder.
- `grep <expression> <file>` – search a plain text (includes code) file for an expression. Stands for ***g****lobally search a* ***re****gular expression and* ***p****rint*. Including `-i` ignores case, and `-r` searches recursively.
- `wget <URL>` – download a file from the Internet. Installed by default on Linux, can be used on macOS if Homebrew (and its `wget` package) is installed.
- `du -hs` – check the disk usage of the current directory and list the components.
- `find <directory> -name <file name>` – find the path to a file. Searches recursively by default, usually easier to set `<directory>` to ".". The exact file name must be given, otherwise use the `*` character to fill in the blanks.
- `<any command> -help` – get information on the use of a specific command, as well as the arguments that can be included.
- `{<element1>,<element2>,<etc>}` – brace expansion.
- Ctrl+A – go to the start of the current line.
- Ctrl+K – delete everything in the line after the cursor.
- Ctrl+R – reverse search for a command.
- Ctrl+Z – kill a currently-running process (if Ctrl+C doesn't work).
- $\leftarrow$ / $\rightarrow$-Ctrl – move left/right quicker (hold the arrow key first to begin moving, then press Ctrl).
- Ctrl+L – clear window.
- Ctrl-Tab or Cmd-Shift-{ or Cmd-<tab no.> – switch between currently open tabs.
- Cmd-` – switch between currently open windows.
- `say "<quote>"` – voice output from the terminal.
- `while true; do <command>; sleep <time interval>; done` – run a command periodically (e.g, checking on jobs/tasks at regular intervals).
- `$(<command>)` – output of command, i.e., command substitution. For example, I can set a variable to the output of a command.

- `${<variable>}` – use a variable in an expression (the braces are delimiters and only required when resolving ambiguities), i.e., variable substitution.

Emacs commands (for more, see reference card at https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf):

- `emacs -nw <file>` – disable the X11/awful GUI version of emacs when in an `ssh -X/-Y` session, and use the standard version.
- Ctrl-X+Ctrl-C+Y – save the file and exit.
- Ctr-A – go to the start of the current line.
- Ctrl-E – go the end of the current line.
- Ctrl-K – delete/cut everything in the line after the cursor.
- Ctrl-Y – paste (can be combined with Ctrl-K to cut and paste).
- Ctrl-U o Ctrl-K – delete/cut everything in the line *before* the cursor.
- Ctrl-S – search in the file.
- `if [[ ! -z "$DISPLAY" ]]; then alias emacs="emacs -nw"; fi` – add this to ∼/.bash_profile or ∼/.bashrc to disable the GUI version of emacs by default for the session if using X11 forwarding. Replaces the `emacs` command with `emacs -nw`.

Screen commands (for attaching and detaching ssh sessions):

- `screen` – start a screen session. Can be done within an ssh session.
- Ctrl-A+D – "detach" a screen session. So if I'm running a process that could take a while but need to leave (which would otherwise disconnect me and quit the process), I can type that command.
- `screen -r` – "re-attach" a screen session. So if I disconnect from a screen session above, then want to reconnect to check how a process is doing, I can type that command.
- `exit` or Ctrl-A+K or Ctrl-A → `:quit` – quit a screen session.
- Ctrl-A+[– enable scrolling.
- Adding `hardstatus alwayslastline '%= 9g[ %G%H %g][%= %= 9w%?%-Lw%?%=b 9R(%W%n*%f %t%?(%u)%?%=b 9R)%= 9w%?%+Lw%?%?%= %g][%Y%l%g]%=b C[ %d %M %c ]%W'` to ∼/**.screenrc** lists extra stuff at the bottom of the window.

Note that when starting a `screen`, any environments that were set up (e.g., CMSSW) are not carried over, so they need to be re-initialised.

Git commands:

- `git remote add <remote name> <url>` – add a remote with a link to the repository.

- `git fetch <remote>` – fetch updates from the remote.
- `git fetch <remote> -prune` – remove branches that have been deleted remotely, i.e., after a pull/merge request has been approved.
- `git pull <remote> <branch>` – add remote changes into branch.

1.3 Methods of communication and miscellaneous details

– University of Bristol details:
  Username – eb16003
  Student number – 1651526

– Email:
  Email address (Bristol) – eb16003@bristol.ac.uk | eshwen.bhal@bristol.ac.uk
  Email address (CERN) – eshwen.bhal@cern.ch

– CERN:
  See Sec. 9

– Skype (correspondence with academics and groups):
  Skype Name – eshwen.bhal

– Slack (correspondence with RA1):
  Email address – eshwen.bhal@bristol.ac.uk
  Team – alphatteam.slack.com

– Mattermost (Slack alternative, correspondence with FAST and Common Analysis):
  Login – <same as CERN login>
  Server URL – https://mattermost.web.cern.ch/

– GitHub (code and repositories):
  Username – eshwen
  Email address – eshwen.bhal@bristol.ac.uk

– GitLab (code and repositories, CERN's version of GitHub):
  Username – ebhal
  Email address – eshwen.bhal@cern.ch

– Dropbox (sharing lab book and other files with Bjoern):
  Email address – eshwen.bhal@bristol.ac.uk

– Evernote (to-do list for PhD work and uploading plots for L1 Jet Energy Corrections):
  Email address – eb16003@bristol.ac.uk

– Indico (hub for CERN-related meetings and conferences):
  Login details – <CERN login>

– Vidyo (CERN-related video meetings):
  VidyoPortal: https://vidyoportal.cern.ch
  Username: ebhal

– Mendeley (organise papers and articles I've downloaded):
  Email address: eshwen.bhal@bristol.ac.uk

– P5/cmsusr/.CMS account (for Trigger monitoring and maintenance):
  <username>@<hostname>: ebhal@cmsusr.cern.ch

– Ian Allan Travel (travel booking):
  Member ID/email address – eshwen.bhal@bristol.ac.uk
  Grant/budget code – SK1606 (legacy), S112751-102 (new, for ERP), S113321-101 (LTA)

– Claiming expenses:
  See https://www.bris.ac.uk/my-erp-support/how-to-guides/expenses/

– UK PP Travel Notices (for submitting travel notices for any foreign travel and long-distance UK travel):
  Website – https://pprc.qmul.ac.uk/~lloyd/travel/?page=home
  Email address – eshwen.bhal@cern.ch

– Library PIN:
  6649

– ORCID (Open Researcher and Contributer ID):
  ID – orcid.org/0000-0003-4494-628X
  Email address – eb16003@bristol.ac.uk

– CERN car sharing (with Mobility, see https://my.mobility.ch/login to reserve a car):
  Mobility number – 1004470
  PIN code – 548169

– UCU (University and College Union):
  Email address – eshwen.bhal@bristol.ac.uk

– Proceedings of Science:
  Username – bhalesh (or eshwen.bhal@cern.ch)

– Researchfish:
  Username – eshwen (or eshwen.bhal@bristol.ac.uk)

– Remote servers I have access to:

Soolin (Bristol): eb16003@soolin.dice.priv ← must be on the University's network to log in (or use proxy, either uses my normal UoB password)

LXPLUS (CERN): ebhal@lxplus.cern.ch ← uses same password as email

lxoo (Imperial College London): ebhal@lxoo.hep.ph.ic.ac.uk

lxo1 (Imperial College London): ebhal@lxo1.hep.ph.ic.ac.uk

cmsusr (CERN, P5): ebhal@cmsusr.cern.ch

– Python package managers I use:

Homebrew (locally). To update packages, use `brew update`.

PyPI via pip (locally, Soolin). To update all packages, use `pip list -outdated | cut -d' ' -f1 | xargs pip install -upgrade -user`.

Unless otherwise stated, my passwords should be locked up with LastPass. And I can refer there if I forget any of them.

Update: I was sent useful link about common pitfalls when thesis writing – https://www.scribd.com/document/401311835/Common-Gotchas-in-HEP-Thesis-Writing. This will probably be handy when it comes to writing up, so is worth documenting here. The University also has guidelines for thesis formatting at http://www.bristol.ac.uk/academic-quality/pg/pgrcode/annex4/. CMS' guidelines for authors is probably also worth a read – https://twiki.cern.ch/twiki/bin/viewauth/CMS/Internal/PubGuidelines. While I won't need to strictly adhere to them when writing my thesis, there are some good recommendations regarding spelling, grammar and punctuation.

Chapter 2

Introduction, software/hardware needs, stuff to remember (26/09/2016)

– Generally, be in the physics building 9–5 on weekdays.

– Bank holidays off.

– $\sim$ 4 weeks of leave.

– Should be getting my Macbook Air soon, then I can get up and running with proper work.

– In the meantime, install a Linux virtual machine (using VirtualBox) so I can use the computing software with my Fujitsu laptop.

Software to install on Linux VM/Mac:

ROOT

MadGraph

pythia

For postgrad lecture courses I'll need to enroll in, see http://mpags.nottingham.ac.uk/mpags (username: eshwen, password: <normal password>). I need to enroll in 4 modules this term and 2 next term $\rightarrow$ statistics, C++, particle physics modules, quantum field theory?

– Need to do some service work (Level 1 Trigger, most likely) and contribute so I can make the authors list.

– Will need to be proficient in C++ (so re-familiarise myself with C) and Python (scripting) because these are the languages used in the software I'll be using (ROOT, etc.).

– Need to get registered with CERN. - Will normally be in room 4.46 (postgrad office). Access code = 412.

So I've installed Linux Mint 18 (Cinammon) as a virtual machine on my Fujitsu after numerous failed attempts with other operating systems. But now it won't connect to the Internet to fetch updates for drivers and codecs, etc. And without internet, I can't install the necessary programs. So get the internet connection sorted!

^^^Not worth the hassle of setting up a VM. I've just installed Cygwin for the time being, so I can use PYTHIA and ROOT, etc.

For the postgrad lecture courses, it's recommended that I take:

– PP1: Introduction to Particle Physics

– PP3: QED and the Standard Model

– PP5: Group Theory (for Particle Physics)

– PP6: Particle Physics Techniques

The times of the lectures are on the webpage (see page 2/3), along with the lecture notes and other material.

The lectures are livestreamed in the lecture room 3.29 (opposite Martisse's office).

Statistics course (also recommended I take) in term 2. Takes place 9^{th}–13^{th} Jan. 2017, 11am–1pm in room 3.29.

For teaching/demonstrating, I'm doing first year maths tutorials in term 2 (weeks 13–24). Should be paid £12.22 per hour. Contract should be issued around Christmas; will need my National Insurance number and passport.

Remember I also have to attend lectures for PHYSM3407 Current Topics in Physics (starts Tuesday 1^{st} November), focusing on the particle physics strand. I won't have to do any assessments, just attend lectures. It runs from weeks 6–11.

Guidelines for CMS publications (font, writing style, spelling, and any other subtleties) are clarified at https://twiki.cern.ch/twiki/bin/viewauth/CMS/Internal/PubGuidelines.

Chapter 3

Useful introductory papers for Supersymmetry and Dark Matter (27/09/2016)

[1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

[9]

[10]

[11]

[12]

[13]

Chapter 4

Using pythia with Cygwin (on Windows) (27/09/2016)

– Make sure Cygwin is installed (with all the "devel" packages for `make` commands, etc.).

– Unpack the PYTHIA tar file in the **Cygwin/home/** directory.

– Follow the README in the **pythia8219/** directory to make the programs, etc.

– Follow the example in the **examples/** folder (using the README in the folder as a guide).

– Generally, just enter into the Cygwin terminal `make mainNN` (where `NN` is the two-digit number after the word `main` in the source files), then type `./mainNN > output`.

– Then look at the "output" file to get an idea of how the output will generally look when running scripts, etc. I can also look at the source files and the scripts to see what their objectives are.

In the command `./mainNN > output`, the name of the file "output" isn't necessarily fixed. I can change the name of the output file to whatever I want. It just makes it easier to keep track of different simulations/programs I run.

Chapter 5

Interesting thoughts and interpretations of dark matter (29/09/2016)

– Thought to be made of a neutral, weakly interacting particle. Because it's neutral (and thought to be elementary), it cannot couple to the electromagnetic field, so there's no way to detect it through conventional astronomy.

– Galaxies form and condense in dark matter wells because the baryonic matter does not produce a gravitational field strong enough to keep ahold of ordinary matter.

– Dark matter is not super dense (like a black hole around the centre of a galaxy). It is more "sprinkled" in a roughly-spherical halo throughout a galaxy. [14], [15] Its average density is still much greater than that of the normal matter in the galaxy because it's the dark matter that leads to the flat rotation curve of most galaxies. [16] If there were only baryonic matter, the rotation curve would drop off as $\propto 1/\sqrt{r}$ as Kepler's laws state. Whilst the surface density of dark matter does decrease with radius – so the density in a thin spherical shell around the galaxy decreases – its inclusive mass increases linearly (see Figure 1 in [17]) to compensate for the Keplerian drop off in the baryonic matter (think Gauss' law). [18], [19] The dark matter "orbits" the centre of the galaxy with the stars, but in some models has a lower velocity because it's angular momentum doesn't dissipate via collisions and collapse into a disc like baryonic matter does (because collisions would result in annihilation). In other models, it's corotational with the stars.

– Dark matter particles must be stable because they've existed for billions of years and have allowed galaxies to form in their potential wells.

– It is thought that dark matter was produced thermally in the early universe (when it was hot and therefore easier to create heavy particles). Dark matter particles could (and did) annihilate, but as the universe expanded and cooled the dark matter became more diffuse. They didn't annihilate as often and the dark matter we see today is a "thermal relic" and is what's left over. Some models suggest the parameter $x = m_{\mathrm{DM}}/T \sim 20$ at freeze out. [20]

– The current leading dark matter candidate is a WIMP (Weakly Interacting Massive Particle), possibly a neutral supersymmetric particle (neutralino) like a higgsino, photino, zino, etc. One of the motivations for WIMPs are that, using the current values of the dark matter density in the universe and approximations for the annihilation cross section, dark matter could self-interact at the electroweak scale to produce Standard Model particles. [21] As the EW scale can be readily accessed at colliders like the LHC, we could detect DM signatures via pair production then annihilation or indirect searches via solely annihilation.

– WIMP annihilation could produce two showers of quarks, which would normally be observed as pions and high energy photons (like gamma rays). The photons may be of a continuum – from hadronisation and radiation of the decay products of annihilation – or contain features (internal radiation from the propagator in the interaction or from loop-level processes).

– Because WIMPs are stable (at least on the timescale of the current age of the Universe), they wouldn't decay into other particles when produced from accelerator collisions. So you would detect them (indirectly) by looking for missing transverse momentum ($\not{p}_{\mathrm{T}}$ or $p_{\mathrm{T}}^{\mathrm{miss}}$) or missing transverse energy ($\not{E}_{\mathrm{T}}$ or $E_{\mathrm{T}}^{\mathrm{miss}}$) and by looking for visible particles recoiling against the WIMPs.

– The mediator (force-carrying particle, like the gauge bosons) for dark matter – between dark matter particles or the dark matter-Standard Model particle interactions – may be a scalar (spin-0, like the Higgs boson) or pseudoscalar (reverses parity under a Lorentz transformation, like the pion) boson. [Support] for a pseudoscalar over a scalar mediator comes from the Feynman diagrams for DM annihilation into, e.g., b-quarks. With a scalar mediator, the vertex factors and the propagator term lead to cancellations in the cross section equation in the low-velocity limit.

– The mediator for dark matter may be heavier than the dark matter particle itself (like with the $W^{\pm}$ and Z bosons being heavier than most of the particles they mediate), maybe 2x heavier or more so than the DM particle. The mediator could decay via DM pair production, so it makes sense that it would be at least twice as heavy.

– There's no consensus on whether dark matter is fermionic or bosonic. If fermionic, it may be

either a Dirac fermion (particle is distinct from its antiparticle, like the electron and positron) or Majorana fermion (particle is the same as its antiparticle, like the neutrino is suspected to be). If it were Majorana, dark matter could annihilate with itself, making discoveries via indirect searches more likely.

– At the LHC, monojets are used most prominently to look for dark matter particles. But multijet plus $E_{\mathrm{T}}^{\mathrm{miss}}$ might provide better sensitivity and constraints (particularly if the mediator is pseudoscalar).

– The Coma Cluster of galaxies seems to contain a very high concentration of dark matter (mass-to-light ratio of 400 $M_{\odot}/L_{\odot}$). See [22].

– Many dark matter candidates include a few supersymmetric particles (the neutralino being the most widely studied), sterile neutrinos [23], axions, Kaluza-Klein states [24] (which are excitations of Standard Model fields in extra dimensions), etc.

– Dark matter has to be cold (non-relativistic), as opposed to hot (relativistic), implying dark matter particles are reasonably heavy. If dark matter was light, it would have a lot of energy when produced in the early universe and would be relativistic (so hot). But if it were hot, it would be too diffusive to allow galaxies to form. However, because of the seemingly bottom-up nature of structure formation in the Universe [25] (smaller galaxies form first around clumps of dark matter, then orbit and merge with other galaxies to form clusters and larger elliptical galaxies), dark matter must be cold so it doesn't diffuse too much and can clump to allow galaxy formation.

– The different aspects of dark matter searches:

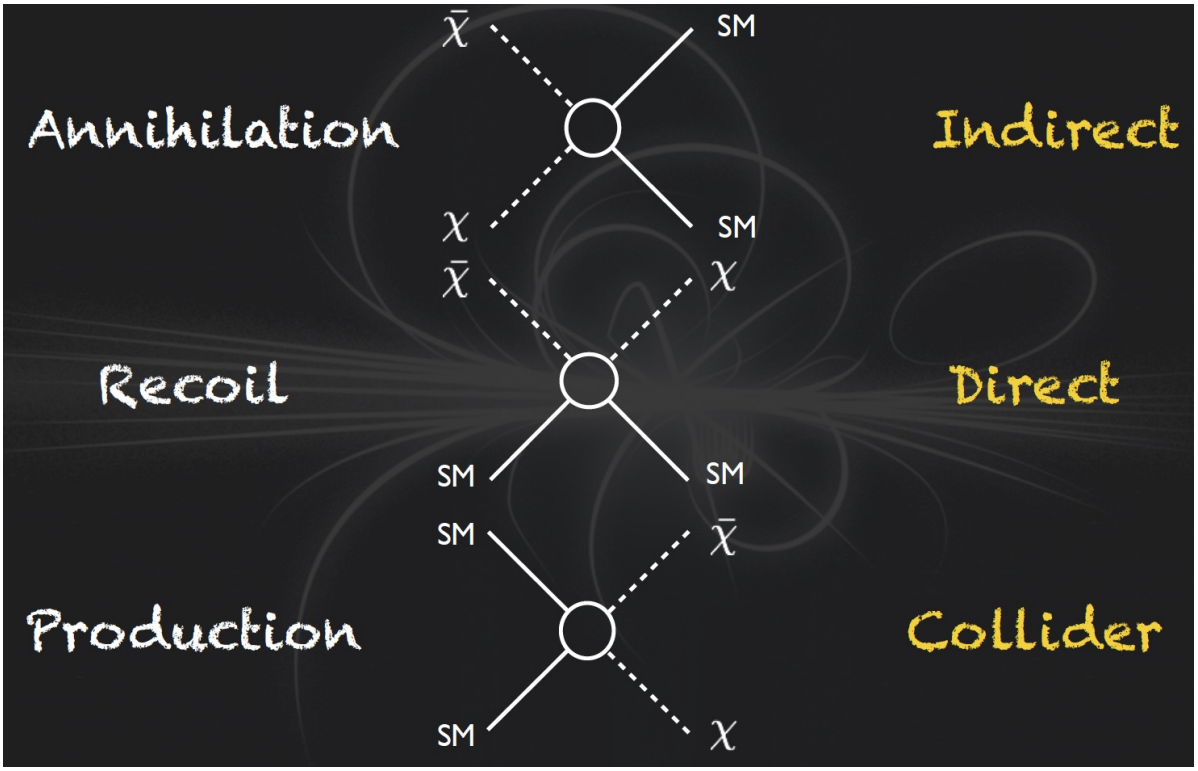

Figure 5.1: The different aspects of dark matter searches: indirect (annihilation), direct (scattering from SM particles), and collider (production). Taken from Bjoern's University of Wisconsin – Madison talk, 27/03/2017.

– Some good results showcasing dark matter masses and that of its mediator from different analyses and decay channels are at [26]. Particularly figures 4 and 6, which I used in my poster for the PGR conference.

– The 2015 results from Planck estimate the dark matter content of the universe to be 25.8%. It displays it in terms of $\Omega_c h^2 = 0.1186$, where $h = 0.678$ (the Hubble constant, in units of km s^{-1} $Mpc^{-1}/100$), giving $\Omega_c = 0.258$ as the cold dark matter density [27].

– Dark matter cannot be solely neutrinos because the flux densities of neutrinos (from stars, as well as the cosmic neutrino background [28]) are precise and well-known, and due to the upper limit on neutrino masses [29], are too small to account for the dark matter content in the universe. Because the neutrinos have such a small mass, they would be highly relativistic in the early universe (as they are today, despite it being cooler now, making them slower) and so could only contribute to hot dark matter [30]. But as experiments show, the vast majority of dark matter must be cold.

– MOND (Modified Newtonian Dynamics) is one theory that tries to explain dark matter, and

can be constrained to explain galactic rotation curves and other astrophysical phenomena attributed to it. However, any certain strand that tries to explain one observation usually falls flat when trying to explain others. It also doesn't work at all the scales to which we can observe the effects of General Relativity, almost confirming that GR is the correct description of gravity and MOND is a failure.

- LUX (Large Underground Xenon experiment) and LZ (LUX-Zeplin) are direct detection experiments that search explicitly for WIMP dark matter. LUX uses an underground liquid xenon tank to detect WIMPs interacting with ordinary matter, the scattering producing photons and electrons of specific energies. LZ is a collaboration between the LUX and ZEPLIN groups, and will have a highly-sensitive WIMP detector over a large range of masses, once completed.
- Evidence for non-luminous, *non-baryonic* matter (an argument for those who ask why dark matter can't be neutrons, etc.):
  - One can use Doppler shifts of light emitted from galaxies in clusters, and therefore determine their masses. Then using the mass-to-light ratios of these galaxies and clusters (e.g., Bullet Cluster [31]), one can determine that most of the mass comes from non-luminous matter [32].
  - One can also use the Cosmic Microwave Background to calculate the average photon and neutrino (mass/energy) densities and Big Bang Nucleosynthesis calculations to determine the baryonic matter density. These can be compared to other measurements (e.g, mass-to-light ratios averaged across the universe) and reveal the discrepancy [32].
  - Neutrons can't contribute to dark matter because isolated neutrons are unstable, decaying in a matter of minutes [33]. They decay into protons and electrons. Being charged, they interact strongly with light and therefore contribute to the luminous matter in the universe.

Chapter 6

Running root in a Linux VM (30/09/2016)

I eventually got the Internet to work on the Linux VM on my Fujitsu laptop. I rebooted the network whilst inside the VM and that seemed to do the trick. So I downloaded ROOT (v5.34/36 because that was the latest version compatible with my VM's architecture), unpacked it and followed the instructions in the **root/README/INSTALL** file. It took a while to configure (using the command `./configure linux` in the terminal) because there were several packages missing that I had to install (hopefully I won't have to go through that again when I try installing it on the Macbook Air). Then after configuring, I had to make the program (just running the `make` command in the terminal) which took several hours but eventually finished. To run it, navigate to the directory containing ROOT (**../root/**) and run the command `root` in the terminal. All commands within ROOT must be preceded by a ".", including the exit command (either `.q` or `.exit`) and should be written as C++ code.

When I want to run ROOT, navigate to the /root folder using the terminal, then type `source bin/thisroot.sh`, then I can run the program by typing `root`.

In my Linux VM, open the terminal then type:

```
cd root
```

```
source bin/thisroot.sh
```

```
root
```

Input is case sensitive.

To open a TTree (an object containing a list of branches – where data and structures, etc., are stored), type

```
TFile::Open("<path to, and name of TTree(.root)>");
```

To see what's inside the .root file, type

```
.ls
```

To access the ROOT Browser (like a GUI), type

```
new TBrowser()
```

and you can access the branches and stuff within a tree. To draw (display) data from the terminal, without the Browser, type

```
<name of tree within .root file>->Draw("<name of thing to
    display>")
```

When creating macros (mini programs to analyse the data in a tree), saving them as a **.c** file makes it a C source file, whereas saving it as a **.C** file make it a C++ source file, which is usually what you want.

To open (any) files in a terminal which contain spaces in their names, enclose the path and name of the file in quotation marks, e.g., `open "this thing.png"`.

If I want to see the contents of a tree from a .root file, go into ROOT and type the commands

```
TFile f("<name of .root file>")
<Name of tree>->MakeClass("<name of your choosing>")
```

then a .C and .h file will be created in the current directory. Then open the .h file to see the names of the branches and leaves, so you know the syntax for, e.g., getting branches and leaves to fill histograms.

Many ROOT histograms seen in papers have the y-axis label as "<some variable or number of events>/<some value>". The <some value> is usually the width of the bin in the x-axis. So, for example, if I'm looking at a histogram with the number of events vs. $E_{\mathrm{T}}^{\mathrm{miss}}$, the y-axis label might read "events/1 GeV". So this will show the number of events with the histogram bins being 1 GeV wide in $E_{\mathrm{T}}^{\mathrm{miss}}$.

From now, until I state otherwise, I will be using ROOT v5.34/36.

Chapter 7

Using Soolin and DICE (05/10/2016)

Soolin is a computer that can be accessed remotely (via ssh) to, e.g., submit jobs or compile and use ROOT (or other software). A handy link is https://github.com/uobdic/Bristol-PHYS14. To log in to Soolin, open a terminal (normal terminal on a Mac/Linux machine, Cygwin on Windows), and type

```
ssh -Y eb16003@soolin.phy.bris.ac.uk
```

then enter my password (normal University of Bristol password) and I can use Soolin. To exit, just type `exit`. Instead of `-Y`, I can use `-X` which opens an X11 connection and allows GUIs to a degree.

You can change directories (and back out of your home directory to go into others'/tmp, etc.), copy files (`cp <path to file to copy> <destination path>`), and everything else you can normally do in a terminal.

ROOT should already be installed on Soolin. To access it, see page 18/later this section.

DICE is the server/cluster that jobs can be submitted to, and is accessed via Soolin (i.e., when you're doing stuff on Soolin, it is using the computing resources of DICE).

To copy files from my local machine to Soolin, type

```
scp <path to file> eb16003@soolin.phy.bris.ac.uk:<path>
```

and swap the commands to copy Soolin → local. To copy a folder, use `scp -r ...`. To *move* a file, folder or directory use `mv <object to move> <destination>`. This command can also be used to rename files/folders by typing `mv <original name> <new name>`.

UPDATE: there's a new, upgraded version of Soolin. It works like the previous version. I just log on with eb16003@soolin.dice.priv now, and everything else works in the same way.

7.1 Getting ROOT on Soolin

A way to get ROOT on Soolin: log in to Soolin, then type

`source /cvmfs/cms.cern.ch/cmsset_default.sh`

`cmsrel CMSSW_7_1_5` ← don't need this command after I've done it once

`cd CMSSW_7_1_5/src`

`cmsenv`

`cd ~`

These commands set up a ROOT environment for me on Soolin. Once I've done that I can use MadGraph with `pythia=ON` and `pgs=ON` to generate .root files that I can analyse in ROOT. To do this, navigate to the directory (the output directory, in this case **sm_test_ppXll/Events/run_01**). Then type

`root tag_1_pgs_events.root`

and then I can inspect and modify the output from the command line or with TBrowser. If the command `new TBrowser()` doesn't work or – when starting ROOT– I get a warning "DISPLAY not set...", install XQuartz, then log off and log back in. Then boot up my normal terminal and try again.

Also, if I get a "DISPLAY not set..." warning, try logging out of Soolin and then reconnecting using `ssh -Y ...` or `ssh -X ...` rather than just `ssh ...`.

The commands used to initialise ROOT have been added to my bash profile on Soolin so that they automatically execute when I log in to Soolin. To edit my bash profile (to execute different commands upon logging in), I need to edit – with emacs/vi – `~/.bashrc`. (The tilde just points to my home directory.)

I can also preload commands to execute automatically when I start ROOT (by creating and editing **rootlogon.C**), then creating a .rootrc file and pointing to the **rootlogon.C** file (see https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookSetComputerNode).

7.2 Generating an ssh key (30/11/2016)

I can `ssh` into Soolin without having to enter my password every time. I just need to enter these commands once (and I'm including them in case something happens to my computer and have to do

it again):

`ssh-keygen -t rsa`

(Keep hitting the "enter" key until it prompts me for my Soolin password, and enter the password.)

`ssh-copy-id -i ~/.ssh/id_rsa.pub eb16003@soolin.dice.priv`

I can also shorten the `ssh` procedure even further. If I'm in my local home directory, type:

`emacs ~/.ssh/config`

Then type

```
Host soolin
  HostName soolin.dice.priv
  User eb16003
  ForwardX11 yes
  ForwardX11Trusted yes
```

then Ctrl-X + Ctrl-C to exit. And then I should just need to type `ssh soolin`, and I should be logged in automatically. If I want to store multiple ssh keys, I can rename the **~/.ssh/id_rsa** and **~/.ssh/id_rsa.pub** files to whatever I want (e.g., **soolin_rsa** and **soolin_rsa.pub**, respectively) and add the following line to the **~/.ssh/config** file:

`IdentityFile ~/.ssh/soolin_rsa`

7.3 Accessing Soolin from outside Eduroam (13/12/2016)

Normally for `ssh`ing into remote servers you need to be on the same network (in the case of Soolin, you need to be on Eduroam on the University of Bristol network). It is, however, possible to `ssh` into Soolin outside of the UoB network. On my Macbook Air, there is already an app called Junos Pulse (which is a VPN) that has everything configured. I just need to hit "connect" and I'll be on the network. On my Macbook Pro, I can download the later iteration called Pulse Secure (https://www.bris.ac.uk/it-services/advice/homeusers/uobonly/uobvpn/howto/osx/). Then I just need to follow the instructions on the webpage to set up the connection. As with Junos Pulse, hit "connect" and I'll be on the University of Bristol network. Then I can just `ssh` into Soolin like normal and do everything else that I can normally do on Soolin.

In case I need to install the app again and there are issues with the webpage or some other problem, the steps are outlined below:

1. Download and install the Pulse Secure app. I should be able to find a .dmg file somewhere under the IT services section on the University of Bristol website (https://www.bris.ac.uk/it-services/). Failing that, ask Winnie.

2. Open the app, then click the "+" icon to add a new connection.
3. Leave Type as "Policy Secure (UAC)" or "Connect Secure (VPN)".
4. In the Name box, type "University of Bristol".
5. In the Server URL box, type "uobnet.bris.ac.uk".
6. Then click "Add" to add the connection.
7. Enter my University of Bristol username and password when prompted.
8. Then just click "connect" and I'm good to go.

I've now replaced Junos Pulse with Pulse Secure on my Macbook Air as it's more recent and still supported.

If I don't want to use the VPN to access Soolin, e.g., if I'm at CERN where using a VPN can block CERN services, I can add the following to my ssh config:

```
Host soolin_proxy
  HostName soolin.dice.priv
  ForwardAgent yes
  ProxyCommand ssh -XY -q eb16003@seis.bris.ac.uk nc %h 22
  User eb16003
```

and do `ssh soolin_proxy`.

Chapter 8

Using MadGraph (first steps) (07/10/2016)

MadGraph [34] is software used to run different physics processes (namely particle collisions). There is a version on the web which has a GUI (of sorts) – `madgraph.physics.illinois.edu` – but requires extra files to be uploaded that you have to create/edit from a template.

Generally MadGraph– more specifically, MadGraph 5 – will be used on my local machine or accessed on Soolin and is used from the command line. In my Soolin directory I have installed it and I also have the tar file if I want to unpack and install it again. From now, until I state otherwise, I will be using MadGraph 5 v2.4.3.

Note that if I use results – when writing a paper/my thesis – that were generated in MadGraph, MadAnalysis, root or any other programs (including extra packages for them you can install), I will need to appropriately reference them. When starting the programs (or in the respective READMEs), there should be information on how to reference them.

To start off with, navigate to the **/mg** directory. For now I'll just be playing with the test file (**sm_test.dat**), but will presumably be able to create my own in the future. Type

```
emacs sm_test.dat
```

to bring up the text editor (emacs) and allow me to to edit the file. To exit the editor, use Ctrl-X + Ctrl-C and hit y to save. To comment stuff out use the # symbol. I can set parameters using the `set` command, e.g., `set misset 60` to set the lower limit on $E_{\mathrm{T}}^{\mathrm{miss}}$ to 60 GeV, `set ebeam1 2000` to change the single-particle energy of beam 1 to 2000 GeV, etc.

If not units are given for values, or when setting/editing them, energy/mass/momentum are assumed to be in GeV and cross section (σ) is assumed to be in picobarns (pb, 1 pb $= 10^{-40}$ m^2).

To run MadGraph, type

`./bin/mg5 sm_test.dat`

and the program will run. The main bits of the output – total cross section, number of events – will be displayed toward the end. It also gives more information in **/users/eb16003/MadGraph/mg/sm_test_met60/index.html**, which gives the input and output in a pretty detailed fashion. I'm not sure how to open it on Cygwin, though it may be able to open on a mac.

There are other text editors on Soolin, like pico. It's probably best to brave it and just learn to use emacs (or "vi" – as of writing, the new Soolin only has vi, and not emacs). If I get stuck, check the Internet for help. Typing

`cat sm_test.dat`

shows some preamble and definitions, such as the labels for particles, which model is being used (Standard Model in this case), the process being described, etc.

Play around with MadGraph, edit parameters (beam energy, $E_{\mathrm{T}}^{\mathrm{miss}}$, etc.) and see how the results change.

I can change the name of the output file (the path shown above is **.../mg/<output file>/index.html**) by looking for the `output` statement toward the end of the .dat file.

After the `launch` statement near the end of the file, I can change certain variables to see what the outcome is like. For example, there's `set`

`misset <number>` – set lower limit on $E_{\mathrm{T}}^{\mathrm{miss}}$ to <number> GeV

`ebeam1 <number>` – set energy of protons in beam 1 to <number> GeV

`ebeam2 <number>` – set energy of protons in beam 2 to <number> GeV

`ptl <number>` – set lower limit on lepton momentum to <number> GeV

The lower limit on a variable is also referred to as a "cut".

To test out the effects of different variables, I'll edit them in **sm_test.dat** and see how the cross section changes. I can't get any more information than that because Cygwin won't let me (nicely) open the hmtl file.

To find out the names of different variables and the parameters that are used in the simulations, navigate to **mg/<output destination>/Cards** and open **run_card.dat**. It shows the different variable names with explanations and the other parameters used in the program. Note that this is just for reference; if I want to edit these parameters, do so in the **sm_test.dat** file.

If I want the html file to open automatically after finishing a run, I would have to run MadGraph locally as there is no browser on Soolin that MadGraph can point to in order to open the file.

Varying lepton momentum (ptl):

ptl (GeV)	Cross section (pb)
0	2.328×10^4
10	843.6
20	639.8
40	322.6
60	4.678
100	0.7843
200	0.07772

Table 8.1: Cross section versus lepton momentum for $pp \to e^+e^-$ in MadGraph.

Varying $E_{\mathrm{T}}^{\mathrm{miss}}$ (misset):

misset (GeV)	Cross section (pb)
0	843.6
10	843.6
20	843.6
40	691.2
60	657.0
100	642.6
200	844.5

Table 8.2: Cross section versus missing transverse energy for $pp \to e^+e^-$ in MadGraph.

Varying beam energy (ebeam1 and ebeam2):

ebeam1, ebeam2 (GeV)	Cross section (pb)
1000, 1000	154.4
2000, 2000	303.6
4000, 4000	556.3
10 000, 10 000	1217
15 000, 15 000	1717

Table 8.3: Cross section versus missing beam energy for $pp \to e^+e^-$ in MadGraph.

For all of these, every other variable was kept the same (10 000 events, etc.). The process was $pp \to e^+e^-$ @ 1$^{\mathrm{st}}$ order.

With the current build of MadGraph I have on Soolin, I can also run pythia (this build can install Pythia6, [35]), Delphes [36] (detector simulation program), and ExRootAnalysis

(convert output files into .root containers). In the **sm_test.dat** file, after the `launch` statement, include

`pythia=ON`

`delphes=ON`

etc., but I have to make sure they are installed. To install these packages, go to the **/mg** directory and type `./bin/mg5` and then type `install <program>`. If you mistype or spell it wrong, etc., a warning should appear to show you the correct syntax and the names of all the programs you can install.

To get time information (how long a particular simulation takes), add `time` before `./bin/mg5` when I run a simulation, i.e., `time ./bin/mg5 <input file>`.

The .lhe.gz output files, once converted to a .root file, contain a TTree called LHEF. One of the branches is named Particle.PID which gives information about which particles are in event. By selecting a certain particle ID you can plot the kinematic distributions for each particle and create other important variables, e.g., H_{T} (scalar sum of the jet momenta), M_{T} (transverse mass, that allows you to distinguish e.g. Ws), or the angle between the $E_{\mathrm{T}}^{\mathrm{miss}}$ and lepton/jet. More information on the particle IDs can be found at http://pdg.lbl.gov/2007/reviews/montecarlorpp.pdf.

Chapter 9

CERN Account (14/10/2016)

After some work, I've managed to get my CERN account up and running. Details are below.

Name: Eshwen Bhal

Login/Username: ebhal

Email address: eshwen.bhal@bristol.ac.uk

CERN email address: eshwen.bhal@cern.ch

Password (for login, EDH, mail): see Sec. 1 about cipher

Hypernews User ID: ebhal

Hypernews password: cernhypernews

To get my CERN email account on an android device, I just have to enter the email address and password when prompted. To get it on the Mail app on macOS, I can add it as an Exchange account. Some useful links (which I've also bookmarked in Firefox) are

Account Management: https://account.cern.ch/account/

Email: https://mmm.cern.ch/owa/

Android Community: https://android.web.cern.ch/

I've also subscribed to the CMS HyperNews System https://hypernews.cern.ch/HyperNews/CMS/top.pl which will keep me up to date with advances in the field and news related to the forums I'm subscribed to (Exotica and Exotica-MET+X, at the moment). The twiki page

at CERN is https://twiki.cern.ch/twiki/bin/view/CMS/EXOTICA. To sign up initially to HyperNews I have to go to the terminal and type `ssh ebhal@lxplus.cern.ch` to log in to CERN's LXPLUS cluster (their Soolin equivalent), then `ssh hypernews.cern.ch` and set up a username and password. My username is just `ebhal` and my password is `cernhypernews`. Mailing lists that I'm subscribed to can be found on HyperNews and on https://e-groups.cern.ch/. If I end up with a constant barrage of emails, look through my subscriptions and unsubscribe to irrelevant lists. With a CERN account, I also get access to Indico (https://indico.cern.ch/); the service which organises meetings and conferences. A useful link is https://indico.cern.ch/category/201/, which lists UK CMS meetings (Bristol CMS weekly meetings, RA1, trigger, etc.).

9.1 The lxplus remote server

As a CERN affiliate, I have access to the remote server and grid located on site, lxplus. I can ssh into it with

```
ssh ebhal@lxplus.cern.ch
```

As LXPLUS uses the AFS file system, changing permissions of files/folders so that other users can access them is not necessarily as simple as on regular Unix systems (i.e., with `chmod`). If they don't work, see https://ist.njit.edu/afs-permissions/ for a list of the options.

Disk quotas:

Home directory **/afs/cern.ch/user/e/ebhal**: 10 GB

Work directory **/afs/cern.ch/work/e/ebhal**: 100 GB

EOS directory **/eos/user/e/ebhal**: 1 TB. Can check usage at https://cernbox.cern.ch/.

9.2 Certificates and running on the grid(s)

I've been having the occasional problem accessing CMS/CERN webpages, citing problems with my certificates. If I get this problem in the future, visit https://ca.cern.ch/ca/ and install the following: New Grid User certificate; New EduRoam certificate; CERN Certification Authorities Files; CERN Grid Certification Authorities Certificates. I can also check the expiry dates of the certificates I have by going to "My User certificates".

To use the grid on Soolin or another remote server, I had to go onto Firefox on my Mac, then go to Preferences → Advanced → Certificates → View Certificates → Your Certificates, then "Backup" the certificate named "Eshwen Bhal". This should download the certificate to my Downloads folder,

with the extension **.p12**. Rename the file to **myCert.p12** and `scp` it to my home directory on the server. Then type the commands

```
openssl pkcs12 -in myCert.p12 -clcerts -nokeys -out $HOME/.
   globus/usercert.pem
openssl pkcs12 -in myCert.p12 -nocerts -out $HOME/.globus/
   userkey.pem
chmod go-rw $HOME/.globus/userkey.pem
```

to extract my certificate and key, and then initialise the key so that I'm able to start proxies with my certificate. The Import Password is the one I encrypted the certificate with. Then choose the PEM pass phrase as the same one for simplicity. I can then delete the file **myCert.p12**. These instructions are also available at https://ca.cern.ch/ca/Help/?kbid=024010. Then I'll be able to issue the command `voms-proxy-init -voms cms -valid 168:00` to start a proxy of a grid certificate (valid for 168 hours) so that I can run jobs (when prompted, the grid pass phrase is the PEM pass phrase I set earlier). The certificate itself is only valid for a year, so when I need to renew it I can look at the instructions in the link above to get one. I can just download that to my computer, upload it to each remote server I use and follow the steps above to properly initialise it.

Chapter 10

Scripting for MadGraph and root (24/10/2016)

For a lot of tasks, especially ones I'll be repeating (e.g., doing a specific analysis), it's quicker to write and then execute a script. For root, these can either be Python scripts or C++ macros. For, e.g., executing a sequence of terminal commands, I can just write a .sh script and type (in the terminal) !source <script name>.sh!.

I wrote a Python script for analysing a .root file, then drawing and saving certain histograms as .png files.

```
import ROOT

f = ROOT.TFile("../MG5_aMC_v2_4_3/sm_test_ppXll/Events/
    run_01/tag_1_pgs_events.root")
t = f.Get("LHCO")

# Initialise canvas
c = ROOT.TCanvas("myCanvas")
c.Divide(1,2,3,4)

# Draws histograms and saves them as .png files
for x in range(1, 5):
    c.cd(x)

```

```
    if x == 1:
        t.Draw("Jet.Eta")
        c.SaveAs("JetEta.png")
    if x == 2:
        t.Draw("Jet.PT")
        c.SaveAs("JetPT.png")
    if x == 3:
        t.Draw("Jet.Mass")
        c.SaveAs("JetMass.png")
    if x == 4:
        t.Draw("Jet.Phi")
        c.SaveAs("JetPhi.png")

    print "Saved png %d" % (x)

print "All done"

# To add a cut include a second argument in t.Draw()
    containing the restriction. For example, t.Draw("Jet.PT",
     "Jet.Mass > 10")
```

Listing 10.1: Printing histograms with a Python script. File name: readjets.py (v1).

Locally, on my machine, when running MadGraph with ExRootAnalysis, it doesn't convert the .lhe and .lhco files into their respective .root files like it does on Soolin. So this script is to convert them, and then I can analyse them with my Python script.

```
# Unpack the lhco.gz and lhe.gz files generated from Pythia
    and pgs and convert to .root files.
# Note: once unpacked, the .gz files are deleted, so only
    run this once after new Madgraph runs.
# To use the converter, the template is <directory to
    converter> <input file> <output file>

gunzip ../MG5_aMC_v2_4_3/sm_test_ppZjets/Events/run_01/
    tag_1_pgs_events.lhco.gz
```

```
../MG5_aMC_v2_4_3/ExRootAnalysis/ExRootLHCOlympicsConverter
      ../MG5_aMC_v2_4_3/sm_test_ppZjets/Events/run_01/
   tag_1_pgs_events.lhco    ../MG5_aMC_v2_4_3/
   sm_test_ppZjets/Events/run_01/tag_1_pgs_events.root


gunzip ../MG5_aMC_v2_4_3/sm_test_ppZjets/Events/run_01/
   tag_1_pythia_events.lhe.gz

../MG5_aMC_v2_4_3/ExRootAnalysis/ExRootLHEFConverter    ../
   MG5_aMC_v2_4_3/sm_test_ppZjets/Events/run_01/
   tag_1_pythia_events.lhe   ../MG5_aMC_v2_4_3/
   sm_test_ppZjets/Events/run_01/tag_1_pythia_lhe_events.
   root
```

Listing 10.2: Unzip .lhe and .lhco files and convert to .root files. File name: rootconverter.sh.

Some I/O (input/output) from MadGraph and root is displayed below.

The input file (**sm_test.dat**) I used to generate the output displayed in the following plots:

```
#*************************************************************
   *
#*                        MadGraph 5
   *
#*
   *
#*                *                       *
   *
#*                  *         * *         *
   *
#*                    * * * * 5 * * * *
   *
#*                  *         * *         *
   *
#*                *                       *
   *
#*
```

```
#*                                                                      *
#*                                                                      *
#*    The MadGraph Development Team - Please visit us at                *
#*    https://server06.fynu.ucl.ac.be/projects/madgraph                 *
#*                                                                      *
#*                                                                      *
#************************************************************************
#*                                                                      *
#*                                                                      *
#*               Command File for MadGraph 5                            *
#*                                                                      *
#*                                                                      *
#*     run as ./bin/mg5  filename                                       *
#*                                                                      *
#*                                                                      *
#************************************************************************
#*                                                                      *
import model sm
# Define multiparticle labels
define p = g u c d s u~ c~ d~ s~
define j = g u c d s u~ c~ d~ s~
define l+ = e+ mu+
define l- = e- mu-
define vl = ve vm vt
define vl~ = ve~ vm~ vt~
# Specify process(es) to run
generate p p > e- e+ @1
# Output processes to MadEvent directory

```

```
output sm_test_ppXll
launch
pythia=ON
pgs=ON
set nevents 50000
set ptl 60
```

Listing 10.3: MadGraph input file sm_test.dat used to simulate $pp \to e^+e^-$.

Syntax for W^+ boson: W+, W^- boson: W-, Z^0 boson: Z.

Here is the output.

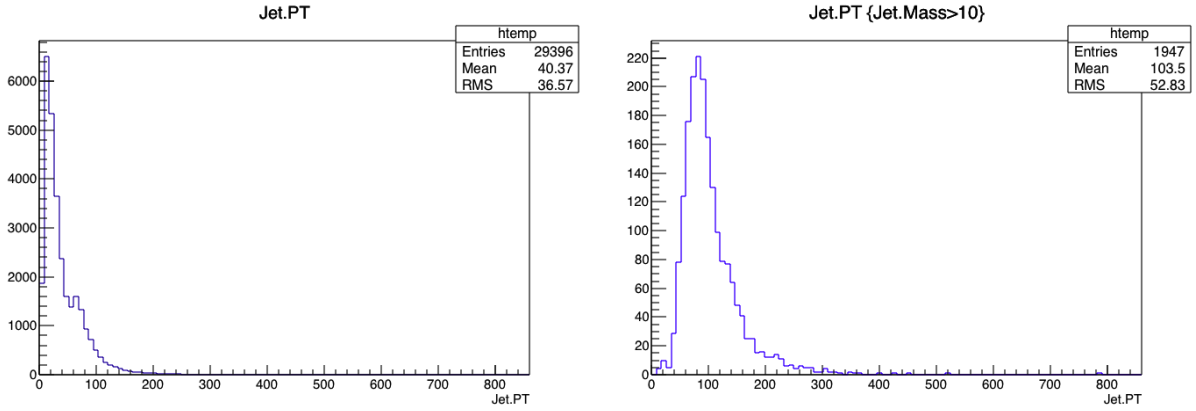

Figure 10.1: The number of entries versus the transverse momentum of the leading jet using 50,000 $pp \to e^+e^-$ events simulated in MadGraph. The left panel shows the raw histogram and the right shows the effect of adding a cut on the mass of the jet $>$ 10 GeV.

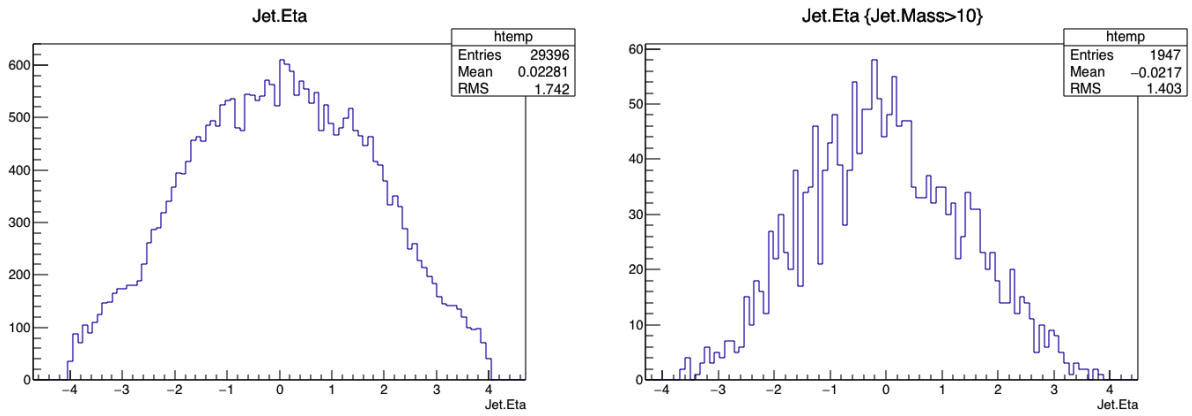

Figure 10.2: The number of entries versus the pseudorapidity of the leading jet using 50,000 $pp \to e^+e^-$ events simulated in MadGraph. The left panel shows the raw histogram and the right shows the effect of adding a cut on the mass of the jet $>$ 10 GeV.

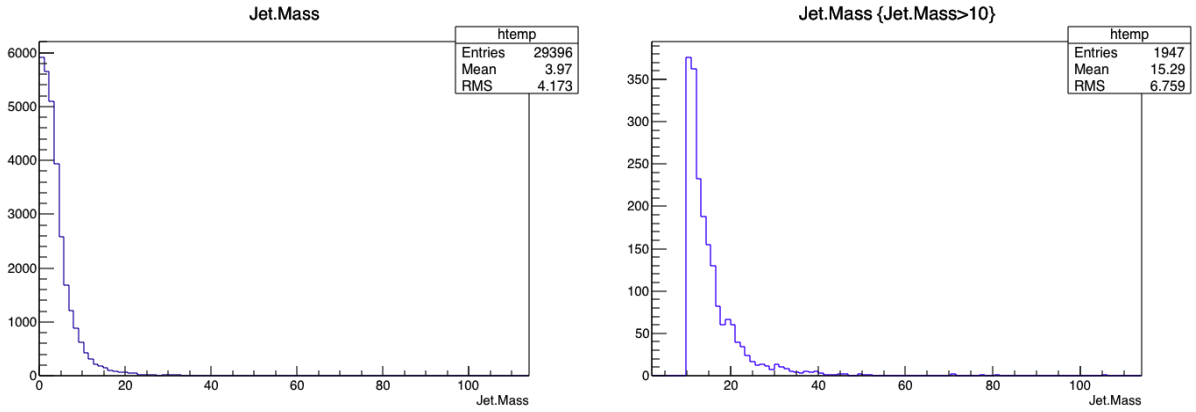

Figure 10.3: The number of entries versus the mass of the leading jet using 50,000 $pp \to e^+e^-$ events simulated in MadGraph. The left panel shows the raw histogram and the right shows the effect of adding a cut on the mass of the jet > 10 GeV.

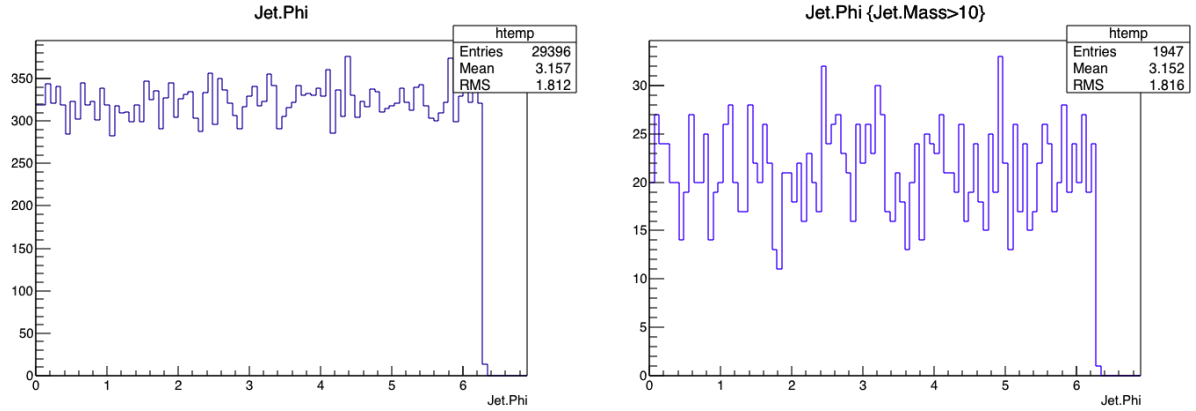

Figure 10.4: The number of entries versus the phi (meaning?) of the leading jet using 50,000 $pp \to e^+e^-$ events simulated in MadGraph. The left panel shows the raw histogram and the right shows the effect of adding a cut on the mass of the jet > 10 GeV.

It is obvious that even with a small cut on jet mass, the histograms noticeably change. For the jet mass plot, it retains the same shape after the cut because were effectively only changing the range. But applying a cut on jet mass excludes different relevant data for jet ϕ, jet η, jet p_{T}, etc. so the plots look quite different. This kind of analysis can be used to determine the dependencies of certain variables on others, e.g., when you change/exclude certain ranges of jet p_{T}, how does the histogram for jet mass change? And so on. This will be especially useful when analysing real LHC data to try and determine, not just the mass, but different properties of dark matter and different particle decay modes.

I've also figured out how to plot a histogram, and then overlay the histogram after a cut on the same axes. I've also streamlined my Python script readjets.py by storing the variable names and the cuts as character strings. So if I want to change the cut or change one of the variables I want to look at,

I only have to edit part of one line as opposed to multiple entries throughout the script. An example plot and the new script are displayed below.

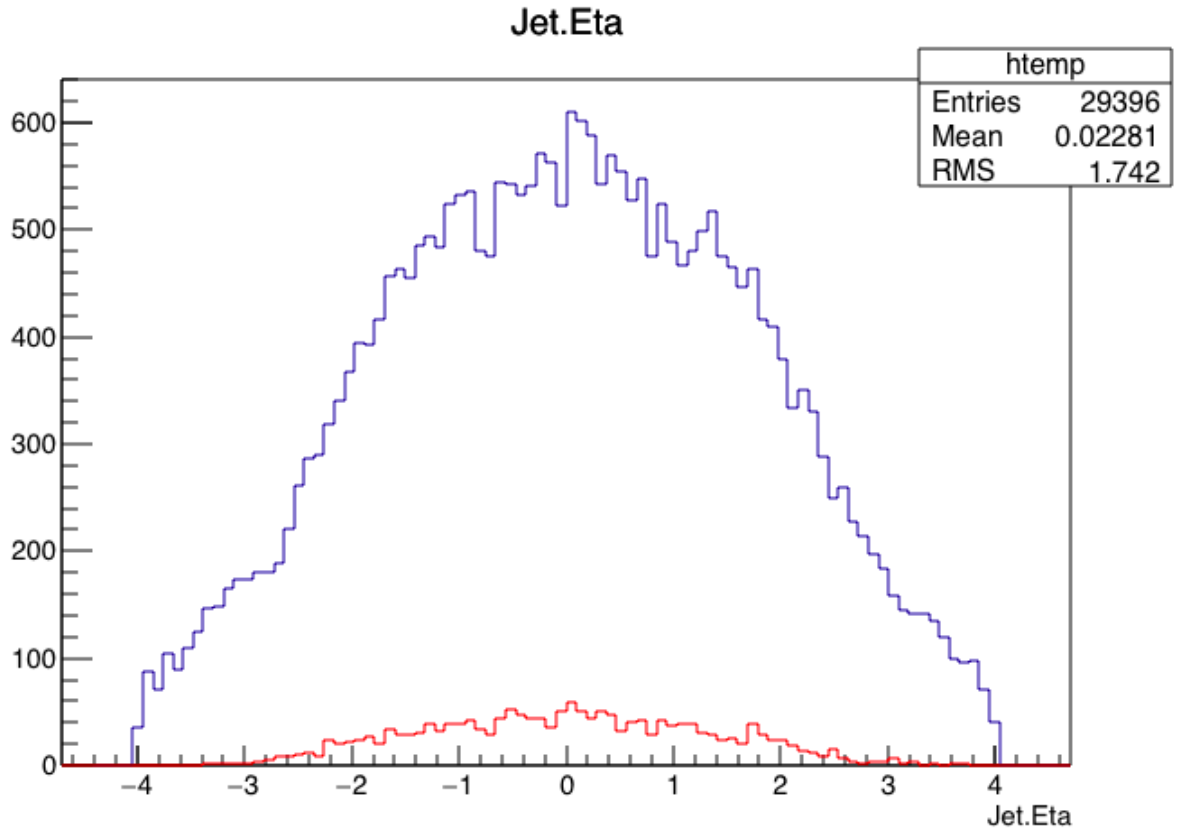

Figure 10.5: The number of entries versus the pseudorapidity of the leading jet using 50,000 $pp \rightarrow e^+e^-$ events simulated in MadGraph. The blue line shows the raw data and the red line shows the effect of a cut on jet $p_T > 100$ GeV.

```
import ROOT

f = ROOT.TFile("../MG5_aMC_v2_4_3/sm_test_ppXll/Events/
    run_01/tag_1_pgs_events.root")
t = f.Get("LHCO")

# Initialise canvas
c = ROOT.TCanvas("myCanvas")
c.Divide(1,2,3,4)

```

```
# Set the cut to impose on the variable being plotted in the
    for loop
cutVariable = "Jet.PT>100"

# Draws histograms and saves them as .png files
for x in range(1, 5):
    c.cd(x)

    # Set the variable that is being plotted
    if x == 1:
        variable = "Jet.Eta"
    if x == 2:
        variable = "Jet.PT"
    if x == 3:
        variable = "Jet.Mass"
    if x == 4:
        variable = "Jet.Phi"

    t.Draw("%s" % variable)
    # Set line colour (red) for next plot
    t.SetLineColor(2)
    # Superimpose (on the same axes) the plot with the cut
    t.Draw("%s" % variable, "%s" % cutVariable, "SAME")
    c.SaveAs("%s.png" % variable)

    print "Saved png %d" % (x)
    # Reset line colour (blue)
    t.SetLineColor(4)

print "All done"

# To plot a histogram, then apply a cut and compare the two,
    type, e.g.,
# t.Draw("Jet.Eta")
# t.SetLineColor(2) <---- This sets the line colour for the
```

```
    next plot
43 # t.Draw("Jet.Eta","Jet.PT>10","SAME")  <--- draws Jet.Eta
    with cut on the same axes
```

Listing 10.4: A Python script used to print histograms with a cut on the same axes. File name: readjets.py (v2).

Chapter II

Filling, Comparing and Normalizing Histograms using Macros (02/11/2016)

With Python, it's quite easy to plot and slightly manipulate the contents of .root files. But once the analysis gets a bit more complicated (e.g., normalizing histograms), it's difficult to accomplish in Python because most of the stuff (like the classes and documentation) is written in/for C++. So for the time being, I'll be writing C++ macros rather than Python scripts for analysis.

With Bjoern's help, I created some macros to compare and analyse histograms. To be able to manipulate them for, e.g., normalizing, you have to create a new histogram and fill it with the contents of the branch/leaf you want to analyse because you can't edit the contents of a root file itself. All of this is difficult in Python, hence the switch to C++.

Three types of files were created: the header file (.h) which contains the information and declarations, etc., pertaining to the tree in the root file; the source file (.C) in which most of the analysis commands are executed; the "daddy" macro, which is what is executed in ROOT and ties the other source files and executes the necessary loops, etc.

As an example, I'm analysing the η component of a jet from a MadGraph analysis from the processes $pp \to t\bar{t}$, $pp \to W+$ jets, and $pp \to Z+$ jets. These are SM backgrounds for a hadronic search because the DM final state is usually large $E_{\mathrm{T}}^{\mathrm{miss}}$ and several jets. The MadGraph output is converted into a .root file using my **rootconverter.sh** script. Then ROOT is launched and the analysis is done by executing the "daddy" macro (**execComparejets.C**) and I get a histogram out of it with three normalized plots of the respective Jet.Eta plots.

So in total I had 10 files used (daddy macro + 3×3 files for each analysis – source file, header file,

root file):

– execComparejets.C

– ppttbar.C ppttbar.h ppttbar.root

– ppWjets.C ppWjets.h ppWjets.root

– ppZjets.C ppZjets.h ppZjets.root

And the output file **comparejets.png**. At the moment, the aim is to plot SM background processes so we can find DM/exotic signals easier.

(Notes on MADGRAPH syntax can be found at http://www.niu.edu/spmartin/madgraph/madsyntax.html.)

The "daddy" macro **execComparejets.C**:

```
// Script to run the macros to plot histograms from multiple
    root files

#include "ppttbar.C"
#include "ppWjets.C"
#include "ppZjets.C"

#include <TLegend.h>

void execComparejets() {
    ppttbar t;
    t.Loop();

    ppWjets u;
    u.Loop();

    ppZjets v;
    v.Loop();

    // Add a legend to the canvas
    // leg = new TLegend(0.1,0.65,0.35,0.9); // Range of the
    legend box (x1,y1,x2,y2)
    // leg->AddEntry(ttbar_jet_eta, "pp -> ttbar", "f");
    // leg->AddEntry(Wjets_jet_eta, "pp -> W + jets", "f");
```

```
23      // leg->AddEntry(Zjets_jet_eta, "pp -> Z + jets", "f");
        // leg->Draw();
25
        c1->SaveAs("comparejets.png");
27  }

29  // FIND A WAY TO ADD THE LEGEND
```

Listing 11.1: A C++ macro designed to execute and loop over the macros tied to their respective root file. File name: execComparejets.C (VI).

The C source file **ppttbar.C**:

```
1   #define ppttbar_cxx

3   #include "ppttbar.h"
    #include <TH2.h>
5   #include <TStyle.h>
    #include <TCanvas.h>
7   #include <TString.h>
    #include <TLegend.h>
9   #include <iostream.h>

11  void ppttbar::Loop() {

13  //   In a ROOT session, you can do:
    //      Root > .L ppttbar.C
15  //      Root > ppttbar t
    //      Root > t.GetEntry(12); // Fill t data members with
       entry number 12
17  //      Root > t.Show();       // Show values of entry 12
    //      Root > t.Show(16);     // Read and show values of
       entry 16
19  //      Root > t.Loop();       // Loop on all entries
    //

21
    //     This is the loop skeleton where:
```

```
23 //    jentry is the global entry number in the chain
   //    ientry is the entry number in the current Tree
25 //  Note that the argument to GetEntry must be:
   //    jentry for TChain::GetEntry
27 //    ientry for TTree::GetEntry and TBranch::GetEntry
   //
29 //       To read only selected branches, Insert statements
     like:
   // METHOD1:
31 //    fChain->SetBranchStatus("*",0);  // disable all
     branches
   //    fChain->SetBranchStatus("branchname",1);  // activate
     branchname
33 // METHOD2: replace line
   //    fChain->GetEntry(jentry);       //read all branches
35 //by  b_branchname->GetEntry(ientry); //read only this
     branch

37       if (fChain == 0)
               return;
39       Long64_t nentries = fChain->GetEntriesFast();

41       // Create histogram to be filled
         TH1F* ttbar_jet_eta = new TH1F("jet_eta", "jet_eta",
     30, -4, 4);
43
         Long64_t nbytes = 0, nb = 0;
45
         // Loop over entries and fill histograms
47       for (Long64_t jentry = 0; jentry < nentries; jentry++)
      {
               Long64_t ientry = LoadTree(jentry);
49             if (ientry < 0)
                     break;
51
```

```
            nb = fChain->GetEntry(jentry);
53          nbytes += nb;

55          // Loop to apply a cut
            if (Jet_PT[0] > 200)
57                ttbar_jet_eta->Fill(Jet_Eta[0]);
     // if (Cut(ientry) < 0) continue;
59      }

61    // Create, then split the canvas into two rows
      TCanvas* c1 = new TCanvas("c1");
63    ttbar_jet_eta->Draw();
      // Normalize histogram so it encloses unit area
65    ttbar_jet_eta->Scale( 1/ttbar_jet_eta->Integral() );
}
```

Listing 11.2: A C++ macro used to analyse the root file with the same name. A histogram is declared and filled with the Jet.Eta branch from the tree, then a histogram is drawn and normalised. File name: ppttbar.C (v1).

The header file **pptbar.h**:

The header file for the macro with the same name (ppttbar). I didn't display the file itself because the code stretches over 10 pages. This file includes all the declarations, branch names and other information relating to the root file. File name: ppttbar.h (v1).

The output **comparejets.png**:

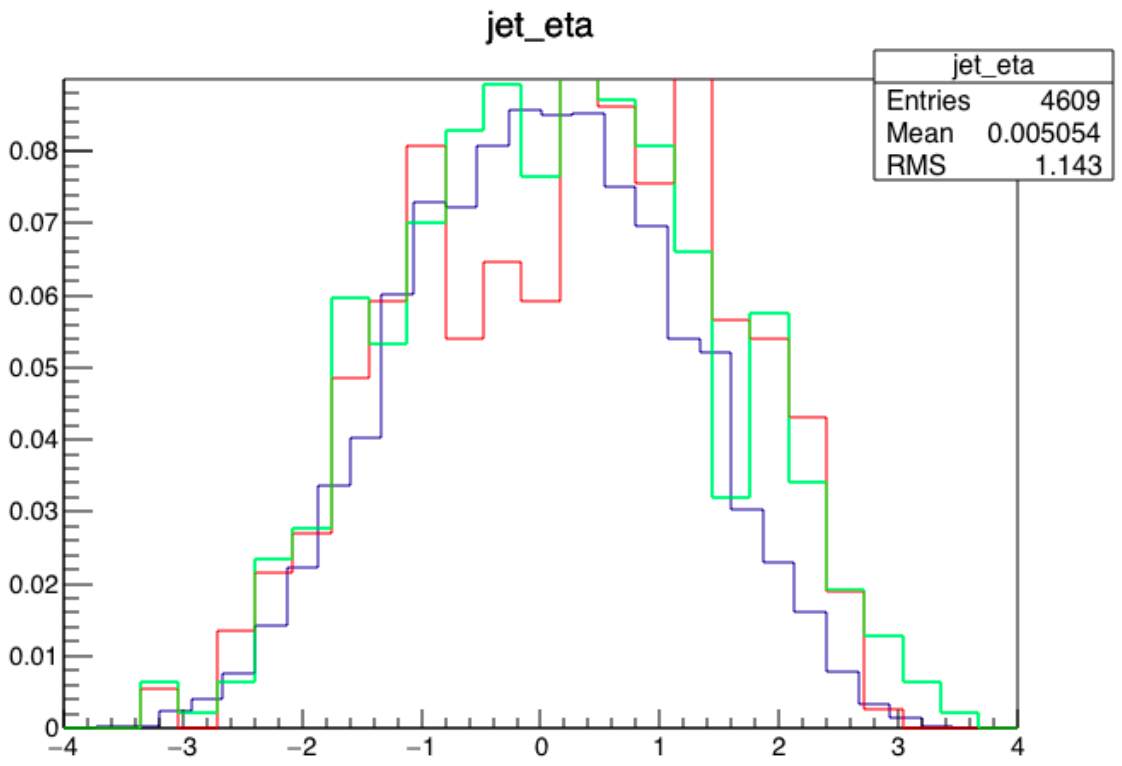

Figure 11.1: Multiple histograms showcasing the normalized entries versus pseudorapidity from three standard model processes: $pp \to t\bar{t}$ (blue); $pp \to W+$ jets (red); and $pp \to Z+$ jets (green).

I should find a way to plot such that the histogram with the largest y-value is plotted first. So for the time being, until I can streamline it, the process for getting and analysing data is as follows:

1. Write an input file for MadGraph detailing the process I want to look at.
2. Run MadGraph with the input file and get the output .lhco files, etc.
3. Convert the .lhco and .lhe files to .root files using my rootconverter.sh script.
4. Run ROOT and get the MakeClass file (.h and .C) for the .root file.
5. Use the templates above to edit the .h and .C files accordingly, paying close attention to the other header files I need to include, the bin sizes of the histograms, the names of the branches I'm accessing, normalization, line colours, whether the histogram is cut off because of the canvas size.
6. Write the "daddy" macro to execute and run all the analyses and get the histogram(s) out.

UPDATE: As a start to the streamlining process, I wrote a header file **global.h** which is intended to store global variables/constants, etc., and also all my "include" headers. At the time of writing I've only included the ranges and number of bins for initialising histograms (because for now I'll be

plotting the same type of histogram in one graph, so it makes sense that they all have the same range and number of bins), and the "include" headers (so in each file I only have to include global.h and the C++ source files that are tied to it). Hopefully I'll be able to expand this to include the names of the histograms and cut variables (as character strings?) so that they only have to be declared once and I only have to edit one line if I want to plot something else/apply a different cut. The current version of global.h is presented below:

```
// Global variables/constants/include files are defined here

// Constants for histogram initialization
const int N_BINS = 25;
const int X_MIN = -4;
const int X_MAX = 4;

// ROOT header files
#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TString.h>
#include <TLegend.h>
#include <THStack.h>

// Header file for the classes stored in the TTree if any.
#include "TClonesArray.h"
#include "TObject.h"

// C++ header files
#include <iostream>
```

Listing 11.3: The global header file to include global variables/constants/headers that my C++ macros need. File name: global.h (v1).

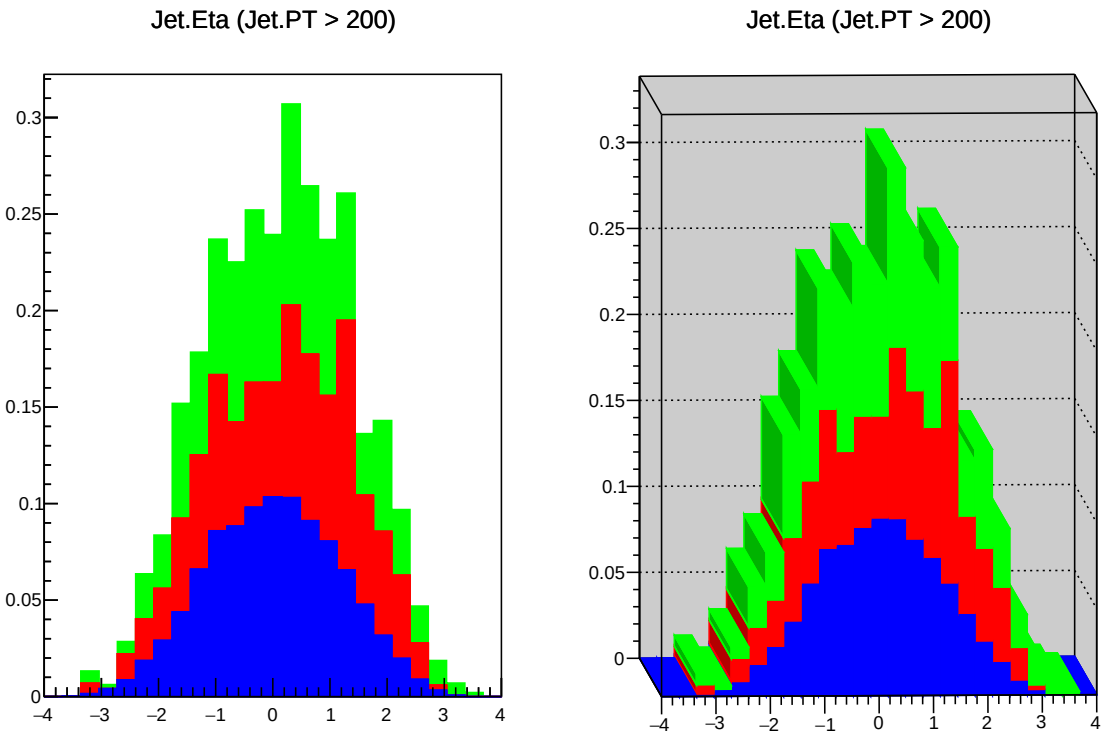

Figure 11.2: Multiple histograms showcasing the normalized entries versus pseudorapidity from three standard model processes: $pp \to t\bar{t}$ (blue); $pp \to W+$ jets (red); and $pp \to Z+$ jets (green). These were plotted as a histogram stack. The left panel is from the base `Draw()` command in ROOT, and the right panel displays the same stack as a "lego" plot.

I've updated my macros slightly to divide the canvas and make a "lego" plot that's angled slightly to give a different perspective. Both plots were created using the `THStack` class so that the histograms can be added and then plotted as a stack so you don't get any weird overlap like you do when plotting them one-by-one. I haven't updated the code enough to justify sticking it in the lab book. Maybe I'll add it once I've figured out how to add the legend and added some other features.

UPDATE: I've figured out how to add legends to the plots and have tidied up/expanded some of the files. I might add the new versions of the code later on.

11.1 Updated code for C++ macros

An updated version (v3) of my files are included below (but not in my physical lab book for waste of time and paper) for future reference. The header files are not included because they have not changed, and are the same across all three processes with the exception of the function names and the names of the .root files. The update notes: more parameters and include files being held in **global.h**; the

canvas declaration and some aesthetics being executed by the "daddy" macro; a legend; and scaling by luminosity instead of area.

The "daddy" macro:

```
// Script to run the macros and plot histograms from
   multiple root files

// Most things (canvas, legend, etc.) can be initialised in
   this file, but the first .C file called must initialise
   the histogram stack (THStack)

#include "ppttbar.C"
#include "ppWjets.C"
#include "ppZjets.C"
#include "global.h"

void execComparejets() {

    // Create canvas of width w and height h (in pixels),
   then split into two columns
    TCanvas* c1 = new TCanvas("c1");
    Int_t w = 900, h = 600;
    c1->SetCanvasSize(w,h);
    c1->Divide(2,1);

    // Run main loops to fill and draw histograms
    ppttbar t;
    t.Loop();
    ppWjets u;
    u.Loop();
    ppZjets v;
    v.Loop();

    // Set y-axis range of plots
    Double_t ymin = stackedhists->GetMinimum(), ymax =
   stackedhists->GetMaximum();
```

```
28        stackedhists->SetMinimum(ymin);
          stackedhists->SetMaximum(ymax);
30
          // Set axes labels and offset (in % of pad width) so
    they don't overlap with axis ticks
32        stackedhists->GetXaxis()->SetTitle("Jet.Eta");
          stackedhists->GetXaxis()->SetTitleOffset(1.4);
34        stackedhists->GetYaxis()->SetTitle("Entries (
    normalized by luminosity)");
          stackedhists->GetYaxis()->SetTitleOffset(1.5);
36
          // Set aesthetics for lego plot
38        c1->cd(2);
          gPad->SetFrameFillColor(17);
40        gPad->SetTheta(3.77);
          gPad->SetPhi(2.9);
42
          // Add the same legend to both plots
44        for (Int_t i = 1; i < 3; ++i) {
                c1->cd(i);
46              gPad->BuildLegend(0.1,0.65,0.43,0.9,"");
                // Range of the legend box (x1,y1,x2,y2). Origin
     at bottom left
48        }
          // Save as pdf because png/bmp doesn't work properly
    at high resolution
50        c1->SaveAs("comparejets.pdf");
}
52
// FIGURE OUT HOW TO SET THE Z-AXIS LABEL (AND REMOVE THE Y-
    AXIS) FOR THE LEGO PLOT
```

Listing 11.4: The C++ macro used to execute the macros pertaining to the individual root files, but updated and streamlined. File name: execComparejets.C (v3).

The global header file **global.h**:

```
// Global variables/constants/include files are defined here

// ROOT header files
#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TString.h>
#include <TLegend.h>
#include <THStack.h>
#include <TPad.h>

// Header file for the classes stored in the TTree if any.
#include "TClonesArray.h"
#include "TObject.h"

// C++ header files
#include <iostream>

// Constants for histogram initialization
const int N_BINS = 25;
const int X_MIN = -4;
const int X_MAX = 4;

// Beam properties. N_EVENTS is defined in MadGraph input
   file, luminosity_pb is assumed
const double N_EVENTS = 100000;
const double luminosity_pb = 20000;

// Declare the type of histogram to draw
TString histType_pad1 = "";
```

```
33 TString histType_pad2 = "lego1";
```

Listing 11.5: The global header file that includes some declarations, global variables/constants/include files, but updated and streamlined. File name: global.h (v3).

The source file **ppttbar.C**:

```
1  #define ppttbar_cxx

3  #include "ppttbar.h"
   #include "global.h"
5
   void ppttbar::Loop() {
7
        if (fChain == 0)
9             return;
        Long64_t nentries = fChain->GetEntriesFast();
11
        // Create histogram to be filled
13      TH1F* ttbar_jet_eta = new TH1F("jet_eta", "pp->tt~",
    N_BINS, X_MIN, X_MAX);

15      Long64_t nbytes = 0, nb = 0;

17      // Loop over entries and fill histograms
        for (Long64_t jentry = 0; jentry < nentries; jentry++)
     {
19              Long64_t ientry = LoadTree(jentry);
                if (ientry < 0)
21                    break;

23              nb = fChain->GetEntry(jentry);
                nbytes += nb;
25
                // Loop to apply a cut
27              if (Jet_PT[0] > 200)
                      ttbar_jet_eta->Fill(Jet_Eta[0]);
```

```
29        // if (Cut(ientry) < 0) continue;
          }
31
          // Normalize histogram by luminosity
33        Double_t cross_sec = 504.9;
          ttbar_jet_eta->Scale( cross_sec * luminosity_pb /
    N_EVENTS );
35
          // Set aesthetics
37        ttbar_jet_eta->SetLineColor(kBlue);
          ttbar_jet_eta->SetFillColor(kBlue);
39        ttbar_jet_eta->SetMarkerStyle(21);
          ttbar_jet_eta->SetMarkerColor(kBlue);
41
          // Create a histogram stack
43        THStack *stackedhists = new THStack("stackedhists", "
    Jet.Eta (Jet.PT > 200)");
          stackedhists->Add(ttbar_jet_eta);
45
        // Canvas is initialised in execComparejets.C before
    calling this file
47      c1->cd(1);
        // Draw 1D plot
49      stackedhists->Draw(histType_pad1);

51      c1->cd(2);
        // Draw lego plot
53      stackedhists->Draw(histType_pad2);
}
```

Listing 11.6: The C++ macro used to analyse ppttbar.root, but updated and streamlined. File name: ppttbar.C (v3).

The source file **ppWjets.C**:

```
#define ppWjets_cxx
2
```

```
#include "ppWjets.h"
#include "global.h"

void ppWjets::Loop() {

      if (fChain == 0)
            return;
      Long64_t nentries = fChain->GetEntriesFast();

      // Create histogram to be filled
      TH1F* Wjets_jet_eta = new TH1F("jet_eta", "pp->W+jets"
   , N_BINS, X_MIN, X_MAX);

      Long64_t nbytes = 0, nb = 0;

      // Loop over entries and fill histograms
      for (Long64_t jentry = 0; jentry < nentries; jentry++)
    {
            Long64_t ientry = LoadTree(jentry);
            if (ientry < 0)
                  break;

            nb = fChain->GetEntry(jentry);
            nbytes += nb;

            // Loop to apply a cut
            if (Jet_PT[0] > 200)
                  Wjets_jet_eta->Fill(Jet_Eta[0]);
      // if (Cut(ientry) < 0) continue;
      }

      // Normalize histogram by luminosity
      Double_t cross_sec = 2.144e+04;
      Wjets_jet_eta->Scale( cross_sec * luminosity_pb /
   N_EVENTS );
```

```
36        // Set aesthetics
          Wjets_jet_eta->SetLineColor(kRed);
38        Wjets_jet_eta->SetFillColor(kRed);
          Wjets_jet_eta->SetMarkerStyle(21);
40        Wjets_jet_eta->SetMarkerColor(kRed);

42        stackedhists->Add(Wjets_jet_eta);

44      // Canvas is initialised in execComparejets.C before
    calling this file
        c1->cd(1);
46      // Draw 1D plot
        stackedhists->Draw(histType_pad1);
48
        c1->cd(2);
50      // Draw lego plot
        stackedhists->Draw(histType_pad2);
52  }
```

Listing 11.7: The C++ macro used to analyse ppWjets.root, but updated and streamlined. File name: ppWjets.C (v3).

The source file **ppZjets.C**:

```
#define ppZjets_cxx
2
#include "ppZjets.h"
4 #include "global.h"

6 void ppZjets::Loop() {

8       if (fChain == 0)
              return;
10      Long64_t nentries = fChain->GetEntriesFast();

12      // Create histogram to be filled
```

```
    TH1F* Zjets_jet_eta = new TH1F("jet_eta", "pp->Z+jets"
, N_BINS, X_MIN, X_MAX);

    Long64_t nbytes = 0, nb = 0;

    // Loop over entries and fill histograms
    for (Long64_t jentry = 0; jentry < nentries; jentry++)
 {
          Long64_t ientry = LoadTree(jentry);
          if (ientry < 0)
                break;

          nb = fChain->GetEntry(jentry);
          nbytes += nb;

          // Loop to apply a cut
          if (Jet_PT[0] > 200)
                Zjets_jet_eta->Fill(Jet_Eta[0]);
    // if (Cut(ientry) < 0) continue;
    }

    // Normalize histogram by luminosity
    Double_t cross_sec = 1.166e+04;
    Zjets_jet_eta->Scale( cross_sec * luminosity_pb /
N_EVENTS );

    // Set aesthetics
    Zjets_jet_eta->SetLineColor(kGreen);
    Zjets_jet_eta->SetFillColor(kGreen);
    Zjets_jet_eta->SetMarkerStyle(21);
    Zjets_jet_eta->SetMarkerColor(kGreen);

    stackedhists->Add(Zjets_jet_eta);

  // Canvas is initialised in execComparejets.C before
```

```
    calling this file
      c1->cd(1);
46    // Draw 1D plot
      stackedhists->Draw(histType_pad1);
48
      c1->cd(2);
50    // Draw lego plot
      stackedhists->Draw(histType_pad2);
52 }
```

Listing 11.8: The C++ macro used to analyse ppZjets.root, but updated and streamlined. File name: ppZjets.C (v3).

The output file **comparejets.pdf**:

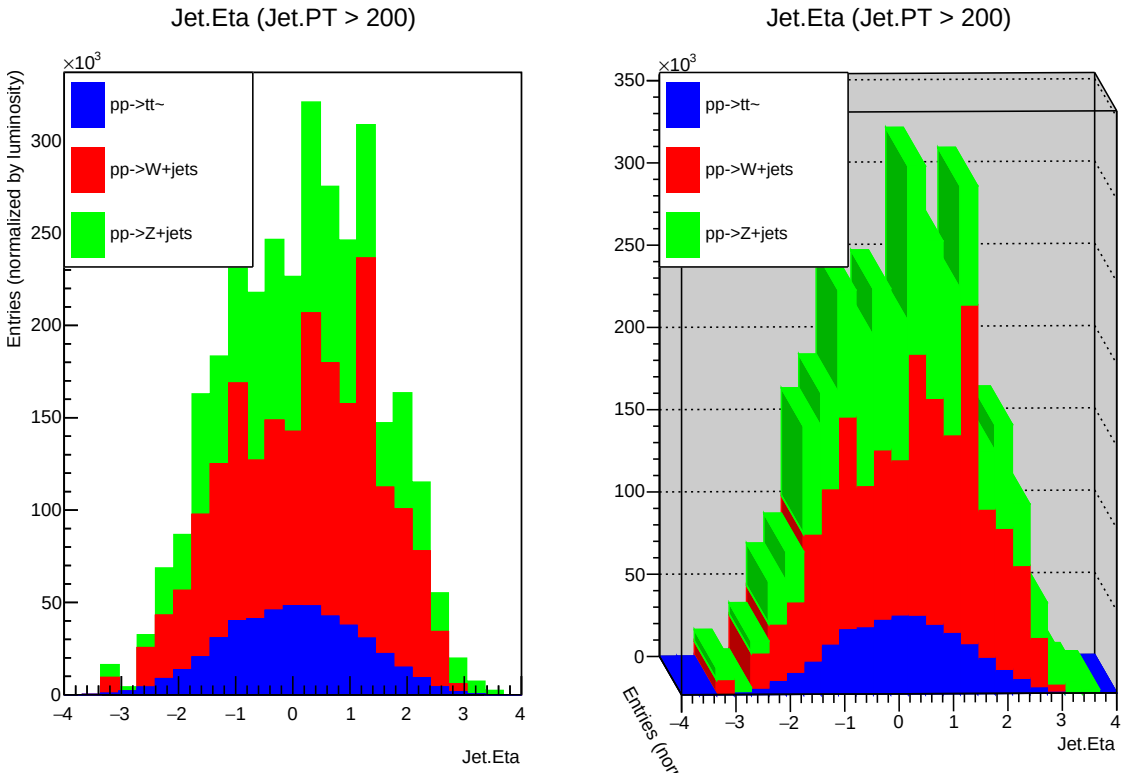

Figure 11.3: The same histograms from the previous subsection plotted as a histogram stack, this time with a legend and axis labels. The entries are also normalized by luminosity rather than area.

In this version of the code, each histogram was normalized according to the luminosity. The scale factor $s.f.$ was calculated using the simple, standard formula

(II.1) $$N = \sigma\mathcal{L}$$

where N is the number of events, σ is the interaction cross section, and $\mathcal{L}$ is the luminosity. These values are detailed in Table II.1. Then the scale factor is

(II.2) $$s.f. = \sigma\mathcal{L}/N$$

and is dimensionless, since $[\sigma]$ = pb and $[\mathcal{L}]$ = pb^{-1}.

Property	$pp \to t\bar{t}$	$pp \to W + \text{jets}$	$pp \to Z^0 + \text{jets}$
Number of Events (MadGraph)	100 000	100 000	100 000
Cross section (pb)	504.9	2.144×10^4	1.166×10^4
Assumed luminosity (pb^{-1})	20 000	20 000	20 000
Jet η events before cut	99 999	99 992	99 996
Jet η events after cut (Jet $p_{\mathrm{T}} > 200$)	4 609	371	470
Efficiency (%)	4.6	0.37	0.47

Table II.1: The properties of the MadGraph processes that were simulated.

The luminosity from a process is given in the stream of output text during a MadGraph run, but I didn't realise until after I had made the plots. So I just assumed the luminosity was 20 fb^{-1} (20,000 pb^{-1}). The luminosity would most likely be different for each decay and so would scale the histograms slightly differently, but the point of this exercise was to demonstrate the technique behind filling and plotting histograms rather than a rigorous analysis. The efficiency was just calculated from the number of events after the cut $\nabla\cdot$ number before the cut.

II.2 Running MadGraph for benchmarks

I've run a few different simulations in MadGraph to get benchmark times to see which processes are the most efficient, possibly for future backgrounds or analyses. The following processes (with the names of the output files) in terms of MadGraph syntax were run with 10,000 events. Bottom quarks and tau leptons were also included in the multiparticle labels in the input files. Some processes included Quantum Chromodynamics (QCD) at Next to Leading Order (NLO) and ran their own subprocesses within MadGraph. All other settings were left as their default.

```
generate p p > t t~, (t > b W+, W+ > l+ vl), (t~ > b~ W-, W-
    > s c~)
output ttbar_onshell_nomerging

generate p p > t t~ [QCD]
output ttbar_onshell_NLO

generate p p > W+
add process p p > W+ j
add process p p > W+ j j
output W_multijets

decay Z > b b~
decay Z > vl vl~
generate p p > Z W- [QCD]
output WmZ_NLO

generate p p > Z W-
add process p p > Z W- j
output WmZj

decay Z > b b~
decay Z > vl vl~
generate p p > Z W+ [QCD]
output WpZ_NLO

generate p p > Z W+
add process p p > Z W+ j
output WpZj

generate p p > Z
add process p p > Z j
add process p p > Z j j
output Z_simplejetmerging
```

```
generate p p > Z, (Z > vl vl~)
add process p p > Z j, (Z > vl vl~)
add process p p > Z j j, (Z > vl vl~)
output Zjets_invisible

generate p p > Z Z [QCD]
output ZZ_NLO

generate p p > Z Z
add process p p > Z Z j
output ZZjets
```

The results are displayed below.

```
PROCESS INFORMATION:

ttbar_onshell_nomerging -
real  0m53.077s
user  1m49.487s
sys 0m8.472s
Effective Luminosity 234.883761898 pb^-1
Cross-section :   51.09 +- 0.1578 pb
Nb of events :  10000


ttbar_onshell_NLO -
real  3m41.027s
user  5m39.618s
sys 0m25.293s
Number of events generated: 10000
Total cross section: 6.795e+02 +- 4.2e+00 pb


W_multijets -
# For this one, you set the decay product to W, but I got an
    error in MadGraph so I set it to W+. Although it shouldn
```

```
   't make a difference whether it's +/-.
real  1m4.011s
user  2m31.371s
sys 0m14.129s
Effective Luminosity 0.118195861677 pb^-1
Cross-section :   1.015e+05 +- 102.8 pb
Nb of events :  10000


WmZ_NLO -
real  2m56.422s
user  4m15.820s
sys 0m19.251s
Number of events generated: 10000
Total cross section: 1.730e+01 +- 5.6e-02 pb


WmZj -
real  1m10.692s
user  2m34.989s
sys 0m9.360s
Effective Luminosity 753.584802907 pb^-1
Cross-section :   15.92 +- 0.04435 pb
Nb of events :  10000


WpZ_NLO -
real  3m2.199s
user  4m36.869s
sys 0m19.974s
Number of events generated: 10000
Total cross section: 2.720e+01 +- 9.3e-02 pb


WpZj -
```

```
real  1m10.980s
user  2m54.885s
sys 0m10.221s
Effective Luminosity 456.765612249 pb^-1
Cross-section :   26.27 +- 0.05938 pb
Nb of events :  10000


Z_simplejetmerging -
real  1m23.092s
user  3m12.965s
sys 0m16.947s
Effective Luminosity 0.226491235864 pb^-1
Cross-section :   5.298e+04 +- 53.23 pb
sNb of events :  10000

Zjets_invisible -
real  1m26.210s
user  3m21.659s
sys 0m16.534s
Effective Luminosity 1.13524830328 pb^-1
Cross-section :   1.057e+04 +- 17.01 pb
Nb of events :  10000


ZZ_NLO -
real  3m27.405s
user  6m47.834s
sys 0m27.019s
Number of events generated: 10000
Total cross section: 1.359e+01 +- 4.7e-02 pb

ZZjets -
real  1m2.725s
user  2m12.709s
```

```
91 sys 0m8.286s
   Effective Luminosity 908.754256946 pb^-1
93 Cross-section :   13.2 +- 0.03977 pb
   Nb of events :  10000
```

Listing 11.9: Benchmark run times for different processes executed in MadGraph. File name: process_comparison.txt.

The NLO processes automatically set the number of events to 10,000 and give me an error message if I use the command `set nevents ...`, so I set the number of events to 10,000 for all runs. pythia and PGS were `OFF`, but the NLO processes run their own extra routines, like MCatNLO and HERWIG6.

To compare the cross sections I got with the currently accepted values, see https://twiki.cern.ch/twiki/bin/viewauth/CMS/SummaryTable1G25ns.

Chapter 12

CMSSW and Fireworks (cmsShow) (08/11/2016)

CMS SoftWare (CMSSW) is software that CMS physicists can access to perform analyses, etc. The GitHub repository is located at https://github.com/cms-sw/cmssw. If I want to set up a CMSSW environment, I need to type the commands

```
ssh ebhal@lxplus.cern.ch # or whatever remote server I'm
   logging in to
cmsrel CMSSW_7_6_3 # Only need to do this once. Can also
   change version I'm working with
cd CMSSW_7_6_3/src
cmsenv
```

and I can follow the instructions in 18 to use the different modules and packages it contains. CMSSW can be used to access and analyse data and Monte Carlo, as well as simulate the latter. If I want to use MadGraph or MadAnalysis or something on Soolin, I need to execute the commands above (or put them in a script and `source` it) before using the programs. There's also a software component called Fireworks (or cmsShow) which is an event-display project, i.e., it displays a visualisation of the events from a ROOT file. It shows particle tracks and other information confined within the detector/simulation. More information can be found at https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookFireworks.

To use Fireworks, download it using the instructions in the above link. Then navigate to the directory **cmsShow-<version>/** and type

```
./cmsShow <name of root file>
```

The example file **data.root** gives this output:

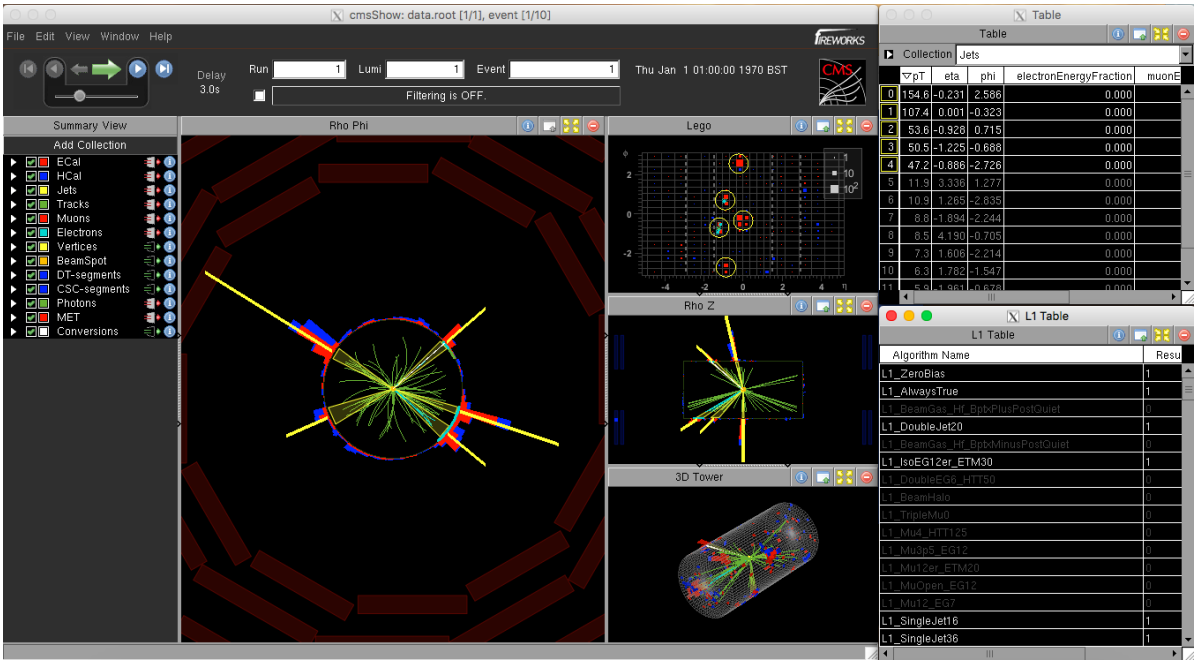

Figure 12.1: The Fireworks interface using the example root file provided by the program (data.root). This program provides a visualisation of the particle tracks and detector information (and probably various tools that I haven't explored yet).

Chapter 13

Using MadAnalysis and searching for dark matter (15/11/2016)

MadAnalysis is a tool used to perform analyses on phenomenological investigations (looking for new particles, new physics, excluding mass ranges of new particles, etc.) at particle colliders. It can analyse data generated by MadGraph and other Monte Carlo event generators. Simple analyses can be run using a Python interface (the "first running mode of the program"), whilst more complex analyses usually require the use of C++ (the "second running modes"). This is similar to ROOT; whilst it's easy and intuitive to use Python for simple scripts, C++ macros are normally needed for more complex stuff.

MadAnalysis 5 [37] can be downloaded in a tar ball from the Internet, with the requirements being listed in the README. It runs similarly to MadGraph: navigating to the directory **MadAnalysis/MA5_v1_4/** on my computer and typing `./bin/ma5` runs the initialiser and dependency-checker. This environment is also where I can check for updates and install packages, just like in MadGraph (`install <name of package>`, etc.). If I'm installing MadAnalysis for the first time, check which packages are already there, and install Delphes and other necessary ones. If a package won't install properly, try typing `./bin/ma5 -R` and installing again. From now, until I state otherwise, I will be using MadAnalysis 5 v1.4.

To run an analysis, type `./bin/ma5 -s -R <name of input file>`, with the extension for the input file being .ma or .ma5. The `-s` stands for "script mode" so the program quits after the script is executed, and the `-R` stands for "reco", i.e., the program runs at the reconstructed level (by

default, without -R, it is launched at the parton level). When running at the reconstructed level, the package FastJet [38] is used, which is software for finding and analysing jets in collider data. An example of an input file – which is a dark Higgs (Higgs-like particle that couples to dark matter) search – is displayed below.

```
set main.fastsim.package = delphes
set main.fastsim.detector = atlas
set main.lumi = 3.2
set main.normalize = lumi
set main.outputfile = "MonoDarkHiggs.lhe"

# import the signal/background files
import /users/bp15067/storage/generator/mg_desy/results/
   ZpHiggs_mhs50_mzp1000_mDM100/ZpHiggs_ms50_mzp1000_mDM100/
   Events/run_01/tag_1_pythia_events.hep.gz as mZp_1000
import /users/bp15067/storage/generator/mg_desy/results/
   ZpHiggs_mhs50_mzp2000_mDM100/ZpHiggs_ms50_mzp2000_mDM100/
   Events/run_01/tag_1_pythia_events.hep.gz as mZp_2000
import /users/bp15067/storage/generator/mg_desy/results/
   ZpHiggs_mhs50_mzp500_mDM100/ZpHiggs_ms50_mzp500_mDM100/
   Events/run_01/tag_1_pythia_events.hep.gz as mZp_500

# declare the imported files as signal/background
set mZp_1000.type = signal
set mZp_1000_bb.type = signal
set mZp_2000.type = signal
# apply a cut
select (j) ETA < 2.0
plot N(b) 3 0 3 [superimpose]
plot N(l) 3 0 3 [superimpose]
# choose number of b/lepton jets you want
select N(b) = 1
select N(l) = 0
plot MET 20 0 1000 [superimpose]
# apply another cut
reject MET < 500
```

```
plot M(b[1]) 9 10 100 [superimpose]
#reject M(b[1]) > 80
#reject M(b[1]) < 40
plot PT(b[1]) 10 0 1000 [superimpose]
plot ETA(b[1]) 10 0 2 [superimpose]
# set aesthetics
set selection[2].logY = true
set selection[3].logY = true
set selection[6].logY = true
set selection[11].logY = true
set selection[12].logY = true
submit MonoDarkHiggs_signal
```

Listing 13.1: A MadAnalysis input file for a dark Higgs search based on the information in [39]. Output files from MadGraph can be imported to use as signal/background, cuts can be applied, and graphs can be plotted. File name: MonoDarkHiggs_signal_esh.ma (VI).

You import the signal/background files, apply cuts, plot graphs/histograms and get information (usually in the form of a LaTeX document) out of it. The paper that this search is based on (that I *need* to read) is [39]. A helpful, brief tutorial can be found at https://madanalysis.irmp.ucl.ac.be/raw-attachment/wiki/Talks/FR2012_conte.pdf.

As an example I ran an input file, importing MadGraph output files (**..._pythia_events.hep.gz**) from the runs I performed earlier in this document – $pp \to t\bar{t}$, $pp \to W+$ jets, and $pp \to Z+$ jets – as backgrounds to see how well the program worked and how output was handled. A LaTeX document, displayed below, was produced containing the input, process details, and a histogram. Other output files (that can be used in other analyses) are also generated in the **<name of output>/Output/** directory.

The PDF (with the .tex and other LaTeX files) is always generated in the directory **<name of output>/PDF/main.pdf**.

As of now, this electronic version will serve as my main lab book. It's easier to type and edit on a computer than by hand, and it's much easier to include (versioned) code, figures and other inclusions this way, rather than printing them out and sticking them in. I may still write some stuff in my physical lab book but, to be honest, it's more of a hassle if I'm writing everything twice.

Chapter 14

Concepts, nomenclature and definitions in high energy physics (17/11/2016)

14.1 Experimental

– Jet: when a quark is produced in a collision, it hadronizes (to become colour-neutral). So another quark is created to pair with the lone quark. But because the particle typically has a lot of energy, more quark-antiquark pairs are produced (the same as when a quark pair is pulled apart, the stored energy in the flux tube between them allows the pair production of quarks, such that there are now two pairs of quarks). This cascades and several quark pairs are produced rapidly, turning into a jet (as each new pair produced travels in the same direction as the original quark(s)).

– b-tagging: when bottom (b) quarks are produced in accelerator collisions, they hadronize and form jets. The hadrons composed of b-quarks have a relatively long lifetime and decay in the detector (or decay between the primary vertex – initial interaction point – and the detector to create a second jet and a secondary vertex). Because b-quarks have a large mass, their decay products tend to have a high p_{T}. This leads to high-multiplicity, wide jets. These properties allow the jets to be "tagged" and identified, which can be useful when studying specific decays that would likely produce b-quarks.

– Pseudorapidity: the pseudorapidity η is a spatial coordinate that describes the angle of a particle

(after a collision in an accelerator) relative to the beam axis. The formula is $\eta = -\ln[\tan(\theta/2)]$, where θ is the angle between the particle's three-momentum and the positive direction of the beam axis. So as $\theta \to 0^\circ$ (the beam line), $\eta \to \infty$. Generally, particles with large η escape the detector (because they are travelling so close to the beam line), so cuts on the order of $|\eta| < 5$ or tighter are usually applied during analysis.

– α_{T}: the kinematic variable α_{T} is used in analyses to suppress QCD backgrounds, and to suppress multijet production and mis-reconstructed p_{T}. It is defined in [40], and further described in [41]. Another form is

$$\alpha_{\mathrm{T}} = \frac{\sum_{i\in\text{jets}} E_{\mathrm{T}}^{i} - \Delta E_{\mathrm{T}}}{2\sqrt{\left(\sum_{i\in\text{jets}} E_{\mathrm{T}}^{i}\right)^{2} - (H_{\mathrm{T}}^{\text{miss}})^{2}}} \tag{14.1}$$

"where $E_{\mathrm{T}}^{i}(\equiv E^{i}\sin\theta^{i})$ is the jet transverse energy, and ΔE_{T} is the difference in E_{T} of the two *pseudo-jets*, two sets of jets into which the jets in the event are combined so as to minimize ΔE_{T}" [42].

– Parton Distribution Function (PDF): a parton distribution function details the momentum distribution of the partons (quarks and gluons) within a proton. There exist many PDFs, that depend on factors such as the energy and the processes that are occurring. See http://www.scholarpedia.org/article/Introduction_to_Parton_Distribution_Functions and [43].

– MiniAOD: mini-analysis object data (miniAOD) is a data format used to store data for CMS analyses, and so called because it dramatically reduced the size of the data files. This meant that computing resources and disk space became less strained. See [44] for more information.

– LHE: The Le Houches Accords was an agreement between particle physicists that standardised the interface between the matrix element programs (e.g., MadGraph) and the event generators (e.g., pythia) used to calculate different quantities. Events that conform to the formats described in the Les Houches Accords are said to be in the Les Houches Event (LHE) format.

– Loose/medium/tight lepton: this type of identification of a lepton refers to the "working points" (cuts) that are applied. The "tighter" the particle, the more (numerous and/or restrictive) cuts are applied. This leads to a lower cut flow efficiency but more accuracy. Usually, the more problems that backgrounds pose, the tighter the cuts are made in order to suppress them. See https://twiki.cern.ch/twiki/bin/viewauth/CMS/SWGuideMuonIdRun2#Muon_Identification and https://twiki.cern.ch/twiki/bin/viewauth/CMS/

CutBasedElectronIdentificationRun2#Electron_ID_Working_Points_WP_de for more information.

– Control region: orthogonal to the signal region, so no signal is expected. Control regions are used to see if backgrounds are well understood. So you form a "background-only" hypothesis (i.e., no signal – in the case of Higgs boson searches, the hypothesis would be that pp collisions would produce everything we've seen previously in colliders with no hint of a Higgs, because we hadn't detected the Higgs yet) and predict what your data *should* look like in the regions that aren't the signal region. Then if your data in the control regions agree with your background-only hypothesis, you can build confidence that you're estimating the backgrounds correctly and not introducing any unexpected bias.

– PromptReco and ReReco:

– ΔR: the variable ΔR is a measure of the angular separation between particles. It is commonly used to define how isolated a particular particle is by constructing a cone of radius ΔR (the apex of the cone being at the main interaction point, i.e., the primary vertex), calculated via

$$\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2} \tag{14.2}$$

A value of $\Delta R = 0.4$ is commonly used for lepton (namely muon) isolation. Whilst most people say that it isn't a "cone", it's more like a cone in 3D space but a circle in the $\phi - \eta$ plane (which is how the trigger crystals and towers are arranged).

– Isolation/isolated lepton: particle isolation is defined by the fractional momentum contribution of other particles "near" the particle you're interested in. First the track of the target particle is identified. Then a cone of radius ΔR is constructed (with $\Delta R = 0.4$ being a common choice for muons) around a point in the track with its apex at the primary vertex, and the sums of the p_{T} and E_{T} of all the particles within this cone are calculated. Then by dividing by the target particle's p_{T} you can calculate the fraction of the momentum that other particles are contributing to the total momentum inside the cone, which is also called the value of the particle's "relative isolation". Then you can impose cuts on these values, depending on how isolated you want a certain particle to be.

– Drell-Yan processes: in the LHC, the Drell-Yan (DY) process is when a quark from one proton annihilates with an antiquark from an oncoming proton. This normally produces one of the electroweak bosons, which decays into a lepton-antilepton pair. Looking at invariant mass plots from DY can show clear peaks at the masses of different particles, like the ϕ. J/ψ, υ hadrons

and the Z boson. However, as protons contain many quarks, antiquarks and gluons, DY is normally messy and the final states can include several jets.

– $\Delta\phi^*_{\text{min}}$: the variable $\Delta\phi^*_{\text{min}}$, known as "biased delta phi", is used in making event selections in signal/data, similar to α_{T} [45]. It is based on the minimum azimuthal separation between a jet and the negative vector $\vec{p}_{\text{T}}$ sum of all of the other jets in the event. So it is essentially the minimal separation between the jet and the $H_{\text{T}}^{\text{miss}}$ vector. It is calculated with the formula

$$\Delta\phi^*_{\text{min}} = \min_{\forall\, \text{j}_k \in [1, n_{\text{jet}}]} \Delta\phi\left(\vec{p}_{\text{T}}^{\,\text{j}_k}, -\sum_{\substack{\text{j}_i=1 \\ \text{j}_i \neq \text{j}_k}}^{n_{\text{jet}}} \vec{p}_{\text{T}}^{\,\text{j}_i}\right). \tag{14.3}$$

– Heavy flavour dark matter: these models include dark matter production in conjunction with heavy-flavour SM particles (top and bottom quarks). The final state products that could be detected would be those quarks (or their decay products) and $E_{\text{T}}^{\text{miss}}$ from the DM. Examples of these models can be found in [46]. In contrast, "light flavour" dark matter models yield a final state of dark matter particles and light quarks, i.e., up, down, charm, and strange.

– Systematic uncertainties: systematics are due to uncertainty from known sources. In particle physics analysis, these are from Monte Carlo statistical uncertainties, initial state radiation (ISR), jet energy corrections (JECs), etc., and need to be taken into account because they can affect the reliability of results.

– Look-elsewhere effect:

– Particle Flow: the reconstruction algorithm used by CMS. It essentially takes the elements (hits, clusters, tracks, etc.) from the different parts of the detector (tracker, ECAL, HCAL, etc.) and correlates them to build up the reconstructed event. See [47] for more information.

– Pileup: in colliders like the LHC, where the proton beams are split into bunches, there is an expected number of collisions/events per bunch crossing at a detector. If there are more than expected, these excess events are deemed as "pileup". At higher luminosities, more pileup events are expected, and so experiments have to find a way to sort the recorded objects correctly to determine which particles came from which event.

– Jet seed: the point of local maximum in a trigger tower (TT) sliding window of some size (currently 9x9 TT in CMS). The trigger tower with the largest energy deposit in the window is used as the seed (the centre of the jet), where the isolation is set by the size of the window. A jet seed threshold is usually applied to prevent recording numerous low-energy/pileup jets. Online Level-1 jets from data are clustered using this method.

- Trigger menu: the collection of trigger algorithms available in firmware at a given time.
- AK4 and AK8: the "AK" stands for "anti-k_{T}", which is a jet clustering algorithm widely used in data analysis from hadron-hadron collisions [48]. The 4 and 8 give the size of the jet ($R = 0.4$ and $R = 0.8$, respectively). In analyses, people sometimes refer to "AK4" or "AK8" jets, which are essentially jets clustered using the anti-k_{T} algorithm with the respective R value. So a larger R jet would be more inclusive if you have, say, a fat jet, but the isolation would be worse. This algorithm is used to cluster offline jets (Calo/PF/Gen).
- $E_{\mathrm{T}}^{\mathrm{miss}}$ vs. $H_{\mathrm{T}}^{\mathrm{miss}}$: $E_{\mathrm{T}}^{\mathrm{miss}}$ (or MET) is the missing transverse energy, i.e., the negative vector sum of the transverse energies of *all* particles in an event. $H_{\mathrm{T}}^{\mathrm{miss}}$ (or MHT) is the negative vector sum of the transverse energies (or momenta) of *only the jets* in an event. An easy way to distinguish is to think of the "H" in MHT standing for "hadronic".
- Skimmed and slimmed trees: a skimmed tree is one in which events have been removed that, for example, fail certain cuts. A slimmed tree is one where branches or leaves have been removed, i.e., redundant leaves that contain unnecessary information can be stripped to reduce the size of the tree.
- Monte Carlo truth level: "truth" level in MC refers to particles produced at the generator level, i.e., the base Feynman diagrams. Monte Carlo events are produced by creating the process/decay in a generator like MadGraph. These events are then hadronised and run through a detector simulation. Those two latter steps can potentially cause problems or issues, so sometimes it's better to look at the particles at the generator level and figure out what's going on. Trees with MC events usually store the gen. particle information so it's not too difficult to access.

This list will be updated whenever I encounter a new definition and so isn't representative of my knowledge at the time this section was first written (17/11/2016). Some of these explanations are pulled directly from Wikipedia (or have been worded a bit differently than the Wikipedia entry). A glossary of acronyms and various terms in CMS, computing, and high energy physics in general can be found at https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookGlossary.

As an aside, below is a cross section of CMS. It's useful for scale and to see the tracks of particles.

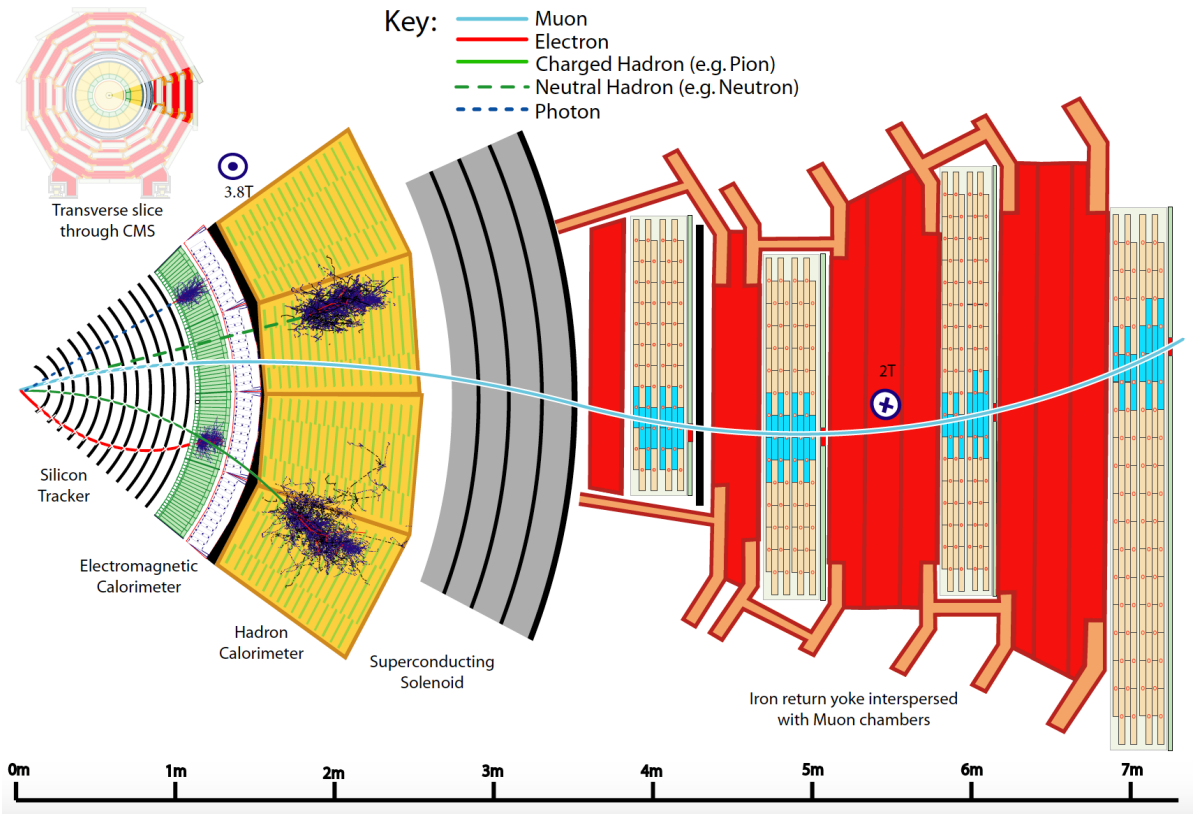

Figure 14.1: A transverse slice through CMS, taken from [47].

14.2 Theoretical

– Spinors: vector-like mathematical objects. In particle physics, spinors are associated with the spin of a particle. In QED and QCD, an object formed of two 2-component spinors (effectively spin-up and spin-down states) are solutions to the Dirac equation. For example, the spinor $u_r(\mathbf{p})$ is a positive energy fermion solution of the form

$$u_r(\mathbf{p}) = \mathcal{N} \begin{pmatrix} \chi_r \\ \frac{\boldsymbol{\sigma}\cdot\mathbf{p}}{E+m}\chi_r \end{pmatrix} \tag{14.4}$$

in compact notation. $\mathcal{N}$ is a normalisation factor, $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \sigma_3)$ are the Pauli matrices, and χ_r $(r = 1, 2)$ are two-component spinors that cover the spin degrees of freedom for fermions:

$$\chi_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } \chi_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{14.5}$$

See [49] for more information.

– Tree level: the process that contributes most to an interaction in a Feynman diagram. There are no loops or extra virtual particles created. It is the simplest interaction; most Feynman diagrams you see describing a specific interaction is shown at tree level.

– Loop level: this is one level deeper when looking at an interaction in a Feynman diagram. When looking at loop level, extra virtual particles are created and destroyed (in a loop). For example, in the Lamb shift – where the $2s$ and $2p$ energy levels in hydrogen aren't exactly degenerate as they should be – is due to the photon (mediating the interaction between the proton and electron) pair-producing two particles that distort the atomic potential and then annihilate to reform the original photon. This is an example of a one-loop effect.

– Yukawa coupling: used in the Standard Model to describe the coupling between a scalar and Dirac field. In the Standard Model, these can correspond to the interaction of the Higgs field with massless quark and lepton fields (i.e., the fundamental fermion particles). Through spontaneous symmetry breaking – due to the minimum of the Higgs potential being non-zero – these fermions acquire a mass proportional to the vacuum expectation value (vev, v) of the Higgs field:

$$m_f = \frac{y_f v}{\sqrt{2}} \tag{14.6}$$

where $f = l, q$ and m_f is the pole mass.

– On-shell and Off-shell: particles that satisfy classical equations of motion – such as the action, Euler-Lagrange equations, and Einstein energy-momentum relationship – are called "on-shell". These include real exchange particles, and any regular, detectable particles. Virtual particles do not obey the above equations and so are considered "off-shell".

– Covariant and contravariant: a contravariant vector is denoted with superscript indices (think "co↑travariant"), and its components transform in the same way as the coordinates but in the opposite way to the reference axes. Examples include position, velocity, and acceleration. A covariant vector is denoted with subscript indices (think "co↓ariant"), and its components transform in the opposite way as the coordinates but in the same way as the reference axes (i.e., scales with the coordinate system). An example is the gradient of a scalar field.

– Types of vector indices: Greek indices (μ, ν, etc.) are used to denote the components of a 4-vector, while Roman indices (i, j, etc.) denote the components of spatial 3-vectors.

– ∂_μ: this is the four-gradient used in tensor calculus. The operator is a four-vector, and so the result of using it on a variable gives a four-vector. As usual, $\mu = 0, 1, 2, 3$. From Wikipedia,

$$\frac{\partial}{\partial X^\mu} = \left(\frac{1}{c}\frac{\partial}{\partial t}, \vec{\nabla}\right) = \left(\frac{\partial_t}{c}, \vec{\nabla}\right) = \partial_\mu = {}_{,\mu} \tag{14.7}$$

The d'Alembertian,

$$\Box = \partial^\mu \partial_\mu = \frac{\partial^2}{\partial t^2} - \vec{\nabla}^2 \tag{14.8}$$

– Why the weak force is "weak": this is due to the Higgs field. In the very early universe ($\mathcal{O}(10^{-12})$ seconds), the Higgs field introducing mass to elementary particles broke a symmetry in the electroweak interaction (before, all particles, including the weak bosons, were massless). When the $W^\pm$ and Z bosons gained mass, their force-carrying ability became constrained by the Heisenberg Uncertainty Principle. As the particles mediating an interaction are virtual, they draw from the vacuum energy and so must "pay it back" in a time dictated by the Principle. The weak bosons are very massive, so require a lot of energy for them to pop into existence. And as they can only travel so fast, this limits both the range and strength of the weak interaction.

– Conservation laws as a consequence of symmetries:

– Quark confinement: single free quarks are never observed because of gluon-gluon interactions. This causes the field to concentrate into a flux tube connecting the quarks within a hadron. The tube stores potential energy and has an associated tension. If the quarks move further apart, the force stays roughly constant but the potential energy (force $\times$ distance) increases linearly with distance. When the potential energy is great enough, the flux tube spontaneously produces a quark-antiquark pair to leave two hadrons.

– Negative energy solutions to Dirac and Klein-Gordon equations: positive energy solutions are just positive energy particles moving with time. Negative energy solutions can be interpreted as negative energy particles moving backward in time, or positive energy antiparticles moving forward in time (because $e^{-i(-E)(-t)} = e^{-iEt}$).

– Gamma matrices: the gamma matrices are a component of the Dirac equation, which in itself is a generalisation of the Klein-Gordon equation that fixes some of its problems. These matrices (γ^i for $i = 0, 1, 2, 3$) are Hermitian, Traceless (where the sum of the elements on the leading diagonal is zero), and have eigenvalues of ± 1. They are normally written as 2x2 matrices, but each of these elements is itself a 2x2 matrix, i.e.,

$$\gamma^0 = \begin{pmatrix} \mathbb{1} & 0 \\ 0 & -\mathbb{1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} = \mathrm{diag}(1, 1, -1, -1) \tag{14.9}$$

The next three are defined as

$$\gamma^i = \begin{pmatrix} 0 & \sigma_i \\ -\sigma_i & 0 \end{pmatrix} \text{ for } i = 1, 2, 3, \text{ where } \sigma_i \text{ are the Pauli matrices} \tag{14.10}$$

For example,

$$\gamma^1 = \begin{pmatrix} 0 & \sigma_1 \\ -\sigma_1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{pmatrix} \tag{14.11}$$

There is also a fifth gamma matrix γ^5, which is useful when discussing the chirality of particles. It is defined as

$$\gamma^5 = i\gamma^0\gamma^1\gamma^2\gamma^3 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{14.12}$$

which, as well as having the same properties, anticommutes with the other four gamma matrices.

– Handedness of particles: the **helicity** of particles can be thought of as either "left-handed" or "right-handed", which is determined by the alignment of the directions of its spin and motion (linear momentum). If the direction of a particle's spin is the same as the direction of its momentum, the particle is said to be right-handed. If they are opposite, the particle is left-handed.

For massive particles – which must always travel below c – it is possible for an observer to boost into a reference frame where it overtakes the particle. In this case, before the observer overtakes,

the particle's helicity will appear one way. But then once overtaken, the particle's helicity will be reversed because the direction of its momentum relative to the observer will be reversed. For massless particles – which always travel at c – there is no reference frame in which its helicity can be reversed.

And so, for massless particles its helicity is fixed and therefore Lorentz invariant. For massive particles, helicity can change and so the property of **chirality** is needed. For massless particles, chirality is the same as helicity. But for massive particles, chirality is the Lorentz invariant property. For a Dirac fermion, it is defined by the γ^5 matrix operating on the particle's wave function. The chirality can be either left- or right-handed, but is fixed, regardless of reference frame. Chirality is important as the weak force only couples to left-handed fermions and right-handed antifermions (which violates parity).

– Hermitian conjugate: applies to matrices. The Hermitian conjugate of a matrix (usually denoted $A^\dagger$) is found by taking its complex conjugate and then transposing it. So, $A^\dagger \equiv A^{*T}$. If a matrix is labelled "Hermitian", $A^\dagger = A$.

– Unitary matrix: a matrix is "unitary" if its Hermitian conjugate equals its inverse, i.e., $A^\dagger = A^{-1}$.

– Special group: in group theory, a group is labelled "special" if the matrices within it have a determinant of $+1$.

– Z boson and photon as mixed states: in the GWS model, the electroweak force (before symmetry breaking) contained four massless bosons (W^+, W^-, W^0 and B^0). The two neutral bosons mixed to give rise to the Z^0 and A (photon) with

$$Z^0 = W^0 \cos(\theta_W) - B^0 \sin(\theta_W),\ A = W^0 \sin(\theta_W) + B^0 \cos(\theta_W) \tag{14.13}$$

where $\theta_W = 28.7^\circ$ is the weak mixing angle, or Weinberg angle. [50]

– Kaluza-Klein theory: this is a field theory that unifies gravity and electromagnetism by extending general relativity to five dimensions (adding a new spatial dimension). Kaluza described his theory classically in 1921 [51], and Klein developed a quantum interpretation in 1926 [52]. As Kaluza's classical theory only extended GR, there were no free parameters. He introduced what he called the "cylinder condition", imposing that none of the components in the new five-dimensional metric depended on the fifth dimension. This meant that the field equations were simpler, and that standard GR could be recovered in four dimensions. The geodesic equation [53],

$$\frac{dU^\nu}{d\tau} + \tilde{\Gamma}^\mu_{\alpha\beta} U^\alpha U^\beta + 2\tilde{\Gamma}^\mu_{5\alpha} U^\alpha U^5 + \tilde{\Gamma}^\mu_{55} (U^5)^2 + U^\mu \frac{d}{d\tau} \ln\left(\frac{cd\tau}{ds}\right) = 0, \tag{14.14}$$

where $U^\nu \equiv dx^\nu/d\tau$ is the four-velocity

yields terms that correspond to standard results: the quadratic term in U^ν gives the 4D geodesic equation with some EM terms; the linear term in U^ν gives the Lorentz force law as long as the fifth-dimensional component of the five-velocity corresponds to electric charge. This means that electric charge is understood as motion along the fifth dimension. Klein proposed that this fifth dimension was very small and tightly curled up (compactified).

Some phenomenological aspects of KK are described in [54], along with dark matter candidates that may be accessible at the LHC. In models with universal extra dimensions (UED) with a TeV-scale extra dimension (the mass gap is on the order of the reciprocal of the compactification radius R), KK excitations of SM particles can be well-defined compared to their SM counterparts (which are the zero KK modes of their respective fields). With minimal universal extra dimensions (MUED), the first KK excitation of the photon is the lightest KK partner (LKP).

– Ultraviolet completion: UV completion refers to a quantum field theory that, either passes to a more general theory, or is applicable, beyond a cut-off energy that would otherwise apply [55].

– Initial state radiation (ISR): this refers to radiation (usually photons, gluons, electroweak bosons or a Higgs) emitted by particles before the primary collision. It can be interesting in searches for invisible particles, such as dark matter. If two quarks produce a Z' in the s-channel which decays into $\chi\bar{\chi}$, there are no visible particles that can be identified. In the case there's ISR, it recoils off the Z', boosting the system. The ISR can be detected which gives insight into the final state.

Chapter 15

Evaluating backgrounds in a dark matter search from a CMS Physics Analysis Summary (17/11/2016)

So that I can begin doing some proper analysis, I will be using a CMS Physics Analysis Summary (PAS) [12] as a template. The information in the paper will be used to create an analysis file to run in MadAnalysis. The paper states that 90% of the backgrounds in this type of dark matter search can be attributed to Standard Model decays into $W^{\pm}/Z$ + jet. So I've created an input file for MadAnalysis to apply the cuts given for the monojet analysis in the paper: $|\eta| < 2.5$, leading jet $p_{\mathrm{T}} > 100\,\mathrm{GeV}$, $E_{\mathrm{T}}^{\mathrm{miss}} > 200\,\mathrm{GeV}$, no leptons, no b-quarks. This way I can plot the backgrounds that I generated previously for $pp \to W+$ jets and $pp \to Z+$ jets (with some minor changes: removing the lepton p_{T} cut, and generating more events). Then we get an estimate of these backgrounds for the DM search – which will be enhanced with more events, etc., in the future – and can then start thinking about implementing the other backgrounds for this search. Then we can look for a signal. At the LHC, monojets are used most prominently to look for dark matter particles. This because when the protons collide and produce dark matter, the incoming protons can emit gluons or jets before the collision and recoil from the dark matter, so you get a boosted jet because the DM particles are heavy.

The only problem, at the moment, with generating my backgrounds is the weight assigned to each event. The paper uses 12.9 fb^{-1} of data, corresponding to a lot of events. With the MadGraph output that I've generated, I only ran 200,000 events for each process, corresponding to a luminosity $\mathcal{O}$(10 pb) [where the symbol $\mathcal{O}$ means "of order"]. So when I scale the luminosity in the MadAnalysis

input file to 12.9 fb^{-1}, it effectively scales the weight of each event by a certain amount to match that luminosity. With reliable data the weight of each event is ∼1, so the backgrounds I have generated at the time of writing are not suitable for detailed analysis. Obviously this will be rectified in the future, but for now I am just demonstrating the technique and the motivation behind the use of MadAnalysis and these data.

The input and output files are displayed below. The values I have after the cuts can be compared with those from the paper via the link http://cms-results.web.cern.ch/cms-results/public-results/preliminary-results/EXO-16-037/ under the "Additional Tables" heading. If I want to examine the .root file generated for each process that I'm importing, it is located in the path **<name of output folder>/Output/_<label given to imported file>/TheMouth.root**.

The input file **DMbkgmonoj.ma**:

```
set main.fastsim.package = delphes
set main.fastsim.detector = atlas
# luminosity in fb^-1
set main.lumi = 12.9
# set variable to normalize plots by
set main.normalize = lumi
set main.outputfile = "DMbkgmonoj.lhe"

# import the signal/background files
# W+jets and Z+jets account for 90% of total background in
   this DM process
import ./Input/ppWjets_pythia_events.hep.gz as ppWjets
import ./Input/ppZjets_pythia_events.hep.gz as ppZjets

# declare the imported files as signal/background
set ppWjets.type = background
set ppZjets.type = background

define l = l+ l- #e mu mu_isol

plot MET 20 200 1000 [logY]
#plot PT(j) 20 0 1000 [logY]

# apply cuts and plot histograms
```

```
# plot <variable> <N_NBINS> <X_MIN> <X_MAX>
# basically trial and error determining the number of bins
reject PT(j) < 100
plot MET 20 200 1000 [logY]

reject MET < 200
plot MET 20 200 1000 [logY]

reject ETA(j) > 2.5
plot MET 20 200 1000 [logY]

reject PT(l) > 0
plot MET 20 200 1000 [logY]

reject PT(b) > 0
plot MET 20 200 1000 [logY]

# Output folder
submit DMbkgmonoj
```

Listing 15.1: The input file to analyse the monojet backgrounds for the dark matter search in [12]. File name: DMbkgmonoj.ma (VI).

Then I created some C++ macros to analyse the file **TheMouth.root** so that I could get the y-values of each bin in the $E_{\mathrm{T}}^{\mathrm{miss}}$ histogram and compare them with those from the paper. Because the entries in each bin must be multiplied by its weight – given in the main.pdf file created during the MadAnalysis run, which is fine because I need to run MadAnalysis to generate the PDF and ROOT files before running the macros – I've included that value as a `const double` in the global header **MouthGlobal.h**. In the same file, I've added a `TString` that contains the directory to **TheMouth.root**. So if I have several background files, as in this case, I can just modify the string once and it will apply globally (as I only need one source file and one header file to analyse *any* **TheMouth.root** file). The commands are the same as with normal .root files; just open ROOT and type `.x execMouth.C`.

The "daddy" macro **execMouth.C**:

```
#include "TheMouth.C"
#include "MouthGlobal.h"
```

```
void execMouth() {

    TheMouth t;
    t.Loop();
}
```

Listing 15.2: The "daddy" macro used to analyse TheMouth.root. File name: execMouth.C (v1).

The global header file **MouthGlobal.h**:

```
// Global variables/constants/include files are defined here

// ROOT header files
#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TH2.h>
#include <TStyle.h>
#include <TString.h>

// Header file for the classes stored in the TTree if any.
#include "TClonesArray.h"
#include "TObject.h"

// C++ header files
#include <iostream>

// Constants for histogram initialization
const int N_BINS = 80;
const int X_MIN = 200;
const int X_MAX = 1000;

// The directory of the TheMouth.root file
TString MouthDirectory = "../DMbkgmonoj/Output/_ppwjets/
   TheMouth.root";
```

```
// Weighting for each event (value in main.pdf)
const double WEIGHT = 1381;
```

Listing 15.3: The global header file to include when analysing TheMouth.root. File name: MouthGlobal.h (v1).

The source file **TheMouth.C**:

```
#define TheMouth_cxx
#include "TheMouth.h"
#include "MouthGlobal.h"

void TheMouth::Loop() {
    if (fChain == 0)
        return;

    Long64_t nentries = fChain->GetEntriesFast();

    // Create histogram to be filled
    TH1F* bkg_MET = new TH1F("MET", "Background MET", N_BINS
   , X_MIN, X_MAX);

    Long64_t nbytes = 0, nb = 0;

    for (Long64_t jentry = 0; jentry < nentries; jentry++) {
        Long64_t ientry = LoadTree(jentry);
        if (ientry < 0)
            break;
        nb = fChain->GetEntry(jentry);
        nbytes += nb;

        bkg_MET->Fill(MissingET_MET[0]);
    }

    // Get the y-value of each bin in the histogram and
   print
    // k is the bin number
```

```
    for (int k = 1; k <= N_BINS; ++k) {
29          bkg_MET->GetBinContent(k);
            // Print the range the bin encompasses and the y-
    value of it
31          cout << "Entries for MET = " << X_MIN + ( (k - 1) *
    (X_MAX - X_MIN) / N_BINS )
                << "-" << X_MIN + ( k * (X_MAX - X_MIN) /
    N_BINS )
33              << ": " << bkg_MET->GetBinContent(k) * WEIGHT
                << endl;
35      }
}
```

Listing 15.4: The source file used to analyse TheMouth.root. File name: TheMouth.C (v1).

In the header file **TheMouth.h**, I've only included a few lines (the ones that need changing from their default when generated by ROOT). Some of the default "include" headers were also removed as they caused errors.

```
#ifdef TheMouth_cxx
2 TheMouth::TheMouth(TTree *tree) : fChain(0)
{
4 // if parameter tree is not specified (or zero), connect the
     file
  // used to generate this class and read the Tree.
6     if (tree == 0) {
         TFile *f = (TFile*)gROOT->GetListOfFiles()->FindObject
    (MouthDirectory);
8        if (!f || !f->IsOpen()) {
            f = new TFile(MouthDirectory);
10       }
         f->GetObject("Delphes",tree);
12
     }
14   Init(tree);
```

```
}
```

Listing 15.5: Lines 620–634 of the header file used to analyse TheMouth.root. File name: TheMouth.h (VI).

The table of contributions from the dominant background processes in the paper:

$E_{\mathrm{T}}^{\mathrm{miss}}$ (GeV)	**200-230**	**230-260**	**260-290**	**290-320**	**320-350**	**350-390**	**390-430**
$Z(\nu\nu)$ + jets	74003	40377	22353	13069	7988	6219	3548
$W(\ell\nu)$ + jets	60160	29703	14636	7708	4212	3012	1529
$Z(\ell\ell)$ + jets	990	441	194	91.3	45.6	26.1	11.5
Total	135153	70521	37183	20868.3	12245.6	9257.1	5088.5
$E_{\mathrm{T}}^{\mathrm{miss}}$ (GeV)	**430-470**	**470-510**	**510-550**	**550-590**	**590-640**	**640-690**	**690-740**
$Z(\nu\nu)$ + jets	2059	1268	725	516	417	252	145
$W(\ell\nu)$ + jets	860	481	283	163	129	68	37.4
$Z(\ell\ell)$ + jets	4.3	2.5	0.4	0.4	0.3	0.1	0.1
Total	2923.3	1751.5	1008.4	679.4	546.3	320.1	182.5
$E_{\mathrm{T}}^{\mathrm{miss}}$ (GeV)	**740-790**	**790-840**	**840-900**	**900-960**	**960-1020**	**1020-1090**	**>1090**
$Z(\nu\nu)$ + jets	97.1	61.3	60.9	31	20.1	14	26.9
$W(\ell\nu)$ + jets	24.4	14.1	12.8	6.4	3.6	2.9	4.3
$Z(\ell\ell)$ + jets	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
Total	121.6	75.4	73.7	37.4	23.7	16.9	31.2

Table 15.1: The number of entries in bins of $E_{\mathrm{T}}^{\mathrm{miss}}$ of the dominant backgrounds in [12]. I used a precision of one decimal point when copying these values.

The table of contributions from the background process I generated in MadGraph:

$E_{\mathrm{T}}^{\mathrm{miss}}$ (GeV)	200-230	230-260	260-290	290-320	320-350	350-390	390-430
Z + jets	72096	41305	17273	12767	9763	8261	3004
W + jets	56621	33144	9667	13810	5524	2762	4143
Total	128717	74449	26940	26577	15287	11023	7147
$E_{\mathrm{T}}^{\mathrm{miss}}$ (GeV)	430-470	470-510	510-550	550-590	590-640	640-690	690-740
Z + jets	2253	751	0	751	2253	0	0
W + jets	0	1381	1381	0	0	0	0
Total	2253	2132	1381	751	2253	0	0
$E_{\mathrm{T}}^{\mathrm{miss}}$ (GeV)	740-790	790-840	840-900	900-960	960-1020	1020-1090	>1090
Z + jets	0	0	0	0	0	0	0
W + jets	0	0	1381	0	0	0	0
Total	0	0	1381	0	0	0	0

Table 15.2: The number of entries in bins of $E_{\mathrm{T}}^{\mathrm{miss}}$ of the dominant backgrounds in the results from my MadGraph runs, multiplied by the weight given by the MadAnalysis output to account for luminosity.

The difference between my results and those from the paper:

$E_{\mathrm{T}}^{\mathrm{miss}}$ (GeV)	200-230	230-260	260-290	290-320	320-350	350-390	390-430
Z + jets	2897	-487	5274	393.3	-1729.4	-2015.9	555.5
W + jets	3539	-3441	4969	-6102	-1312	250	-2614
Total	6436	3928	10243	6495.3	3041.4	2265.9	3169.5
$E_{\mathrm{T}}^{\mathrm{miss}}$ (GeV)	430-470	470-510	510-550	550-590	590-640	640-690	690-740
Z + jets	-189.7	519.5	725.4	-234.6	-1835.7	252.1	145.1
W + jets	860	-900	1098	163	129	68	37.4
Total	1049.7	1419.5	1823.4	397.6	1964.7	320.1	182.5
$E_{\mathrm{T}}^{\mathrm{miss}}$ (GeV)	740-790	790-840	840-900	900-960	960-1020	1020-1090	>1090
Z + jets	97.2	61.3	60.9	31	20.1	14	26.9
W + jets	24.4	14.1	-1368.2	6.4	3.6	2.9	4.3
Total	121.6	75.4	1429.1	37.4	23.7	16.9	31.2

Table 15.3: The difference between my results and those from the paper (see Tables 15.1 and 15.2). The differences were calculated by subtracting my results from those in the paper, hence the reason for minus signs in some of the elements. When calculating the Z + jets difference, I combined the $Z(\nu\nu)$ and $Z(\ell\ell)$ values, then subtracted my results. The total is calculated by adding the absolute values (i.e., the modulus) of the values in the previous rows.

Note: uncertainties are not included. The paper *does* include uncertainties, but I'm not sure how to calculate the error in *my* results. My MadGraph process for Z + jets does not specifically include $Z(\nu\nu)$ or $Z(\ell\ell)$ so I'm assuming they are just combined in my results. In Table 15.3, the total is the sum of the *absolute* values (i.e., the modulus) of the elements in the rows above. It is interesting to note that, for the most part, the differences between my results and those of the paper are smaller for the Z + jets process(es) than for the W + jets process. This is most likely due to the weighting that MadAnalysis implements in my results to match the luminosity in the paper. The weighting for the W + jets events was 1381, whilst it was only 751 for the Z + jets events.

As I mentioned earlier, with my MadGraph results I have to multiply the events in each bin of the histograms by the weight given by MadAnalysis. However, because the weight is so high, each entry effectively contributes more, which is a problem in the low-entry limit (where I only have a few entries in a bin). So the accuracy will not be good because one event multiplied by the weight effectively gives lots of entries. So if I want to be accurate, I need to run more events so the luminosity is higher, so has to be scaled to a lesser degree to match the luminosity in the paper, therefore assigning a smaller weight ($\to 1$) for each event. Then the law of large numbers should give me smoother distributions and finer results.

15.1 Submitting jobs to Condor

HTCondor (sometimes called Condor, as is the case on Soolin and in the Bristol wiki documentation) is a framework used to run computing-intensive jobs. See https://en.wikipedia.org/wiki/HTCondor for more information. For us in Bristol, Condor can be thought of as the software on Soolin that executes and manages jobs that are submitted to DICE. So I must be ssh'd into Soolin and have the jobs I want to submit be there before I can submit them with Condor. At the moment, I will be using Condor to submit MadGraph runs with lots of events so that I can get reliable Monte Carlo backgrounds. To run a simple, single job, I need Bjoern's Python script **run_mg.py** (which can be found within the **submit_mg.tar.gz** attachment at https://wikis.bris.ac.uk/pages/viewpage.action?title=Phenomenological+Tools&spaceKey=pprg#PhenomenologicalTools-Batchsubmission). The code is displayed below:

```
#!/usr/bin/env python

import sys, getopt
import os

def main(argv):
        if len(sys.argv)<2:
                print 'run_mg_grid <initial string>'
                sys.exit(2)
        inputfile = sys.argv[1]
        idString = inputfile
        if not os.path.isfile(inputfile):
                print inputfile+" does not exist"
                sys.exit()


        idString = idString.replace('configs/', '')
        idString = idString.replace('.input', '')
        os.system('mkdir -p results/'+idString)
        os.system('cp '+inputfile+' results/'+idString+'/')


```

```
names=[]
fC = open(inputfile)
      for line in fC:
  line = line.replace("\n", " ")
  if ("output") in line and not ("loop_optimized_output")
 in line:
    names.append(line.split(' ', 1)[1].strip())
fC.close()

      f1 = open('templates/template.sh', 'r')
      f2 = open('results/'+idString+'/'+idString+'.sh', 'w
')
      for line in f1:
              line=line.replace('SAMPLE', idString)
              f2.write(line)
      f1.close()
      f2.close()


      f1 = open('templates/template.submit', 'r')
      f2 = open('results/'+idString+'/'+idString+'.submit'
, 'w')
      for line in f1:
              line=line.replace('SAMPLE', idString)
  if ("transfer_output_files") in line:
    f2.write("transfer_output_files = "+', '.join(names)+'
\n')
  else:
    f2.write(line)
      f1.close()
      f2.close()

      print('cd results/'+idString+'/; condor_submit '+
idString+'.submit; cd -')
```

```
52          os.system('cd results/'+idString+'/; condor_submit '
    +idString+'.submit; cd -')


54
if __name__ == "__main__":
56          main(sys.argv[1:])
```

Listing 15.6: Bjoern's script to submit MadGraph runs to the cluster using Condor. File name: run_mg_bjoern.py.

When the **submit_mg.tar.gz** file is unpacked, a folder is created called **submit_mg** containing the Python script, and a folder called **templates** that contains a shell script and submission file. These are displayed below.

/templates/template.sh:

```
#!/bin/bash
2 echo "== path == "
pwd
4 echo "== starting == "
ls -alrth
6 tar -xzf mg_image.tar.gz
echo "== done untaring == "
8 bin/mg5 SAMPLE.input
ls -alrth
10 echo "== done == "
```

Listing 15.7: The shell script for Condor submission of MadGraph runs. File name: template.sh (mg).

/templates/template.submit:

```
Universe   = vanilla
2 Executable = SAMPLE.sh
getenv = True
4 Arguments  =
Log        = SAMPLE.log
6 Output     = SAMPLE.out
Error      = SAMPLE.err
8 Request_Memory = 2000
```

```
request_disk = 3000000

should_transfer_files = YES
#when_to_transfer_output = ON_EXIT
transfer_input_files = SAMPLE.input, ../../mg_image.tar.gz
transfer_output_files = SAMPLE_mDM10_lambda100,
   SAMPLE_mDM100_lambda100, SAMPLE_mDM1000_lambda100
#output_destination = /users/bp15067/mg/results

Queue
```

Listing 15.8: The submission file for Condor submission of MadGraph runs. File name: template.submit (mg).

The **submit_mg** folder needs to be copied to my *Soolin* directory – but on **/storage**, which is designed to store lots of data – **/storage/eb16003/**. Then the input file for MadGraph, say, **ppWjets.dat**, needs to be copied to the folder **submit_mg/configs**. Its file extension also needs to be changed from ".dat" to ".input". Once this is done, I need to create a compressed copy of my MadGraph set up by going to the **/users/eb16003/MadGraph/MG5_v2_4_3** directory and typing

```
tar -cvzf mg_image.tar.gz --exclude=mg_image.tar.gz --
   exclude=results/* *
```

This will create a tar ball called **mg_image.tar.gz**. It is an image file that the Condor script copies to the machines on DICE so it knows what program to execute in conjunction with my input file(s). This image file then needs to be moved to the **submit_mg** folder.

I also need to make sure I've sourced CMSSW so that I have a ROOT environment on Soolin (so that the output files I need are generated). In my home directory, I have a shell script called **cms.sh**, that I just need to `source`. Once these steps are complete, I can submit the job to Condor by typing

```
./run_mg.py configs/ppWjets.input
```

To check the status of the job, type `condor_q eb16003`. I don't think it tells you when the job is complete, but when it has finished the output is located in **submit_mg/results/ppWjets**. Then the entire MadGraph output can be accessed like normal.

I should be submitting jobs with around 50,000 events each and then running multiple jobs to get lots of output files (and therefore lots of events in total). Then I need to combine them into one. If they are .root files, I can use the `hadd` command if I've sourced ROOT:

```
hadd <merged file name>.root <input1>.root <input2>.root ...
```

But I'm not sure if I can combine the **tag_1_pythia_events.hep.gz** files from MadGraph into one file and then analyse them in MadAnalysis, as opposed to running MadAnalysis for each .hep.gz file and *then* combining the output .root files into one. If possible, I need to write a script to create lots of input files and append an index to them, e.g., **job1.input**, **job2.input**, etc., and also a script that will allow me to submit all the input files (each as one job) at once.

UPDATE: I wrote a Python script that allows me to create lots of input files with an index appended to the file name and the MadGraph output destination. It is displayed below.

```
# This script generates several indentical input files, but
    with an index in the file name and the intended output
    file. This is useful for running several similar MadGraph
     jobs on a cluster.

totalEvents = 1000000
# Number given to variable 'nevents' in MadGraph
eventsPerJob = 50000
# N_JOBS must be an integer
N_JOBS = totalEvents / eventsPerJob

for i in range(1, N_JOBS + 1):
    # Create a string with the value of i
    str(i)
    # Open ppWjets<value of i>.input
    jobName = "ppWjets%s" % (i)
    jobFile = open("%s.input" %jobName, "w")
    # Write the MadGraph input
    jobFile.write("\
#*********************************************************\n\
#*                       MadGraph 5                      *\n\
#*                                                       *\n\
#*               *                       *               *\n\
#*                 *         * *         *               *\n\
#*                   * * * * 5 * * * *                   *\n\
#*                 *         * *         *               *\n\
#*               *                       *               *\n\
```

```
#*                                                                      *\n\
#*                                                                      *\n\
#*  The MadGraph Development Team - Please visit us at  *\n\
#*  https://server06.fynu.ucl.ac.be/projects/madgraph   *\n\
#*                                                                      *\n\
#**********************************************************\n\
#*                                                                      *\n\
#*              Command File for MadGraph 5                             *\n\
#*                                                                      *\n\
#*   run as ./bin/mg5  filename                                         *\n\
#*                                                                      *\n\
#**********************************************************\n\
import model sm\n\
\n\
# Define multiparticle labels\n\
define p = g u c d s u~ c~ d~ s~\n\
define j = g u c d s u~ c~ d~ s~\n\
define l+ = e+ mu+\n\
define l- = e- mu-\n\
define vl = ve vm vt\n\
define vl~ = ve~ vm~ vt~\n\
\n\
# Specify process(es) to run\n\
generate p p > W+ j\n\
\n\
# Output processes to MadEvent directory\n\
output %s\n\
launch\n\
pythia=ON\n\
pgs=OFF\n\
set nevents %d\n\
" % (jobName, eventsPerJob) )
```

```
57      jobFile.close()
```

Listing 15.9: A script to create several identical input files but with an index appended to the file name (for job submission with Condor). The files created are called ppWjets1.dat, ppWjets2.dat,..., ppWjets<N_JOBS>.dat. File name: writeinputfiles_mg.py (v1).

So now that multiple, identical input files have been created – the only differences being the indices at the end of the input and output files – I can begin to run multiple jobs at once. In the bash programming language that the Mac terminal (and most terminals) uses, the `*` character encompasses the objects that share the name of what is typed before the `*`. So if I write `for f in *; do echo $f; done` in the terminal, everything in the current directory will be printed (the `echo` command prints stuff to the screen). Or if I wrote `for g in cats*; do echo $g; done`, then everything in the current directory whose file name starts with `cats` gets printed. So I can apply this to Condor.

Because my input files all start with the common string "ppWjets" (see Listing 15.9), I can use the command `<blah> ppWjets*` to encompass all those input files. To run multiple jobs on Condor, simply type

```
for f in configs/ppWjets*; do ./run_mg.py $f; done
```

which assumes I'm still in the **submit_mg** directory, all my input file names start with `ppWjets` and are in the `./configs` folder.

A more compact list of things to do:

1. Get the **submit_mg.tar.gz** folder from the Internet (address listed above) and unpack it.
2. Copy the unpacked **submit_mg** folder to my storage directory on Soolin (**/storage/eb16003/**).
3. Create the MadGraph input file I want to run multiple jobs for.
4. Edit my **writeinputfiles_mg.py** script to match the MadGraph input file, making sure I've got `pythia=ON`; and the correct number of total events, events per run, and indices for each file. Then run the script in the terminal (`python writeinputfiles_.py`) to generate the numerous files.
5. Copy these files to the **submit_mg/configs** folder on Soolin.
6. Make sure I have the re-tarred version of MadGraph (containing the packages I need) **mg_image.tar.gz** in my **submit_mg** folder.
7. Source CMSSW so I have a ROOT (+ other) environment(s), which allows MadGraph to generate the output files I need. I already created a shell script containing the commands (see Sec. 7). I just need to go to my home directory on Soolin (**/users/eb16003**) and type `source cms.sh`.

8. Then, navigate to the **/submit_mg/** directory in the terminal, and run the command `for f in configs/<common name of input files>*; do ./run_mg.py $f; done` to submit the jobs to Condor.
9. Wait for the output to be generated and check everything ran correctly.

For each run in MadGraph, the seed for the random number generator – like the Monte Carlo I did when computing radiative transfer in my master's project – should be set to the default (0) so that it's automatically (randomly?) assigned. Because if the seed is the same for each input file, the string that the random number generator reads will be the same for each run, and I'll get the same output from each run (so one set of 50,000 events will be exactly the same as the next set of 50,000 events, etc.). If in doubt, the seed for a given run can be checked in **submit_mg/results/<name of output>/<name of output>/Cards/run_card.dat**. But because MadGraph is designed for batch processing, it should account for that and generate a (pseudo)random seed for each job.

15.2 Analysis of backgrounds

With the ability to easily create multiple input files and run them at the same time using Condor, analysing them is the next step. As I've done previously, I need to use MadAnalysis to analyse the **tag_1_pythia_events.hep.gz** files that are generated by MadGraph. I can get MadAnalysis to work on Soolin using the following commands (found at https://wikis.bris.ac.uk/pages/viewpage.action?pageId=94962075) when I log in to Soolin:

```
. /cvmfs/sft.cern.ch/lcg/views/LCG_latest/x86_64-slc6-gcc62-
   opt/setup.sh}
export HADOOP_CONF_DIR=/etc/hadoop/conf
```

I have added these commands to my **cms.sh** script so that the CMS environment and these commands are sourced together. So my full **cms.sh** script looks like this:

```
# Sets up ROOT environment using CMSSW
source /cvmfs/cms.cern.ch/cmsset_default.sh
#cmsrel CMSSW_7_6_3
cd CMSSW_7_6_3/src/
cmsenv
cd ~

# Sets up necessary environment for MadAnalysis
```

```
. /cvmfs/sft.cern.ch/lcg/views/LCG_latest/x86_64-slc6-gcc62-
    opt/setup.sh
# fix hadoop config
export HADOOP_CONF_DIR=/etc/hadoop/conf
```

Listing 15.10: The shell script that I run to source the CMS environment and the MadAnalysis packages when I log into Soolin. File name: cms.sh (v2).

Then to copy the **tag_1_pythia_events.hep.gz** files, I just navigate to **/storage/eb16003/** on Soolin and type

```
cp submit_mg/results/ppWjets<index>/ppWjets<index>/Events/
    run_01/tag_1_pythia_events.hep.gz MadAnalysis/MA5_v1_4/
    Input/ppWjets<index>.hep.gz
```

where my MadAnalysis set up is in the directory **/storage/eb16003/MadAnalysis/MA5_v1_4**, and the destination of the file is **MadAnalysis/MA5_v1_4/Input/**. This action renames the file to **ppWjets<index>.hep.gz** as well so I don't have to do it after copying. And, obviously, I replace `<index>` with the respective numbers 1, 2,..., N_JOBS. But, I do have to manually copy over the files for the time being, unless I can make a bash/Python script to use a `for` loop to copy them all over.

If MadAnalysis isn't working properly on my Soolin directory, I'll have to copy the necessary files over to my local machine and do it locally. It's not too much trouble; I first need to make sure I'm in my local home directory in the terminal (*not* on Soolin), and then `scp` the files over from Soolin. The command is

```
scp eb16003@soolin.dice.priv:/storage/eb16003/submit_mg/results/
ppWjets<index>/ppWjets<index>/Events/run_01/tag_1_pythia_events
.hep.gz ./ppWjets<index>.hep.gz
```

to save the files in my home directory. Or I could be lazy and copy *everything* from the **results** folder (which contains a lot of files). To do this, just type

```
scp -r eb16003@soolin.dice.priv:/storage/eb16003/submit_mg/
results ./
```

(just for reference, on Soolin it's **/users/**, on my Mac it's **/Users/**).

Once I have the necessary files, I need to create an input file to analyse everything. This will be sim-

ilar to Listing 15.1. But I can use the * trick and write `import ./Input/ppWjets* as ppWjets` rather than writing a new line for each file. Then I can write out the analysis commands like normal, and they will operate on all the input files as they are now collectively labelled as "ppWjets". This means that only one **TheMouth.root** file will be generated, meaning I only need to use my "TheMouth" macros once, and don't have to use `hadd`.

I will also run MadGraph for the $pp \rightarrow Z$ + jets process with 1,000,000 events (20 runs $\times$ 50,000 events run^{-1}). Then I will analyse both processes with MadAnalysis and be able to simulate the backgrounds better because of the greater amount of data I have.

After simulating 1,000,000 MadGraph events with each process, I then analysed them with MadAnalysis using the input file displayed below (under **1.1 Command history**). The weight for each event is still high (245 for W + jets, and 150 for Z + jets), but a factor of $\sim$5 lower than before (see Tables 15.2, 15.3).

The PDF from MadAnalysis after analysing the processes:

The histograms before and after the cuts:

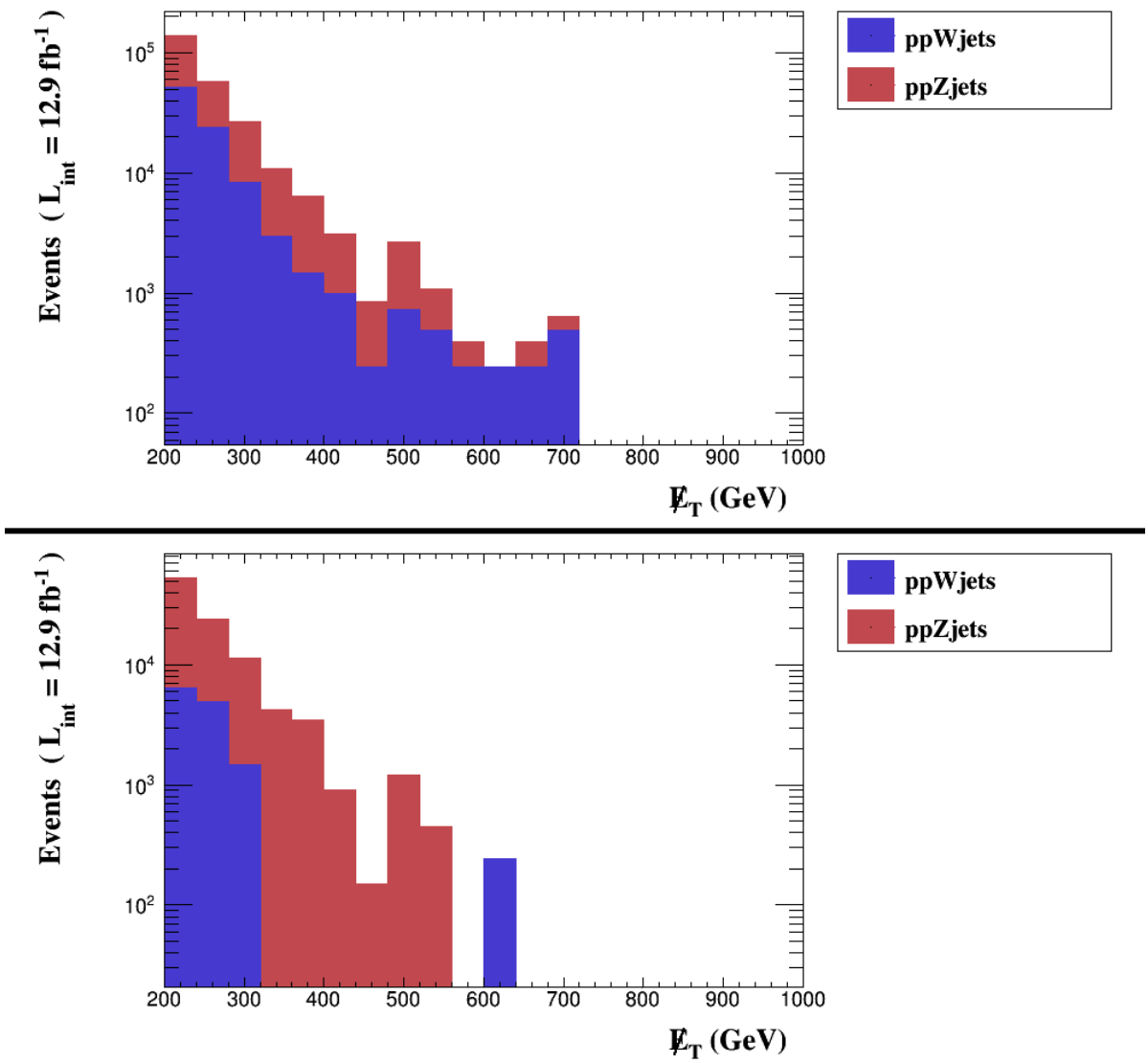

Figure 15.1: Histograms showing the number of weighted events against missing transverse energy for the background processes $pp \to W$ + jets and $\to Z$ + jets. The upper panel shows the events before any cuts, and the lower panel shows the events after several cuts: jet $p_{\mathrm{T}} > 100$ GeV, $E_{\mathrm{T}}^{\mathrm{miss}} > 200$ GeV, $|\eta| < 2.5$, no b-quarks, and no leptons.

Whilst these results look better, the problems are that the weights are *still* quite high and need to be $\mathcal{O}(1)$, and that high MET ($E_{\mathrm{T}}^{\mathrm{miss}}$) events are still under-populated even after running lots of MadGraph events (a problem with the law of large numbers – we need to simulate a **large number** of events). So to help fix the latter issue, I will try running MadGraph with the same processes but including jet HT (H_{T}, the scalar sum of transverse momentum) cuts in 200 GeV intervals to produce these high MET events. So I need to produce sets of results with jet HTs of 200–400, 400–600, 600–800, and 800+ (GeV). In the MadGraph input file, I'll need to use the command

`set htjmax <value>` to get a minimum HT (default is 0), and `set htjmax <value>` for a maximum HT (default is -1.0, meaning no maximum). And the syntax to plot/make cuts on HT in the MadAnalysis input file is `THT`. I can also check the list of variables/observables in **<MadAnalysis directory>/madanalysis/observable/observable_list.py**.

These results will need to be handled independently from the other processes (with different HT cuts). I'll need to document the cross sections and weights so I know how I need to include/combine everything. I'm using HT cuts rather than straight MET cuts in the MadGraph input files because HT is an easier variable to deal with than MET. When analysing output from implementing MET cuts – instead of HT – with the same ranges, the results were weird and overlapped when they shouldn't have.

So this is the procedure as a more compact list:

1. Copy the MadGraph output files (**tag_1_pythia_events.hep.gz**) from **/storage/eb16003/submit_mg/results/** to my MadAnalysis directory – renaming them to something more appropriate – and consolidate them in the directory **/storage/eb16003/MadAnalysis/MA5_v1_4/Input/**.
2. Write an input file for MadAnalysis to analyse the .hep.gz files, using Listing 15.1 as a template. Pay attention to the files I'm importing, the cuts, and the histograms I want plotted. The cuts described in that file are applicable to this analysis because I'm still modelling the backgrounds from the dark matter search in [12].
3. Make sure I've sourced my **cms.sh** script to set up the necessary environments.
4. Run MadAnalysis (with the `-s -R` commands to ensure the creation of the .root file) with the input file.
5. Open the PDF in **/MA5_v1_4/<name of output directory>/PDF/main.pdf** to get information about the weight assigned to the events in each process, the histograms, and the cut flow. I can't open it on Soolin, so just `scp` it to my local machine and use Adobe Reader.
6. My macros won't work on the version of ROOT I have on Soolin for whatever reason, so `scp` the **TheMouth.root** files to my local machine (renaming them to something appropriate to avoid confusion). Then modify my local copies of **MouthGlobal.h** (see Listing 15.3) in **/Users/eb16003/PhD_Software/MadAnalysis/MA5_v1_4/Macros_TheMouth/** to point to the directory containing the renamed **TheMouth.root** file of the process I want to look at, change the variable `weight` to the value given in the PDF, and make sure `N_BINS`, `X_MIN` and `X_MAX` are the correct values for what I'm looking at.
7. Run **execMouth.C** to print the *weighted* events in each MET bin for the **TheMouth.root** file I want to look at. Then document the results in a .txt file or an Excel spreadsheet.

The results from running MadGraph and MadAnalysis with different HT ranges (as discussed in the previous paragraphs) are presented below, along with an input file – created by my **writeinputfiles_mg.py** script – as an example. These runs were implemented with 50,000 events and constrained HT.

One of the MadGraph input files:

```
import model sm

# Define multiparticle labels
define p = g u c d s u~ c~ d~ s~
define j = g u c d s u~ c~ d~ s~
define l+ = e+ mu+
define l- = e- mu-
define vl = ve vm vt
define vl~ = ve~ vm~ vt~

# Specify process(es) to run
generate p p > W+ j

# Output processes to MadEvent directory
output ppWjetsHT200-400_1
launch
pythia=ON
pgs=OFF
set htjmin 200
set htjmax 400
set nevents 50000
```

Listing 15.11: A MadGraph input file for the SM background process $pp \to W+$ jets with constrains of jet H_T. File name: ppWjetsHT200-400_1.input.

The MadAnalysis input file to analyse my output from MadGraph:

```
set main.fastsim.package = delphes
set main.fastsim.detector = atlas
# luminosity in fb^-1
set main.lumi = 12.9
# set variable to normalize plots by
```

```
set main.normalize = lumi

# import the signal/background files
import ./Input/ppWjetsHT0-200* as ppWjetsHT0-200
import ./Input/ppWjetsHT200-400* as ppWjetsHT200-400
import ./Input/ppWjetsHT400-600* as ppWjetsHT400-600
import ./Input/ppWjetsHT600-800* as ppWjetsHT600-800
import ./Input/ppWjetsHT800* as ppWjetsHT800

import ./Input/ppZjetsHT0-200* as ppZjetsHT0-200
import ./Input/ppZjetsHT200-400* as ppZjetsHT200-400
import ./Input/ppZjetsHT400-600* as ppZjetsHT400-600
import ./Input/ppZjetsHT600-800* as ppZjetsHT600-800
import ./Input/ppZjetsHT800* as ppZjetsHT800

# declare the imported files as signal/background
set ppWjetsHT0-200.type = background
set ppWjetsHT200-400.type = background
set ppWjetsHT400-600.type = background
set ppWjetsHT600-800.type = background
set ppWjetsHT800.type = background

set ppZjetsHT0-200.type = background
set ppZjetsHT200-400.type = background
set ppZjetsHT400-600.type = background
set ppZjetsHT600-800.type = background
set ppZjetsHT800.type = background

define l = l+ l- #e mu mu_isol

# plot observables before any cuts
plot MET 30 200 1500 [logY]
plot THT 30 200 1500 [logY]

# apply cuts
```

```
reject PT(j) < 100
#reject MET < 200
reject ETA(j) > 2.5
reject PT(l) > 0
reject PT(b) > 0

# plot after cuts
plot MET 30 200 1500 [logY]
plot THT 30 200 1500 [logY]

# Output folder
submit bkgtestHTall
```

Listing 15.12: The MadAnalysis input file to analyse my SM background processes with constrained H_T. File name: bkgtestHTall.input (VI).

The histograms from MadAnalysis showing the number of events vs. total hadronic H_T (THT in MadAnalysis syntax):

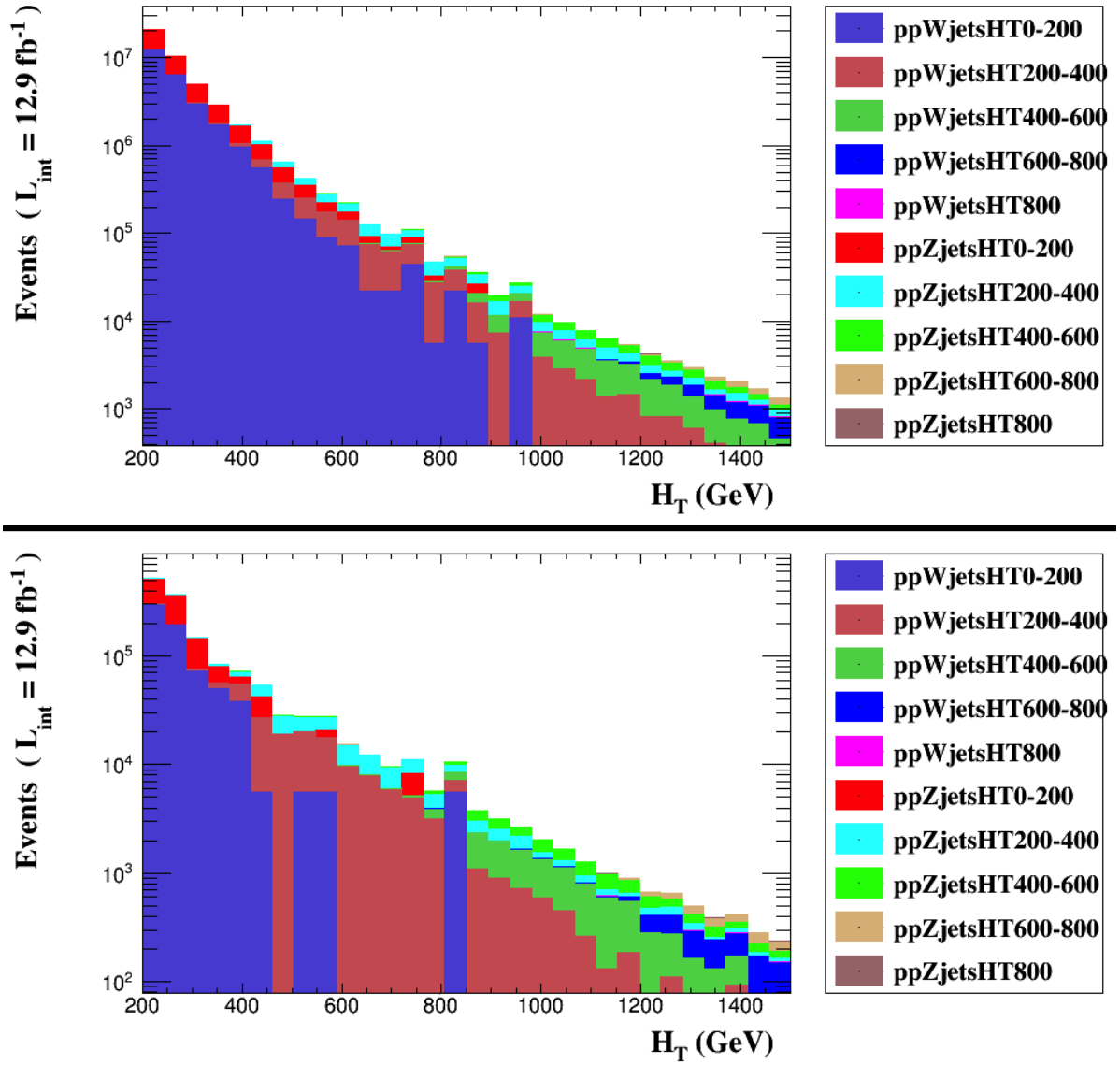

Figure 15.2: The number of weighted events vs. total hadronic H_{T} for the Standard Model background processes $pp \to W+$ jets and $Z+$ jets. These were generated by Monte Carlo with different ranges of H_{T}. The upper panel shows the events before any cuts, and the lower panel shows the events after several cuts: jet $p_{\mathrm{T}} > 100$ GeV, $E_{\mathrm{T}}^{\mathrm{miss}} > 200$ GeV, $|\eta| < 2.5$, no b-quarks, and no leptons.

The histograms from MadAnalysis showing the number of events vs. $E_{\mathrm{T}}^{\mathrm{miss}}$:

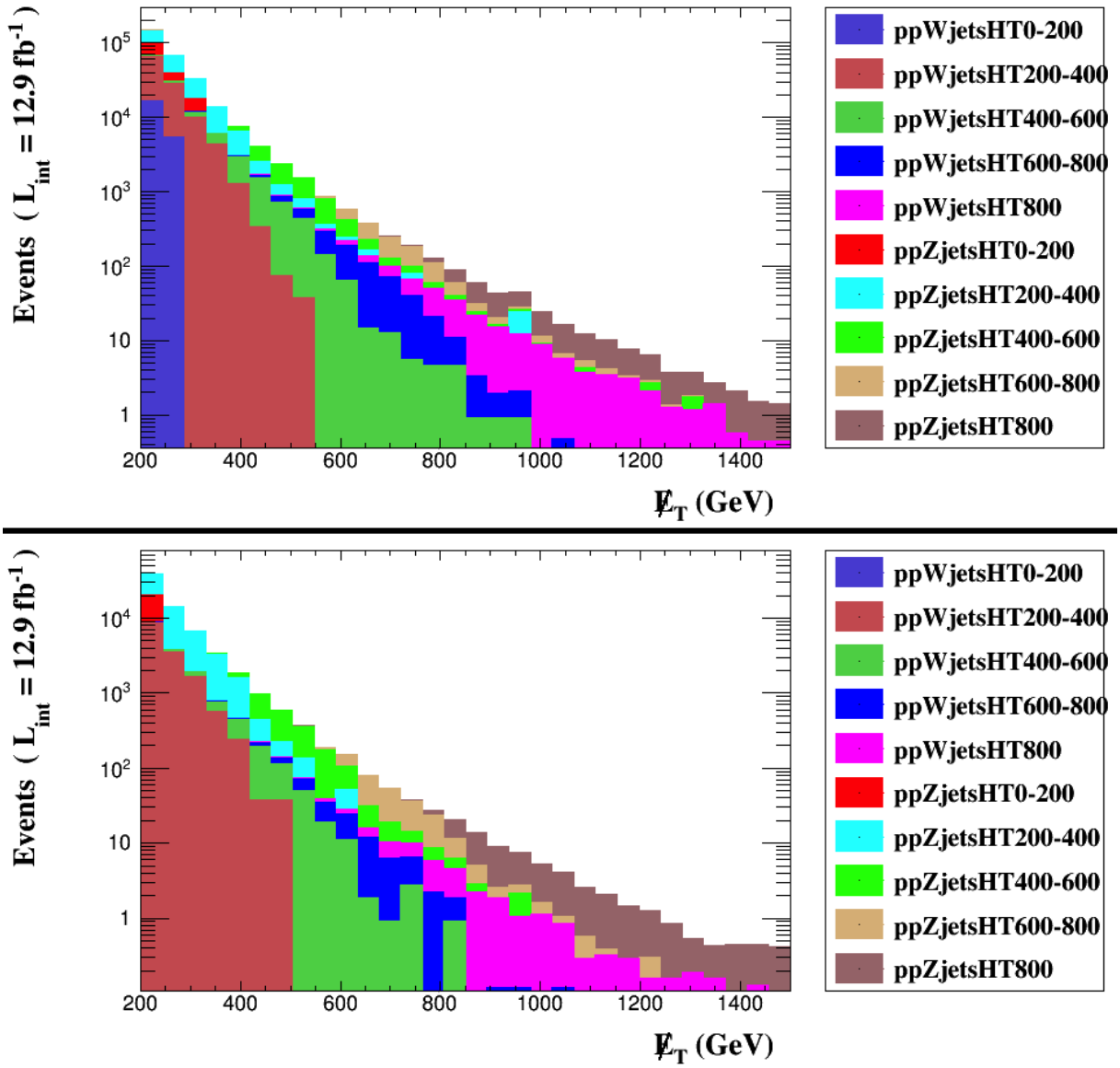

Figure 15.3: The number of weighted events vs. $E_{\mathrm{T}}^{\mathrm{miss}}$ for the Standard Model background processes $pp \to W+$ jets and $Z+$ jets. These were generated by Monte Carlo with different ranges of H_{T}. The upper panel shows the events before any cuts, and the lower panel shows the events after several cuts: jet $p_{\mathrm{T}} > 100$ GeV, $E_{\mathrm{T}}^{\mathrm{miss}} > 200$ GeV, $|\eta| < 2.5$, no b-quarks, and no leptons.

Some numerical data are displayed in the tables below.

$pp \to W+$ jets:

H_T range (GeV)	Generated events	Cross section (pb)	Weight ($\mathcal{L} = 12.9\,\text{fb}^{-1}$)
$0 - 200$	50,000	21381	5516
$200 - 400$	50,000	72.2	18
$400 - 600$	50,000	3.57	0.92
$600 - 800$	50,000	0.461	0.12
800+	50,000	0.124	0.032

Table 15.4: Information from the SM background process $pp \to W+$ jets, generated by Monte Carlo with different H_T ranges.

$pp \to Z+$ jets:

H_T range (GeV)	Generated events	Cross section (pb)	Weight ($\mathcal{L} = 12.9\,\text{fb}^{-1}$)
$0 - 200$	50,000	11612	2995
$200 - 400$	50,000	46.5	12
$400 - 600$	50,000	2.24	0.58
$600 - 800$	50,000	0.279	0.072
800+	50,000	0.072	0.019

Table 15.5: Information from the SM background process $pp \to Z+$ jets, generated by Monte Carlo with different H_T ranges.

I will be analysing these backgrounds in more detail (with more events) in the future. This subsection can be thought of as a guide and instructions on how to generate and analyse the Monte Carlo data, rather than a complete, comprehensive analysis in and of itself.

Final addition: a script to copy the output files from MadGraph to a destination (so that I don't have to manually copy everything) is displayed below.

```
# Copy files from a source to destination

from shutil import copy2, rmtree

master_process = "ppWjetsHT"
# String appended to name of each process to designate range
    /cut
common_appendage = "%s0-200" % (master_process)
# Number of jobs submitted to the grid for each
    master_process+common_appendage
```

```
N_JOBS = 4

for i in range (1, N_JOBS + 1):
    # Put i in string form
    str(i)

    file_string = "%s_%s" % (common_appendage, i)
    print "Copying %s" % (file_string)
    source_dir = "./submit_mg/results/%s/" % (file_string)

    # copy the file: copy2("<source>", "<destination>")
    copy2("%s/%s/Events/run_01/tag_1_pythia_events.hep.gz" %
    (source_dir, file_string), "./submit_ma/MA_Input/%s/%s.
   hep.gz" % (master_process, file_string) )
    print "Copied."

    rmtree("%s/" % (source_dir) )
    print "Original directory removed."

print "All done."
```

Listing 15.13: A script to copy files from a source to a destination. File name: copyfiles.py (v1).

Chapter 16

Rivet – another analysis tool (13/12/2016)

Rivet is another analysis tool, similar to MadAnalysis but more powerful. From the documentation (https://rivet.hepforge.org/) it seems more difficult to use but it means that I'll know exactly what's going on and it better suits the analyses I'll be doing.

I can download and install it – either locally or on Soolin – by following the instructions on the website. The input files required for Rivet are of the type .hepmc.gz, which are not produced by the Pythia6 module that my version of MadGraph contains. So, first I have to update MadGraph. The version I had been using up to this point was v2.4.3. An update was launched on December 10 (v2.5.2) which I need to update to. These are the instructions:

- (If I'm doing it on Soolin, I need to `source /cvmfs/sft.cern.ch/lcg/views/LCG_latest/x86_64-slc6-opt/setup.sh` first.)
- Go to my MadGraph directory and type `./bin/mg5`, then `install update`, then latest version will be installed. If there is a newer version that contains bugs, I should be able to find the v2.5.2 tarball on the internet.
- Once I've downloaded/updated to this version, I need to install hepmc [56], Pythia8 [57], Delphes, and lhapdf6 [58].
- (If on Soolin, open the file **input/mg5_configuration.txt** and type `lhapdf = /cvmfs/sft.cern.ch/lcg/external/lhapdfsets/current/` after the line "#! lhapdf-config".)
- Then once these are done, everything works as before, except in the input files replace `Delphes=ON` with `detector=Delphes`.

– When I run MadGraph, the files **tag_1_delphes_events.root** (which can be used to generate the "MakeClass" files with ROOT) and **tag_1_pythia8_events.hepmc.gz** should be created which can be used with Rivet.

Chapter 17

Searching for dark matter by identifying invisible (or semi-visible) jets (13/12/2016)

One method of detecting dark matter – that I'll probably be looking at in the foreseeable future – is from the production of invisible (or semi-visible) jets. If dark matter is a composite particle that can couple to Standard Model particles via a weak portal, these "hidden-sector" states could be produced (and then undergo a QCD-like shower) at the LHC. So a jet/spray of *stable* dark matter particles would be produced along with *unstable* states that decay into Standard Model particles. So if the production and decay of these particles are driven by a Z' (dark matter–Standard Model mediator) resonance that's leptophobic (doesn't couple strongly to leptons), the signature that you'd detect would be significant MET aligned roughly in the direction of a hadron jet. This is all explained well in [59].

These channels haven't been probed very well at the LHC so far because of the types of searches that have been, and currently are, conducted. The minimum azimuthal separation (difference in azimuthal angle) between jets is

$$\Delta\phi \equiv \min\{\Delta\phi_{j_1\ E_{\mathrm{T}}^{\mathrm{miss}}}, \Delta\phi_{j_2\ E_{\mathrm{T}}^{\mathrm{miss}}}\} \tag{17.1}$$

where $j_{1,2}$ are the two hardest (highest momentum) jets [59]. Typical searches require $\Delta\phi \gtrsim 0.4$, which is quite a bit larger than what are expected in the semi-visible jet channels. When including other cuts on, i.e., $E_{\mathrm{T}}^{\mathrm{miss}}$ and α_{T}, the efficiency in this type of search drops to low values such that

little data can be collected for interpretation. The horseshoe symbol $\supset$ indicates a superset. So the left-hand side of the formula is a superset of the right-hand side, i.e., the left-hand side includes, but is not limited to, the contents of the right-hand side. In the case of the Lagrangian, only the dark sector particles we are interested in are written, and it is implied that the full Lagrangian contains that, as well as more terms representing the other particles and interactions.

In this scenario, the dark matter is strongly coupled (i.e., couples to the strong force). One of the leading dark matter candidates, the WIMP, is weakly coupled. The paper compares sensitivity, $E_{\mathrm{T}}^{\mathrm{miss}}$ and $\Delta\phi$ between the two.

Chapter 18

Cut flow tables for SUS-15-005 (14/12/2016)

I've been asked by the CMS SUSY RA1 (Reference Analysis 1) team to produce cut flow tables to help them in their analyses. The cut flow efficiencies for some example benchmarks signals for their Run 1 paper [60] are detailed in Table 6 of the Additional Material at https://twiki.cern.ch/twiki/pub/CMS/SUS14006/SUS-14-006_aux.pdf. The team want this replicated for for their first Run 2 paper [45] (note that this is the *paper* CMS-SUS-15-005, not the *PAS* CMS-PAS-SUS-15-005). Details of the various benchmark models they consider are in Table 5 of the paper. My task is to produce cut flow tables for each of these benchmark models. They are useful in a number of ways: firstly, they are a requirement for all SUSY analyses; they help you understand the data better (if there are unexplained efficiencies from a cut, etc.); they help to optimise the data (you can see what cuts are removing lots of signal, etc., **and the main goal is to find a group of cuts that maximise the amount of signal you keep and minimise the background**).

I'll need to use Heppy (Python framework to run over EDM files, such as miniAODs), CMGTools (the "tree production code" that utilises Heppy), and AlphaTools (the "analysis code") in some capacity to produce these.

18.1 Configuring GitHub and CMSSW for CMGTools

For CMGTools I first need to subscribe to GitHub (https://github.com) and set it up for CMSSW. I deleted my old GitHub profile and made a new one for my uni account. My details are

– Name: Eshwen Bhal

– Username: eshwen

– Password: <normal password, normal numbers>

– User email: eshwen.bhal@bristol.ac.uk

– GitHub profile: https://github.com/eshwen

Then I can "fork" the following repositories by clicking the links and pressing "fork":

- **cmg-cmssw** – https://github.com/CERN-PH-CMG/cmg-cmssw
- **cmgtools-lite** – https://github.com/CERN-PH-CMG/cmgtools-lite
- **CMSSW** – https://github.com/cms-sw/cmssw
- **cmgtools-lite-private** – https://github.com/CMSRA1/cmgtools-lite-private

I've been added to the CMSRA1 organisation on GitHub (https://github.com/CMSRA1), which contains all the software I'll need. The repo cmgtools-lite-private is RA1's fork of CMGTools from CERN's repository.

The first time I set up CMSSW and git, there's quite a lot I need to do. I need to open the Terminal and type the following:

```
ssh soolin
source /cvmfs/cms.cern.ch/cmsset_default.sh
cd /storage/eb16003/CMScmg
export SCRAM_ARCH=slc6_amd64_gcc493
cmsrel CMSSW_8_0_20
cd CMSSW_8_0_20/src
cmsenv
```

give a reference to the relevant git repository on Soolin:

```
export CMSSW_GIT_REFERENCE=/software/SUSY/RA1/cmg-cmssw-bare
```

then add my git details and initialise for CMS:

```
git config --global user.name Eshwen Bhal
git config --global user.email eshwen.bhal@bristol.ac.uk
git config --global user.github eshwen
git cms-init
```

generate an ssh key to allow the machine access (also found at https://help.github.com/articles/generating-an-ssh-key/):

```
ssh-keygen -t rsa -b 4096 -C "eshwen.bhal@bristol.ac.uk"
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
```

```
cat ~/.ssh/id_rsa.pub
# Highlight and copy the key, then go to \url{https://github
   .com/settings/keys} to add it.
```

The rest of the installation instructions are found at https://github.com/CMSRA1/RA1OPS/wiki/1.-Flat-Tree-Production-(80X), mainly adding the relevant repos in the right directories and compiling the code. The "80X" is a reference to CMSSW_8_0_X, so there are different data sets that were analysed by different versions of CMSSW. After following to the end of the heading "Set up and enter an environment (in the first session)", I need to type the following commands to compile the software I've checked out, and push my version to my repository (so that if I edit files and cause some errors, they are contained and don't affect everyone else's version):

```
cd $CMSSW_BASE/src # the directory $CMSSW_BASE is "/storage/
   eb16003/CMScmg/CMSSW_8_0_20/" and is initialised at the
   command "cmsenv"
scram b -j 9 # the code compiles at this point
cd CMGTools
git remote add origin https://github.com/eshwen/cmgtools-
   lite-private/tree/80X-ra1-0.6.x/RA1
git remote add origin https://github.com/eshwen/cmgtools-
   lite-private.git
git remote set-url origin git@github.com:eshwen/cmgtools-
   lite-private.git
git checkout -b 80X-ra1-0.6.x_<some extension with date>
git push origin <branch>
```

Remember that the version numbers of CMSSW and those of the branches can change. I'll need to know which branch(es) to use for a specific analysis, especially if I'm using old data/Monte Carlo.

18.2 Flat tree production with CMGTools

To initialise CMSSW and git, and check out the branches I need *after* I've set everything up the first time, I need to do the following:

```
ssh -X soolin
source /cvmfs/cms.cern.ch/cmsset_default.sh
cd /storage/eb16003/CMScmg/CMSSW_8_0_20/src
cmsenv
```

```
cd CMGTools
git checkout <branch I've created for my analysis>
OUTDIR=/storage/eb16003/SUSY_RA1/<output path>
```

I've included these commands (except the first one, obviously) in a shell script **cmssw_git.sh**, which resides in my home folder on Soolin.

The **$CMSSW_BASE/src** directory contains CMSSW code which doesn't really need to be touched/edited. Then the **CMGTools/** directory is where the code I need is stored. Within it, the **RA1/** directory contains the analysis code, where trees/flat trees/ntuples are generated. This includes what variables will be contained in the trees, and the event selection. In the **cfg/** folder, there are several python files ending in **_cfg.py**. In any analysis searching for new physics there are 3 core collections of flat trees that need to be produced: data, Standard Model Monte Carlo, and Monte Carlo of new physics. So running, for example, **run_AtLogic_MC_SM_cfg.py** will produce the flat trees for Standard Model MC, and **run_AtLogic_MC_SUSY_SMS_cfg.py** will produce the flat trees for signal MC, etc. The "AtLogic" part of the file name means "alphat-logic", i.e., using the α_T variable.

Once I've sourced the script above, I need to continue following the tutorial using the link in the previous subsection, from the heading "Decide the output path". Under the heading "Start a proxy of a grid certificate", I had to set up my grid certificate(s) on Soolin. To do this, I could follow the instructions in 9.2. After that, to the directory **$CMSSW_BASE/src/CMGTools/RA1/cfg/** and carry on with the tutorial from where I left off.

When I check out my own branch, call it something like "eshAnalysis<date>". To see a list of available branches type `git branch`. To switch to a branch type `git checkout <branch name>`, and to create a branch to then checkout type `git checkout -b <new branch>`. If I am developing and make loads of mistakes, etc., I can delete all those changes and revert to the last commit by going to the top directory of the branch and typing `git checkout -*` Or I can undo the changes I've made to specific files by typing `git checkout - <path to file>`.

So now my complete bash script **cmssw_git.sh** looks like this:

```
source /cvmfs/cms.cern.ch/cmsset_default.sh
cd /storage/eb16003/CMScmg
cd CMSSW_8_0_20/src
cmsenv
cd CMGTools
# This is currently the branch I'm doing analysis on
git checkout eshAnalysis22122016
# This is currently my output directory
```

```
OUTDIR=/storage/eb16003/SUSY_RA1/data/80X/Data/20161222_S01/
# Start proxy of grid certificate (pass phrase is password I
    use for Soolin)
voms-proxy-init --voms cms --valid 168:00
cd $CMSSW_BASE/src/CMGTools/RA1/cfg
```

Listing 18.1: My bash script for initialising CMSSW and navigating to the correct git branch, and starting a proxy of a grid certificate. The branch I checkout and the path OUTDIR don't have to be fixed and can be changed depending on what I'm doing. File name: cmssw_git.sh (v2).

If I want today's date in the terminal, I just need to use the syntax `$(date "+%Y%m%d")` which prints the date in the format YYYYMMDD (i.e., the preferred format for dating trees and other stuff in the RA1 group). So if, for example, I wanted to include the current date in **$OUTDIR**, I just need to type

```
OUTDIR=/storage/eb16003/SUSY_RA1/data/80X/Data/$(date "+%Y%m%d")_S01/
```

18.2.1 Running a test and tagging the code

To run a test, do

`cd $CMSSW_BASE/src/CMGTools/RA1/cfg`

`heppy TEST_01 run_AtLogic_Data_cfg.py -N 100 -o test=1` ← runs the configuration for *data* on *100* events in one MiniAOD file. The output is in **./TEST_01/**.

A flat tree is created in the directory **TEST_01/HTMHT_Run2016H_PromptReco_v3_ DCSONLY/treeProducerSusyAlphaT/tree.root** (although the path up to the folder **/treeProducer...** can be different). I can then check the contents of the tree with ROOT. If successful, I can remove the **TEST_01** folder. Then I can tag the code in CMGTools and Heppy with:

```
git status # to check I don't have anything else to commit
echo $OUTDIR
TAG=$(echo $OUTDIR | rev | cut -d "/" -f 1-3 | rev | tr "/"
   "_")
git tag $TAG -m "$TAG"
git push ra1-private $TAG
pushd $CMSSW_BASE/src
git tag $TAG -m "$TAG"
git push ra1-private $TAG
```

```
popd
```

I can probably include these commands in a shell script as well if needed.

18.2.2 Submitting jobs and creating flat trees

To submit jobs to DICE using Soolin, I'll still need to be in the **$CMSSW_BASE/src/CMGTools/RA1/cfg/** directory. Then type

```
heppyBatchAlphaT.py -o $OUTDIR -c AtLogic_Data
```

for data. If I'm running over Standard Model Monte Carlo or Signal Monte Carlo, I can just replace the final argument with the relevant extension (`AtLogic_MC_SM`, etc.). These jobs are submitted via Condor so I can monitor and assess them like normal batch jobs. Once they have been prepared and run, I need to extract the output files with

```
heppy_untarcfg.py $OUTDIR
```

I can also check to see whether all the jobs have run correctly with

```
for dir in $OUTDIR/*; do (cd $dir; heppy_check.py *; cd -;) done
```

Information about resubmitting jobs is detailed in the flat tree production tutorial linked earlier in this section. Note that it is normal for some jobs to fail on the first attempt. I just need to keep resubmitting the failed jobs until they've all succeeded. To combine the output files, type

```
for dir in $OUTDIR/*; do (cd $dir; heppy_hadd.py . -c; cd -;) done
```

and then check if the ROOT files can be opened successfully:

```
shopt -s extglob
find $OUTDIR/*/!(Chunks) -name \*.root | xargs -L 1000 -t
   root -b -l
shopt -u extglob
```

Now the flat trees have been produced! The next step is to calculate the integrated luminosity by continuing to follow the tutorial. I don't need to do this if I'm creating trees for signal Monte Carlo, but do if it's data or SM MC. A tool called "BRILcalc" (**B**eam **R**adiation **I**nstrumentation and **L**uminosity **calc**ulator) is used to calculate the luminosity, but it only works on CERN's lxplus. So I would need to transfer files back and forth between that server and Soolin.

Finally, I need to copy the flat trees to **/hdfs** and Imperial (College London), delete the temporary files in **/storage** and inform Tai Sakuma/the RA1 group of the location of the new flat trees. The commands for these steps are

```
hadoop fs -copyFromLocal $OUTDIR /SUSY/RA1/80X/Data # for
   data
```

```
hadoop fs -copyFromLocal $OUTDIR /SUSY/RA1/80X/MC # for MC
rsync --exclude=cmgdataset.tar.gz --exclude=cmssw.tar.gz --
   exclude=cmsswPreProcessing.root --exclude=Chunks --
   exclude=failed --exclude=*~ -vuz -rltoD  --compress-level
   =9 ${OUTDIR%/}/ ebhal@lx01.hep.ph.ic.ac.uk:"/vols/cms/RA1
   /$(echo ${OUTDIR%/} | sed -e 's#^.*/\(.*/.*/.*\)$#\1#')"
nice -n19 rm -rf $OUTDIR
```

which is the same as on the Flat Tree Production primer, but without the -n option.

Once I've finished the tutorial, I should have enough information to produce flat trees. The list of analysers that are used in selecting events for a flat tree can be found in **CMGTools/RA1/python/buildSequence.py**. It is recommended to look at each analyser to understand what it does. In the file various things are imported. If I look on GitHub (or Soolin assuming I've cloned the correct repositories), the path to these files are as follows:

- `CMGTools.RootTools.RootTools`:
  CMSRA1/*cmgtools-lite-private*/RootTools/python/RootTools.py
- `CMGTools.TTHAnalysis.analyzers.susyCore_modules_cff`:
  CERN-PH-CMG/*cmgtools-lite*/TTHAnalysis/python/analyzers/susyCore_modules_cff.py
- `PhysicsTools.Heppy.analyzers.<etc.>`:
  CMSRA1/*cmg-cmssw-central*/PhysicsTools/Heppy/python/analyzers/<etc.>

The organisation is in **bold** and the repository is in *italics*. So entering a subdirectory is represented by a ".", with other Python scripts that are imported being assumed to reside in the **/python** folder. Hopefully these examples will allow me to easily find other imported files in the future.

18.3 AlphaTools

During the flat tree production, I should set up the main analysis framework AlphaTools. There's a README at https://github.com/cmsra1/alphatools that I can follow to accomplish this. There is more event selection in this stage of the workflow so an additional cut flow table implementation is needed. I've cloned the repository to **/storage/eb16003/CMScmg/AlphaTools/** because it was suggested to install AlphaTools *outside* of my CMSSW release. I've included the initialisation procedure in my **cmssw_git.sh** script so I don't have to do it manually; it's basically checking out my own branch for my analysis (v1.9.x-eshDevs), and sourcing **setup.sh** in the directory

AlphaTools/analysis/. The README that I should be following is slightly out of date as it references the branch "v1.6.x", when it should be "v1.**9**.x".

The 1.6.x branch was used in the 2015 analysis so that will be the branch I need. CMGTools/Heppy produce the flat trees, and then AlphaTools is used for further analysis and will be needed to create the cut flow tables.

18.4 One method of creating the cut flow efficiencies

18.4.1 Relevant files and functions

It is possible to find the tags (on GitHub) that were used for the production of the trees used in the SUS-15-005 analysis – this is detailed in the following subsubsection. Then I can check out these tags which will allow me to make the cut flow tables without remaking the trees. To produce trees, any of the files in **$CMSSW_BASE/src/CMGTools/RA1/cfg/** can be run, e.g., **run_AtLogic_Data_cfg.py** for data. The event selections in this config file are specified at line 66, i.e., the function `buildEventSelection_options`. This calls the function `event_selection_path_cfg_tree_production` constructed in **eventSelectionPathCfgDicts.py**, L52–109. (This and the following files are located in **$CMSSW_BASE/src/CMGTools/RA1/python/atlogic/**.) It is a nested combination of conditions combined with `All` and `Any`. If specified by `All`, the events need to satisfy all the conditions. If specified by `Any`, then at least one of the conditions must be met. The aliases (shorthands) for these conditions are stored in **eventSelectionAliasDict.py**.

The `dict()` constructor that's used is detailed at https://docs.python.org/2/tutorial/datastructures.html#dictionaries. All the cuts are stored in a dictionary, like in **eventSelectionAliasDict.py**. Each cut is in the format `'<name>': "<cut>"` – like `'ht40': "ev : ev.ht40[0] >= 200"` – where `<name>` is the shorthand you would type when calling the dictionary, e.g., `eventSelectionAliasDict['ht40']` would return `ev : ev.ht40[0] >= 200`. So the function `eventSelectionAliasDict` in the Python file acts as a database (like a .bib file containing loads of references) and you call specific entries for the cuts you want to apply (like doing `\cite{<reference>}` in a LaTeX document).

This Python dictionary – `event_selection_path_cfg_tree_production` – is given to the function `buildEventSelection` defined in **buildEventSelection.py**, L16–25. This function parses the dictionary and builds a module for event selection. For example, say

```
module = buildEventSelection(path_cfg =
```

`event_selection_path_cfg_tree_production)`

then `module` is built by the function. In each iteration of the event loop, the event object can be given to `module`, i.e., `module(event)`. If this returns `True`, the event passes the condition. If `False`, it doesn't pass.

The default `All` class is defined in **./EventSelectionModules/EventSelectionAll.py**, with an equivalent file for the `Any` class. Looking at L19–22:

```
def event(self, event):
  for s in self.selections:
    if not s(event): return False
  return True
```

it loops over the selections, giving the event object to each selection. It returns `False` if *any* selection returns `False`, and returns `True` if *all* selections return `True`. A user can develop an alternative version of this class and give it to `buildEventSelection()`. If this alternative version counts the number of events that pass and fail each selection, that will give the cut flow efficiencies. We will only need the `All` class and won't need a nested dictionary like the one found in **eventSelectionPathCfgDicts.py**; ours will be much simpler than that.

To push my edits to my remote repository, first go to **$CMSSW_BASE/src/CMGTools/** because that's where I've initialised my branch (called something along the lines of "eshAnalysis<date>") in my **cmssw_git.sh** script. Then add the necessary files using `git add <file(s)>`. The `*` character is valid to add all modified files.

Then commit them with `git commit` to commit the files I've edited *and* added. I'll be prompted to enter a commit message in Vim (so I'll need to type `i`, then write the message). When done type `<esc>:wq` to write, then quit. Or, instead use `git commit -m "<message>"`. This commits the files to the HEAD (i.e., the current branch), but they're not in my remote repository yet. I can also add the `-a` option in the command to commit all modified files such that the `git add` command isn't needed.

Finally, I can use `git push` to push those changes to the current branch, or `git push <remote> <branch>` to specify further. To get an updated version of a branch, first type `git fetch` which updates the metadata and changes made for everything in the repository and allows for new branches to be pulled. Then type `git pull <remote> <branch>`, which actually pulls the changes. A simple, good primer for git is http://rogerdudler.github.io/git-guide/.

18.4.2 Tagged code and tree locations

For a proper cut flow table, each selection needs to have some kind of ID, otherwise we cannot distinguish between different selections in the table. The selections used in the analysis for SUS-15-005 would be specified (in a particular order) in a dictionary. I just need to find the tags of the code used in the analysis, then I can find the dictionary used in it. These tags are written in a text file in the directory containing a tree. The trees used in the analysis are from 2015 and would be in **/hdfs/SUSY/RA1/74X/** on Soolin. The next step is to find the code with that tag and check it out. This would allow me to find the dictionary – and therefore the selections – used to produce the trees, giving me the cut flow. The analysis was conducted before cmgtools-lite became the tool the RA1 group uses, so the relevant files would still be part of the regular CMGTools. The Heppy tags are as follows:

– SUSY Signal models: 74X_MC_20151207_D01

– DM Signal models: 74X_MC_20151112_B01

– SM background MC: RA1cmg_SCF_v4.0_7414_01 (has the bdphi problem) (https://github.com/CMSRA1/cmg-cmssw-private/tree/RA1cmg_SCF_v4.0_7414_01)

– Data: 74X_Data_20151202_D01 (https://github.com/CMSRA1/cmg-cmssw-private/tree/74X_Data_20151202_D01)

– AlphaStats tag: v1.2.2_approval

– AlphaTools tag: v1.6.12_Approval_151210

To check out the tags for Data (as a test, as the cut flow efficiencies we need are for Monte Carlo), just follow the commands at https://gist.github.com/dsmiff/7cfe5821573ada85311319bc7c334eef. I've sourced the necessary commands in a shell script **load_tagged_data_cutflow.sh**, in my home folder on Soolin.

Because we're creating these cut flows for "benchmark models", they correspond to Simplified Model Spectrum (SMS) models. These correspond to SUSY particles decaying into specific particles with a 100% branching fraction, hence "simplified". So taking a look at Table 5 in the SUS-15-005 paper, the benchmark models are listed along with their properties. So, for example, the label T1qqqq is the pair production of gluinos with each gluino decaying into two quarks and a neutralino. So the T1 is the ID of the parent SUSY particle, and the qqqq are the Standard Model particles that are detected. Note that a neutralino – which is being classified as the LSP in this model – is always produced at the end of the decay chain and will be a source of MET. The numbers in brackets correspond to (m_{SUSY}, m_{LSP}) which are the mass of the parent SUSY particle (a gluino in this model) and the mass of the LSP. So flat trees were produced for these SMS models.

The simplest example of a cut flow is something like this:

```
buildEventSelection_options = dict(
  path_cfg = dict(All = (
    'ev : -2.5 < ev.jet_eta[0] < 2.5',
    'ev : ev.nJet40[0] >= 2',
)))
```

where the `'ev : -2.5 < ev.jet_eta[0] < 2.5'` is a selection. So I just need to find the variables that correspond to other selections (might be in **eventSelectionAliasDict.py**). Then implement the necessary selections in the right order to create the cut flow. I need to find out where these would be documented. I'll then also need to include some sort of counting variable to get the number of events that pass each cut (and then get a cumulative efficiency *after* each selection when all events are taken into account). Then this information would be used to construct the cut flow table.

The directories containing the trees used in the 2015 analysis are located (on Soolin) at:

- MC Trees: **/hdfs/SUSY/RA1/74X/MC/20151014_B01/20151014_SixCutflows_MC_25ns/**
- Data Trees: **/hdfs/SUSY/RA1/74X/Data/20151202_D01/20151202_run_susyAlphaT_AtLogic_Data_25ns_cfg/**

18.4.3 Dom's method to generate cut flow tables

I've pulled several repositories from Dom to get the code he's written that allow me to generate cut flow efficiencies. The necessary information is detailed in the shell script **load_doms_alphatools.sh** in my home folder on Soolin. The branch I'm performing the analysis on is "v1.6.12_Approval_151210_cutflow" or "v1.6.12_Approval_151210_cutflow_eshDev" in the AlphaTools repo. If I get an error when loading the script try `grep -r HEAD` to find conflicts, then resolve them before committing. If that fails or there are too many conflicts, delete everything and just do a fresh install by following this script:

```
# Initialise AlphaTools and switch to Dom's branch for 2015 analysis cut flow code
# First time set up:
source /cvmfs/cms.cern.ch/cmsset_default.sh
cd DomsAlphaTools
cmsrel CMSSW_8_0_26 # newer version so pandas library works
```

```
cd CMSSW_8_0_26/src
cmsenv
cd ../
git clone -o alphatools git@github.com:CMSRA1/AlphaTools.git
    --recursive
cd AlphaTools
git remote add dsmith git@github.com:dsmiff/AlphaTools.git
git fetch dsmith
cd analysis
source setup.sh
# Pull latest alphatools for verification - will checkout
   tag later
cd ..
git pull alphatools v1.10.x
git submodule update
git submodule init
# Checkout tag from 2015 result
cd $ALPHATOOLSDIR
git checkout v1.6.12_Approval_151210_cutflow
git checkout -b v1.6.12_Approval_151210_cutflow_eshDev
```

Listing 18.2: A shell script to download and install AlphaTools with Dom's branch needed to generate the cut flow tables for the 2015 analysis. File name: load_doms_alphatools_setup.sh.

Every relevant file from here is in **$ALPHATOOLSDIR/**, whose path is **/users/eb16003/DomsAlphaTools/CMSSW_7_4_3/AlphaTools/analysis/**. Some minor tweaking was needed, which is detailed in my script. Now I just need to go to **/SkimTreeProducer** and type

```
python SkimTreeProducer_PSet_cfg.py --data --pset=Signal --
   outDir ./
```

to produce the trees and cut flows for signal data (as a test), the output being stored in **SkimTreeProduction/Signal/<YYYY_MM_DD>_REV_1/Signal_Data/**. The cuts in **Sequences/NewDataSequence.py** are applied to produce a "skimmed" tree. The skimming procedure is defined in the config file **SkimTreeProducer/SkimTreeProducer_PSet_cfg.py**. In L38, the flag `makeCutFlowTable` is added which calls the relevant code to produce a cut flow table. We need to skim over the benchmark models which are located at Imperial College (so I need my IC account). The branch has been updated so it will, by default, skim over the T1bbbb benchmarks ($m_{\text{SUSY}}, m_{\text{LSP}}$)

= (1500, 100) and (1000, 800). The "T1" is the label for gluino pair production, and https://twiki.cern.ch/twiki/bin/view/LHCPhysics/SUSYCrossSections13TeVgluglu give the cross sections for this process at different masses. So I can use the table to get the cross sections for the T1bbbb benchmark processes. With the luminosity (defined in **Configuration/config_cfi_6CF.py**, L54) and this cross section, I can compare the expected number of events – using eq. 11.1 – to those in the cut flow table. More details about the models, variables, and the analysis in general for the paper can be found at https://github.com/CMSRA1/AlphaTDR2. See the wiki for details and instructions on how to compile the analysis note AN-15-004.

There are individual cut flow tables for each sample in a benchmark model, so I first need to combine them into one table. There's a script in **/Utils** called **makeCutFlowTable.py**. The arguments that need to be supplied are something like

```
python makeCutFlowTable.py --inputDir $ALPHATOOLSDIR/
   SkimTreeProducer/SkimTreeProduction/Signal/2017
   _01_25_REV_1/Signal_Data/ --process Signal_Run2015D_25ns
   -l -pdf
```

with the `-l` flag converting the .txt file into a LaTeX file, and the `-pdf` flag converting the LaTeX file into a PDF. For each event category, only the number of events that pass each cut is given. So to get efficiencies I need the total number of events. When I'm running the command to produce the raw cut flow tables, the number of events for each event category is given, so make sure I note them down. The order of the cuts are given in **Sequences/NewDataSequence.py**, so can be modified if need be.

Once I've got my IC account I should be able to run over all of the SMS models and generate the cut flow tables for all of them. It's a bit more complicated, but the gist of it is detailed here. The file **config_cfi_6CF.py** in **/Configuration/** contains the directories to all of the trees. L33 – `basedirSignalModels = "/vols/cms/RA1/74X/MC/T1bbbb_approvalReprise"` – is the path to the T1bbbb trees at IC. This assigns the path to the variable `psetSignalModels25ns`. So when running over samples, it will search for the samples under the given directory. The samples can be defined in **SkimTreeProducer/SkimTreeProduction.py**, L44 (the function `sampleSelection`). So if I want to run over, say, T1qqqq, I would need to point `basedirSignalModels` in **config_cfi_6CF.py** to the location of the trees for that model, *and* also define the samples in the `sampleSelection` function in **SkimTreeProduction.py**. I can then run **SkimTreeProducer_PSet_cfg.py** with a different pset argument: instead of `-pset=Signal`, use `-pset=SignalModels` to run over the samples I've targeted.

Once I'm on the IC remote server and have set up the necessary repos and branches, navigate to **$ALPHATOOLSDIR/SkimTreeProducer** and run

```
python SkimTreeProducer_PSet_cfg.py --mc --pset=SignalModels
    --outDir ./
```

to get the raw cut flow tables for the SMS model I'm targeting. I can then run **makeCutFlowTable.py**, making sure the arguments are correct, to produce the condensed cut flow table. I'll need to translate the skimmers into something more meaningful. By default, they'll be named bDPhiSkim and objectSkimmer, etc. But they are defined in the **Skimmers/** folder so I can figure out what actual cuts are and write them in. If there isn't a corresponding .py file for a particular skimmer, it might be defined in **HeppySkimmers.py**.

Step-by-step instructions:

1. Get my Imperial College account and log in to their remote server.
2. Set up AlphaTools using the **load_doms_alphatools.sh** script I have on Soolin, and pull the necessary repos and branches.
3. Edit L33 (the function `basedirSignalModels`) in **$ALPHATOOLS/Configuration/config_cfi_6CF.py** to give the path of the benchmark model I'm targeting.
4. Define the samples in L44 (`psetSignalModels25ns.sampleSelection`) of **$ALPHATOOLS/SkimTreeProducer/SkimTreeProduction.py**.
5. Produce the raw cut flow tables by staying in the same directory and typing `python SkimTreeProducer_PSet_cfg.py -mc -pset=SignalModels -outDir ./`. Note down the total number of events in each sample during the skimming so that I can create efficiencies.
6. Condense the cut flow tables for each sample by typing `python makeCutFlowTable.py -inputDir $ALPHATOOLSDIR/SkimTreeProducer/SkimTreeProduction/<location of cut flows> -process <name of sample> -l -pdf` (where <name of sample> is what I've defined in step 4).
7. Find the file corresponding to each skimmer by looking in **$ALPHATOOLS/Skimmers/**. Figure out what the actual cut is and then replace the default name of the skimmer with that.
8. LaTeX the table to make it look presentable and display it in the correct format (the models are defined in the paper SUS-15-005, and the symbols for the particles are *not* italicised). Use the total number of events for each event category to put all values in terms of percentages. By default, the efficiencies should be inclusive (cumulative). Also include a column for exclusive (cut-by-cut) efficiencies.
9. Repeat steps 6–8 for each sample in the benchmark model, then repeat steps 3–8 for each benchmark model.

Rob has asked whether the preliminary tables (for signal data) contain *all* the cuts applied in the analysis, i.e., those applied in Heppy, the skims, and AlphaTools. I think they do. There's a file called **HeppySkimmers.py** in **/Skimmers/** which contains a few classes that are translated to cuts. And all of the skims are in AlphaTools so I think *every* cut used in the analysis is present in the tables. All of these selections are detailed below – with translations if possible – and whether they're required for cut flows in the signal Monte Carlo (benchmark) models.

- <Skimmer name, as detailed in Dom's AlphaTools branch> | <event selection> | <necessary for benchmark model cut flows?>
- defaultSkim | α_{T} H_{T}-dependent cuts; $H_{\mathrm{T}}^{\mathrm{miss}} > 130$ (for jets with $p_{\mathrm{T}} > 40$); $H_{\mathrm{T}} \geq 200$ (for jets with $p_{\mathrm{T}} > 40$); n_{jets} with $p_{\mathrm{T}} > 40$ is > 1; n_{jets} with $p_{\mathrm{T}} > 100$ is ≥ 1 | Y
- bDPhiSkim | $\Delta\phi^{*}_{\mathrm{min}} > 0.5$ | Y
- objectSkimmer | Lepton and photon vetoes, depends on data type (signal MC, single mu, double mu, etc.) | N
- primaryDatasetSkimmer | no cut, specifies parent sample if data | N
- cutFlowSkimmer | determine type of data for cut flows (data, control region MC, signal region MC) | Y
- mllSkimmer | $66.2 < m_{\ell\ell} < 116.2$ | N
- minDRJetSkimmer | $\Delta R > 0.5$ | N
- JSONSkimmer | Checks if sample is MC, or if run/lumi pair is in JSON file if data | N
- badMCEventSkimmer | Returns True if sample is data. If MC, checks for "bad" events | Y
- relIsoSkimmer | Cut on relative isolation of leptons | N
- eleEtaSkimmer | asserts that $|\eta|$ for ALL electrons < 1.479 | N
- promptPhotonSkimmer | if sample is data, return True (no cut). If not data, want 0 photons | N
- ttJetsSkimmer | if sample is data, return True. If parent sample is not TTJets, return True. Else, lots of conditions | N
- photonPtSkimmer | cut on photon momentum | N
- triggerSkimmer | Trigger cuts (only affects data and control region MC) | N
- filterSkimmer | If signal (benchmark model), return True | N
- fwdJetSkimmer | Forward jet veto | Y
- tighterJetIdSkimmer | jet_chHEF $\geq$ 0.1 | Y

- leadJetEtaSkimmer | $|\eta^{j_1}| < 2.5$ | Y
- mhtDivMetSkimmer | $H_T^{miss}/E_T^{miss} < 1.25$ | Y
- OddJetSkimmer | if inclusiveJet.newId == 0 and inclusiveJet.pt > 40 : return False | N
- mtSkimmer | $30 < M_T < 125$ | N
- IsoTrackSkimmer | Isolated track veto | Y

For clarification: j_1 describes the leading jet; all values are assumed to be in GeV unless stated otherwise; some cuts are not needed because they apply only to data or to control regions. All of these variables and terms should be defined in [45].

18.5 Presentation/talk on cut flow tables progress

Below is the progress report I gave to the RA1 group regarding the progress we've made with creating the cut flow tables. For this, and future, talks, it is important to remember the following:

- Include name, email address, University of Bristol logo, CMS logo, and slide numbers in every slide.
- At the very least, include the date that I'm giving the presentation on the title slide. I can also include it in each slide as part of the footer.
- For consistency, create a "Slide Master" in PowerPoint when I'm creating the presentation (taking into account the first point in this list), and give it some name. Then each new slide can use that template.
- Don't have a "busy" background. It needs to be relatively plain so that the reader can focus on the information and won't get distracted by the background.
- Consider making the slides in the 4:3 aspect ratio rather than the default 16:9 as (some) projectors still use 4:3.
- Remember to credit the people who helped in the work/analysis, regardless of whether they helped to write the talk itself. Just follow the same rules as authorship in papers.

The link to the slides is

18.6 Configuring event selections using Tai's method

Tai has created another method that implements cut flows which could be used in future analyses, and possibly also the 2015 analysis if it can be perfected and streamlined quickly. The code, with

an example of how to run it, is located at https://github.com/TaiSakuma/cutflowirl. The command, using an example SMS model, is

```
./cutflowirl/twirl_mktbl.py --components
    SMS_T1tttt_madgraphMLM --max-events-per-process 500000 --
    logging-level INFO --parallel-mode htcondor
```

where `components` requires the sample(s) being run over, `max-events-per-process` is only required if using batch submission, and if `htcondor` doesn't work, I can set `-parallel-mode multiprocessing` instead). This yields a raw table in **./tbl/out/** in the following format:

component	depth	class	name	pass	total
*	1	+AllCount	All	39074	30799443
*	2	LambdaStr	"ev : ev.smsmass1[0] == 1300"	768345	30799443
*	2	LambdaStr	"ev : ev.smsmass2[0] == 1050"	39074	768345
*	1	LambdaStr	"ev : ev.nJet40[0] >= 3"	34486	39074
*	1	LambdaStr	ht40	31061	34486
*	1	LambdaStr	nJet100	22984	31061
*	1	+AnyCount	Any	22984	22984
*	2	LambdaStr	ht40	22984	22984
*	2	LambdaStr	nJet100	0	0
*	1	+NotCount	Not	0	22984
*	2	LambdaStr	ht40	22984	22984

Table 18.1: The output from Tai's example cut flow table generator. Symbols: * = SMS_T1tttt_madgraphMLM; + = EventSelection.

The first column displays the model that's being run over (T1tttt → gluino pair production decaying into four t quarks and two neutralinos). The "depth" refers to the nesting of some event selections, discussed below. The "name" column gives the names of the cuts. These are the conditions as expressed in the function that configures the event selections (`path_cfg`); some of these are actual Python code for the cuts, whilst others are aliases that are shorthand for the cuts described in the file **eventSelectionAliasDict.py**. Then the "pass" column shows the number of events that pass the condition, and the "total" column shows the total number of events left to iterate over (i.e., the events that have passed all of the previous cuts).

To conjure the table above, the `path_cfg` function at L81 of **twirl_mktbl.py** looked like this:

```
path_cfg = dict(All = (
```

```
  dict(All = ('ev : ev.smsmass1[0] == 1300', 'ev : ev.
   smsmass2[0] == 1050')),
  'ev : ev.nJet40[0] >= 3',
  'ht40',
  'nJet100',
  dict(Any = ('ht40', 'nJet100')),
  dict(Not = 'ht40'),
))
```

where the dictionary contains *one* element, whose key is `All` (discussed earlier) and value is encompassing everything from the second line (and is a tuple). Each line from the second to the penultimate line is a specific condition, i.e., an event selection/cut. So the first condition uses the `All` class – meaning that all of the conditions within it need to be satisfied to pass – and has two nested selections. Referring to the "depth" column in Table 18.1, it is clear now that the depth is 2 for these selections because there are two selections in this `All` class. The row above (EventSelectionAllCount) shows the number of events that pass *both* conditions, and the two rows below give a breakdown of the cut. This first cut actually refers to the target sample: the variables `smsmass1` and `smsmass2` are the masses of the parent SUSY particle (m_{SUSY}) and the LSP (m_{LSP}), respectively. So you're making sure you iterate over the correct model and sample before proceeding with the other cuts. For the EventSelectionAnyCount row the `Any` class is used, so any of the (two, in this case) conditions needs to be met. Notice that the nJet100 row shows 0 in the "pass" and "total" columns. This is because in the previous condition (ht40), all events pass (because the condition was called previously in the cut flow) so no events need to be passed to the next condition contained within the `Any` selection.

Using the `path_cfg` function above, I can construct the cut flow for a particular model and sample from the 2015 analysis by including the cuts Dom pointed me to (the skimmers in his AlphaTools branch), using **eventSelectionAliasDict.py** for the syntax of the cuts. I also have to make sure that I'm pointing to the correct model and sample I'm interested in each time I run the code. The absolute path to the model is defined in `default_heppydir`, at L17 of **twirl_mktbl.py**. Then I would just need to set `smsmass1[0]` and `smsmass2[0]` to the values pertaining to the sample I want to run over.

After including the skimmers applicable to the 2015 analysis (then removing the unnecessary skimmers and translating the remaining ones that are applicable to signal Monte Carlo, i.e., the benchmark models), the (more-or-less) finished cut flow table for the example model is below. Note that, for SUSY papers, the standard symbols for particles are from the package `hepnicenames`. The documentation can be consulted for clarifications on syntax.

Event Selection	Model (sample)		
	$\widetilde{g}\,\widetilde{g} \to t\,t\,\bar{t}\,\bar{t}\,\widetilde{\chi}_1^0\,\widetilde{\chi}_1^0$ ($m_{\mathrm{SUSY}} = 1300, m_{\mathrm{LSP}} = 1050$)		
	Events passed	Inclusive efficiency (%)	Exclusive efficiency (%)
–	39074	100	100
$n_{\mathrm{jet}} \geq 2$ ($p_{\mathrm{T}}^{\mathrm{j}} > 40$ GeV)	39074	100	100
Forward jet veto	35488	90.82	90.82
Jet failed ID check	35488	90.82	100
jet CHF ≥ 0.1	34605	88.56	97.51
$\lvert\eta^{\mathrm{j}_1}\rvert < 2.5$	34457	88.18	99.57
Isolated track veto	12017	30.75	34.88
$n_{\mathrm{jet}} \geq 1$ ($p_{\mathrm{T}} > 100$ GeV)	7971	20.40	66.33
$H_{\mathrm{T}} \geq 200$ GeV ($p_{\mathrm{T}}^{\mathrm{j}} > 40$ GeV)	7837	20.06	98.32
$H_{\mathrm{T}}^{\mathrm{miss}} \geq 130$ GeV ($p_{\mathrm{T}}^{\mathrm{j}} > 40$ GeV)	4818	12.33	61.48
$H_{\mathrm{T}}^{\mathrm{miss}}/E_{\mathrm{T}}^{\mathrm{miss}} < 1.25$	4050	10.36	84.06
α_{T} H_{T}-dependent cuts	2863	7.33	70.69
$\Delta\phi_{\mathrm{min}}^{*} > 0.5$	859	2.20	30.00

Table 18.2: The preliminary version of the cut flow table for the example 2016 dataset SMS_T1tttt_madgraphMLM (30799443 events, reduced to 39074 *unweighted* events when selections on m_{SUSY} and m_{LSP} were applied). The cuts included comprise all of the relevant selections that were implemented for the benchmark models in the 2015 analysis. The *final* tables will only contain the inclusive/cumulative efficiencies, and the column headers will be replaced by the sample parameters (equivalent to the cut flow table in SUS-14-006_aux).

The jet CHF variable is the charged hadron fraction of the jet. We've added support for including multiple samples. So for a particular model, we can include all the samples (i.e., the different combinations of m_{SUSY} and m_{LSP}) and create cut flow tables for each one in a single run of the code. This was achieved by relocating the standard cut flow to new function (one I called `std_cutflow`) and then adding the sample selection criteria to `path_cfg`. So the relevant code now looks like this:

```
std_cutflow = dict(All = (
    'ev : ev.nJet40[0] >= 2',
    'ev : ev.nJet40Fwd[0] == 0',
    'ev : ev.nJet40failedId[0] == 0',
    'ev : ev.jet_chHEF[0] >= 0.1',
    'ev : -2.5 < ev.jet_eta[0] < 2.5',
    'cutflow_Signal',
```

```
        'isoTrackVeto',
        'nJet100',
        'ht40',
        'mht',
        'ev : ev.MhtOverMet[0] < 1.25',
        dict(Any = (dict(All = ('htbin_200', ('alphaT', dict
(v = 0.65)))),
                    dict(All = ('htbin_250', ('alphaT', dict
(v = 0.60)))),
                    dict(All = ('htbin_300', ('alphaT', dict
(v = 0.55)))),
                    dict(All = ('htbin_350', ('alphaT', dict
(v = 0.53)))),
                    dict(All = ('htbin_400', ('alphaT', dict
(v = 0.52)))),
                    dict(All = ('htbin_600', ('alphaT', dict
(v = 0.52)))),
                    dict(All = ('htbin_800', ))
                    )
            ), # AlphaT cut
        'biasedDPhi',
    ))

    path_cfg = dict(Any = (
        dict(All = ('ev : ev.smsmass1[0] == 1300', 'ev : ev.
smsmass2[0] == 1050', std_cutflow)),
        dict(All = ('ev : ev.smsmass1[0] == 1800', 'ev : ev.
smsmass2[0] == 500', std_cutflow)),
        # Can add more samples here in the same vein as
above
    ))
```

Listing 18.3: The relevant functions that need to be edited to produce cut flow tables. The first function specifies the standard cut flow for a sample, and the second specifies the selection criteria for the samples themselves. File name: twirl_mktbl.py (VI, lines 86–114).

I'll probably use Tai's method, as opposed to Dom's, to generate the cut flow tables for the

benchmark models. With Tai's method, it could be possible to run over the MiniAOD files rather than the flat trees (which are created from MiniAODs), which might be better because tree production already applies some loose cuts which could then skew the results of the cut flow.

I wrote a short talk giving an update on the cut flows using Tai's method. The link to the slides is After presenting, the general consensus on how to proceed is to create flat trees from MiniAOD files in Heppy, applying no cuts. Then incorporate the code from Tai's cutflowirl repo into AlphaTools and rerun the procedure to generate cut flows, this time including every selection (as detailed in Sec. 18.4.3) regardless of whether they apply to these SMS models or not. Then we can discuss the order and grouping of the cuts and make nice, polished tables.

18.7 Creating the cut flow tables for the benchmark models

18.7.1 Making the flat trees (with no cuts) for the models

Using the information detailed in the previous subsections, I should be able to start generating the cut flow tables for the SMS models in the 2015 analysis. The first task is to produce the flat trees for the models in Heppy without any cuts (except the mass points from the different samples). To remain consistent between the production of these trees and the original ones, I'll need to use the same version of CMSSW and the same tag of the code in Heppy/CMGTools. They used CMSSW_7_4_X at the time so I'm using 7_4_14. Stefano gave me the tag for the SMS models (74X_MC_20151207_D01). So I can follow the instructions at https://github.com/CMSRA1/RA1OPS/wiki/B1.-Flat-Tree-Production-(74X) and just check out that tag. Although the original trees would have been produced at different times, they should have been so with consistent settings (otherwise the results in the paper would be wrong), so using the tag above should be fine. Once I was on that tag I checked out my own branch *esh-MCtreeprod20170221*. I will now need to add a skimmer for the sparticle masses. I can take inspiration from https://github.com/CMSRA1/cmgtools-lite-private/blob/80X-ra1-0.7.x/TTHAnalysis/python/analyzers/susyParameterScanAnalyzer.py to get the variables corresponding to the masses and then add some `if` statements to match the sample(s).

However, after some trial and error, and nosing around the files in Heppy and CMGTools, I realised I didn't even need to include mass cuts in the sequence for tree production. The configuration file I would be using to create the flat trees would be **run_susyAlphaT_AtLogic_MC_SUSY_SMS_25ns_cfg.py** in the directory **$CMSSW_BASE/src/CMGTools/TTHAnalysis/cfg/**. At L12, the variable `componentArgsList` lists the components (i.e., the models) that are run over during tree

production. These components, along with the samples that make up the components (i.e., the miniAOD files), are defined in **TTHAnalysis/python/components/components_alphaT_13TeV_MC_signal_74X.py**. By default, these lists contain *all* of the samples for each model. So I can simply comment out that list and replace it with my own that includes the sample(s) which contain the mass points I'm interested in. So, for example, I want to look at the $m_{\mathrm{SUSY}} = 1300, m_{\mathrm{LSP}} = 100$ mass points for the T1tttt model. I can look through all the samples defined in the file, and then write my own list:

```
componentArgsList_T1tttt_25ns = [
   SMS_T1tttt_mGluino_1300_mLSP_1to1075_25ns ,]
```

Then when I run the flat tree production, only that sample would be included, meaning that tree production is *much* faster and I don't have to apply mass cuts in the sequence (because that gave me problems before). In the file, samples for only three of the benchmark models were included. The others weren't pushed to the tag I based my analysis on. So Stefano uploaded the component files containing all of these other samples. I called them **components_alphaT_13TeV_MC_signal_74X_extra.py** and **components_alphaT_13TeV_MC_signal_74X_extra_MiniAODv2.py**. Now I had available all of the samples I would need to produce the trees needed for the cut flows.

The next step was removing the other loose cuts applied when making the flat trees. After trudging through the event selection modules, I just commented out the selections in the dictionary `buildEventSelection_options` in **run_susyAlphaT_AtLogic_MC_SUSY_SMS_25ns_cfg.py**. Throughout these phases I was recompiling the code and then testing each modification to see if it worked.

Once these changes had been implemented, I was almost ready to submit jobs for tree production. Whilst attempting this I ran in to errors, namely one that caused the batch script associated with each job to remain empty. This was due to the fact that the tag I had based my edits on was out of date. Since that tag had been created, the computing architecture at Bristol had changed. We are using a new hostname (soolin.dice.priv instead of soolin.phy.bris.ac.uk), so that had to be updated in **$CMSSW_BASE/src/PhysicsTools/HeppyCore/python/utils/batchmanager.py** L302, as well as in any other file that still contained the old hostname. I could just navigate to **$CMSSW_BASE/** and type `grep -ir soolin.phy.bris.ac.uk`, then update the files containing that string.

I could now finally submit jobs to Condor – preferably from **$CMSSW_BASE/src/**, because everything in the submission directory is re-tarred and sent to the worker nodes that run the jobs – with the command

```
heppyBatchAlphaT.py -o $OUTDIR -c AtLogic_MC_SUSY_SMS_25ns
```

where **$OUTDIR** = **/storage/eb16003/Cutflow2015Storage/74X/MC/20170221_S01**. If any

jobs fail and then need to be resubmitted, errors can be encountered at this step. If the error references the file **$CMSSW_BASE/bin/slc6_amd64_gcc491/cmgResubChunk**, check to see if the hostname is correct. That might solve the problem. Another problem I faced was when specific jobs quit as soon as they started running. This was due to the file **pycfg.py** being empty in the failed chunks. This file is the same for each chunk in a specific sample, so if faced with that error, I can just copy the file from a successful chunk in that sample into the directory of the failed one.

Once the jobs have finished, I can extract and check the output files, and resubmit any failed jobs, using the commands in the flat tree production tutorial. Once *all* of the jobs have succeeded, I can combine the output files so that I get a single flat tree for each sample (stored in **<sample>/treeProducerSusyAlphaT/**). I should double-check that everything is in order by opening the root file and looking at the histograms. A quick cross check would be to compare the number of events in the tree to that of the corresponding miniAOD file (which I can do using Tai's cut flow code); these should be the same (which they were when I tested it on a T1tttt sample), indicating that everything wrote to the root file properly and that no loose cuts were applied during tree production. Another would be to take a look at the histograms of the sparticle masses. These should effectively be delta functions where the different masses are, and also gives an insight into how many events you should expect when running over specific mass points. The final task is to copy all the trees to **/hdfs** because I shouldn't have $\sim$ 600 GB of stuff clogging up **/storage**. The locations are as follows:

– /hdfs/SUSY/RA1/74X/MC/20170302_S01

– /hdfs/SUSY/RA1/74X/MC/20170303_S02

– /hdfs/SUSY/RA1/74X/MC/20170304_S02

– /hdfs/SUSY/RA1/74X/MC/20170306_S01

18.7.2 Making the cut flow tables themselves

Now Tai's cut flow code – see Sec. 18.6 – comes into play. As some of the samples used in tree production contain mass *ranges*, I can make mass *point* cuts on m_{SUSY} and m_{LSP} here. At the moment, the repo containing the code is in **/storage/eb16003/TEMPCUTFLOW/** on Soolin. In this directory, I need to edit the files **cutflowirl/twirl_mktbl.py** to point to the directory containing my flat trees, and **cutflowirl/FrameworkHeppy.py**, changing all mentions of `'roctree'` to `'treeProducerSusyAlphaT'`. Then I can run the command

```
./cutflowirl/twirl_mktbl.py --components <sample(s)> --
   logging-level INFO --parallel-mode multiprocessing
```

to get the raw cut flow table. The argument `<sample>` should be the name of the sample that contains the tree we're interested in, e.g., `SMS-T1tttt_mGluino-1300_mLSP-1to1075_25ns`. The output folder **./tbl/out/** contains several files, including the cut flow table and the actual event selections as programmed. There's also a file which details the number of events in the miniAOD for the sample, so the cross-checking between it and the tree can be done.

If there are more selections that are needed for the cut flow table, or I want the syntax for a particular selection, I can just look at the leaves of the tree (either with TBrowser or by creating the class files – see Sec. 6). Then the syntax I would include in **twirl_mktbl.py** would be `"ev : ev.<leaf> <cut>"`. Most importantly, for the mass points, the syntaxes would be

```
"ev : ev.GenSusyMGluino[0] == 1300",
"ev : ev.GenSusyMNeutralino[0] == 100",
```

for the T1tttt example I used above.

If I look at Table 5 of the paper, showcasing the benchmark models and their samples, there's a column called "Most sensitive n_{jet} categories". These show the four most sensitive categories. Some of them contain an "a" following the number, which stands for "asymmetric". Asymmetric jets refer to the p_{T} of the two hardest jets. In the cut flow, it is required that at least two jets have $p_{\text{T}}^{\text{j}} > 40$ GeV, and the leading jet (j_1) has $p_{\text{T}}^{\text{j}_1} > 100$ GeV. If the second-hardest jet (j_2) also has $p_{\text{T}}^{\text{j}_2} > 100$ GeV, the topology is "symmetric". But, if it instead is $40 < p_{\text{T}}^{\text{j}_2} < 100$ GeV, the topology is asymmetric. In the rare case of $p_{\text{T}}^{\text{j}_2} < 40$ GeV, we have the "monojet" regime. I need to account for these n_{jet} categories in the cut flow. Essentially, I have a standard cut flow that is executed for each sample. Then at the end of that cut flow, I specify the sample-specific selections (i.e., the n_{jet} categories).

When making the cut flow tables for a sample I need to make sure that (in **twirl_mktbl.py**) I've specified the correct path in `default_heppydir`, the sparticles and their masses are correct, the n_{jet} categories are correct, and that I'm including the correct `--components` argument when executing the command in the terminal.

After I made a few cut flow tables, I compared the end-of-cut-flow efficiencies with those listed in Table 5 of the paper. In the comparisons for all of the models I had tested, there was a discrepancy between the efficiencies. My values were consistently $\sim 2.0 - 2.5$x those in the paper. My initial reaction was that I had made a mistake somewhere. But on closer inspection, Matt and Rob discovered that the values in the paper were actually incorrect because they hadn't been scaled by the luminosity (2.3 fb^{-1}). So I inadvertently found a mistake in the paper (which I'm proud of). And it's good that this was found now rather than after publication. After generating the rest of the tables, I found another mistake. My efficiencies for the T2bb samples were a factor of more than 20 greater than in the paper. This was due to the values in the paper using the squark cross section as opposed to the sbottom. When corrected, the efficiencies were more consistent. For my contributions to the cut

flows, I've also been added to the internal authors list. Because I'm not a CMS author and the public author list for the paper was finalised ages ago, I'm not an author on the *public* paper but I am on the *internal* paper (i.e., the draft versions) and the accompanying analysis note CMS AN-15-004. This is a huge step because it shows that I've made a significant enough contribution to the analysis that the rest of the group recognise it, and I know how much work and effort is needed to earn authorship.

18.7.3 The cut flow

I've sent Rob Bainbridge the list of skimmers used in the 2015 analysis, the leaves in the flat trees and the event selection dictionary from Tai so he could decide what cuts we should use and in what order. After some deliberation, the cut flow for these benchmark models is as follows:

```
    std_cutflow = dict(All = (
        dict(All = ("ev : ev.nElectronsVeto[0] == 0",
                    "ev : ev.nMuonsVeto[0] == 0",
                    )), # Lepton vetoes
        "ev : ev.nIsoTracksVeto[0] <= 0",
        "ev : ev.nPhotonsVeto[0] == 0",
        "ev : ev.nJet40Fwd[0] == 0",
        "ev : ev.nJet40[0] >= 2",
        "ev : ev.jet_pt[0] > 100",
        "ev : -2.5 < ev.jet_eta[0] < 2.5",
        "ev : ev.ht40[0] > 200",
        "ev : ev.mht40_pt[0] > 130",
        "ev : ev.MhtOverMet[0] < 1.25",
        dict(Any = (dict(All = ('htbin_200', ('alphaT', dict
(v = 0.65)))),
                    dict(All = ('htbin_250', ('alphaT', dict
(v = 0.60)))),
                    dict(All = ('htbin_300', ('alphaT', dict
(v = 0.55)))),
                    dict(All = ('htbin_350', ('alphaT', dict
(v = 0.53)))),
                    dict(All = ('htbin_400', ('alphaT', dict
(v = 0.52)))),
```

```
                    dict(All = ('htbin_600', ('alphaT', dict
(v = 0.52)))),
                    dict(All = ('htbin_800',))
                    )
            ), # HT-dependent AlphaT cuts
        "ev : ev.biasedDPhi[0] > 0.5",
        dict(Any = ( dict(All = ("ev : ev.nJet100[0] >= 2",
"ev : ev.nJet40[0] >= 5",) ), #>=5
                     dict(All = ("ev : ev.nJet100[0] >= 2",
"ev : ev.nJet40[0] == 4",) ), #4
                     dict(All = ("ev : ev.nJet100[0] == 1",
"ev : ev.nJet40[0] >= 5",) ), #>=5a
                     dict(All = ("ev : ev.nJet100[0] == 1",
"ev : ev.nJet40[0] == 4",) ), #4a
                    )), # Most sensitive n_jet categories (
sample-specific)
        ))

 path_cfg = dict(Any = (
        dict(All = ('ev : ev.GenSusyMGluino[0] == 750', 'ev
: ev.GenSusyMNeutralino[0] == 600', std_cutflow)),
        # Can add more mass points here in the same vein as
above
 ))
```

Listing 18.4: The final cut flow to generate raw cut flow tables for the benchmark models used in the 2015 analysis. The mass points are specified in the function `path_cfg` and the rest of the selections are in `std_cutflow`. File name: twirl_mktbl.py (v2, L86–119).

18.8 The final cut flow tables

After the months of slogging away, I have finally generated the cut flow tables for SUS-15-005. The tables are linked from the auxiliary material, located here: https://twiki.cern.ch/twiki/pub/CMS/PhysicsResultsSUS15005/SUS-15-005_aux_temp.pdf. They can also be viewed on the public webpage here: http://cms-results.web.cern.ch/cms-results/

public-results/publications/SUS-15-005/index.html. The raw tables were generated by me, which I then passed to Rob for formatting in LaTeX so the final versions were consistent with the style of the paper.

All of the cut flow tables, taken directly from the auxiliary material (https://github.com/CMSRA1/AlphaTDR2/blob/master/papers/SUS-15-005/trunk/SUS-15-005_aux.tex) are here:

Table 18.3: A summary of the cumulative signal acceptance times efficiency, $\mathcal{A}\,\varepsilon$ [%], for various benchmark models with both compressed and uncompressed mass spectra, following the application of the event selection criteria used to define the signal region. Values for $\mathcal{A}\,\varepsilon$ are also shown following the application of additional requirements that define the four most sensitive n_{jet} event categories, as defined in Table 5 of the paper. Scale factor corrections to simulated signal events that account for the mismodelling of theoretical and experimental parameters are not applied, and so the values for $\mathcal{A}\,\varepsilon$ differ with respect to those in Table 5 by up to 15%.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})					
	T1qqqq (1300, 100)	T1qqqq (900, 700)	T2qq_8fold (1050, 100)	T2qq_8fold (650, 550)	T2qq_1fold (600, 50)	T2qq_1fold (400, 250)
Before selection	100	100	100	100	100	100
Event veto for muons and electrons	99	100	100	100	100	100
Event veto for single isolated tracks	94	91	96	95	96	95
Event veto for photons	92	90	95	94	95	95
Event veto for forward jets ($\lvert\eta\rvert > 3.0$)	81	78	82	81	80	80
$n_{\text{jet}} \geq 2$	81	78	81	72	80	75
$p_{\text{T}}^{\text{j}_1} > 100\,\text{GeV}$	81	71	81	57	79	66
$\lvert\eta^{\text{j}_1}\rvert < 2.5$	81	70	81	55	79	65
$H_{\text{T}} > 200\,\text{GeV}$	81	69	81	50	79	60
$H_{\text{T}}^{\text{miss}} > 130\,\text{GeV}$	77	50	78	33	71	40
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	74	44	75	28	65	33
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 800\,\text{GeV}$)	74	30	71	15	50	17
$\Delta\phi_{\text{min}}^{*} > 0.5$	22	18	44	10	33	13
Four most sensitive n_{jet} event categories	22	13	43	5.5	31	6.1

Table 18.4: A summary of the cumulative signal acceptance times efficiency, $\mathcal{A}\varepsilon$ [%], for various benchmark models with both compressed and uncompressed mass spectra, following the application of the event selection criteria used to define the signal region. Values for $\mathcal{A}\varepsilon$ are also shown following the application of additional requirements that define the four most sensitive n_{jet} event categories, as defined in Table 5 of the paper. Scale factor corrections to simulated signal events that account for the mismodelling of theoretical and experimental parameters are not applied, and so the values for $\mathcal{A}\varepsilon$ differ with respect to those in Table 5 by up to 15%.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})					
	T1bbbb (1500, 100)	T1bbbb (1000, 800)	T1tttt (1300, 100)	T1tttt (800, 400)	T1ttbb (1300, 100)	T1ttbb (1000, 700)
Before selection	100	100	100	100	100	100
Event veto for muons and electrons	99	98	41	42	61	64
Event veto for single isolated tracks	94	91	31	32	51	54
Event veto for photons	93	91	30	32	50	54
Event veto for forward jets ($\lvert\eta\rvert > 3.0$)	82	79	27	27	44	47
$n_{\text{jet}} \geq 2$	82	78	27	27	44	47
$p_{\text{T}}^{\text{j}_1} > 100\,\text{GeV}$	82	69	27	25	44	43
$\lvert\eta^{\text{j}_1}\rvert < 2.5$	82	68	27	25	44	42
$H_{\text{T}} > 200\,\text{GeV}$	82	66	27	25	44	42
$H_{\text{T}}^{\text{miss}} > 130\,\text{GeV}$	79	48	25	15	41	32
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	77	43	24	11	38	26
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 800\,\text{GeV}$)	77	29	24	8.3	38	19
$\Delta\phi_{\text{min}}^{*} > 0.5$	23	17	5.6	1.3	9.5	8.8
Four most sensitive n_{jet} event categories	23	12	5.6	1.3	9.5	7.4

Table 18.5: A summary of the cumulative signal acceptance times efficiency, $\mathcal{A}\,\varepsilon$ [%], for various benchmark models with both compressed and uncompressed mass spectra, following the application of the event selection criteria used to define the signal region. Values for $\mathcal{A}\,\varepsilon$ are also shown following the application of additional requirements that define the four most sensitive n_{jet} event categories, as defined in Table 5 of the paper. Scale factor corrections to simulated signal events that account for the mismodelling of theoretical and experimental parameters are not applied, and so the values for $\mathcal{A}\,\varepsilon$ differ with respect to those in Table 5 by up to 15%.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})			
	T5tttt_DM175 (800, 100)	T5tttt_DM175 (700, 400)	T5ttcc (1200, 200)	T5ttcc (750, 600)
Before selection	100	100	100	100
Event veto for muons and electrons	41	42	63	63
Event veto for single isolated tracks	30	32	53	53
Event veto for photons	30	31	53	52
Event veto for forward jets ($\lvert\eta\rvert > 3.0$)	25	27	46	45
$n_{\text{jet}} \geq 2$	25	27	46	41
$p_{\text{T}}^{\text{j}_1} > 100\,\text{GeV}$	25	21	46	25
$\lvert\eta^{\text{j}_1}\rvert < 2.5$	25	21	46	24
$H_{\text{T}} > 200\,\text{GeV}$	25	21	46	23
$H_{\text{T}}^{\text{miss}} > 130\,\text{GeV}$	17	9.4	44	15
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	11	5.6	42	12
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 800\,\text{GeV}$)	11	3.9	41	7.5
$\Delta\phi_{\text{min}}^{*} > 0.5$	0.4	0.5	13	3.2
Four most sensitive n_{jet} event categories	0.4	0.4	13	2.3

Table 18.6: A summary of the cumulative signal acceptance times efficiency, $\mathcal{A}\varepsilon$ [%], for various benchmark models with both compressed and uncompressed mass spectra, following the application of the event selection criteria used to define the signal region. Values for $\mathcal{A}\varepsilon$ are also shown following the application of additional requirements that define the four most sensitive n_{jet} event categories, as defined in Table 5 of the paper. Scale factor corrections to simulated signal events that account for the mismodelling of theoretical and experimental parameters are not applied, and so the values for $\mathcal{A}\varepsilon$ differ with respect to those in Table 5 by up to 15%.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})					
	T2bb (800, 50)	T2bb (375, 300)	T2tb (600, 50)	T2tb (350, 225)	T2tt (700, 50)	T2tt (350, 100)
Before selection	100	100	100	100	100	100
Event veto for muons and electrons	99	99	72	80	63	63
Event veto for single isolated tracks	96	94	61	72	53	53
Event veto for photons	95	94	60	72	52	52
Event veto for forward jets ($\lvert\eta\rvert > 3.0$)	81	81	51	62	45	45
$n_{\text{jet}} \geq 2$	80	61	51	53	45	44
$p_{\text{T}}^{\text{j}_1} > 100$ GeV	80	36	50	36	44	35
$\lvert\eta^{\text{j}_1}\rvert < 2.5$	80	34	50	34	44	34
$H_{\text{T}} > 200$ GeV	80	30	50	30	44	33
$H_{\text{T}}^{\text{miss}} > 130$ GeV	75	18	44	17	40	20
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	72	15	38	12	38	15
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 800$ GeV)	62	7.2	30	5.5	34	8.8
$\Delta\phi_{\text{min}}^{*} > 0.5$	39	4.5	17	3.2	21	4.0
Four most sensitive n_{jet} event categories	37	2.9	14	2.1	19	3.0

Table 18.7: A summary of the cumulative signal acceptance times efficiency, $\mathcal{A}\varepsilon$ [%], for various benchmark models with both compressed and uncompressed mass spectra, following the application of the event selection criteria used to define the signal region. Values for $\mathcal{A}\varepsilon$ are also shown following the application of additional requirements that define the four most sensitive n_{jet} event categories, as defined in Table 5 of the paper. Scale factor corrections to simulated signal events that account for the mismodelling of theoretical and experimental parameters are not applied, and so the values for $\mathcal{A}\varepsilon$ differ with respect to those in Table 5 by up to 15%.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})		
	T2cc (325, 305)	T2tt_degen (300, 290)	T2tt_mixed (300, 250)
Before selection	100	100	100
Event veto for muons and electrons	100	100	89
Event veto for single isolated tracks	97	98	83
Event veto for photons	97	97	83
Event veto for forward jets ($\lvert\eta\rvert > 3.0$)	83	84	72
$n_{\text{jet}} \geq 2$	26	21	36
$p_{\text{T}}^{\text{j}_1} > 100\,\text{GeV}$	16	14	19
$\lvert\eta^{\text{j}_1}\rvert < 2.5$	15	13	18
$H_{\text{T}} > 200\,\text{GeV}$	13	11	15
$H_{\text{T}}^{\text{miss}} > 130\,\text{GeV}$	11	9.2	10
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	9.2	7.5	8.4
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 800\,\text{GeV}$)	4.8	4.3	3.7
$\Delta\phi_{\text{min}}^{*} > 0.5$	3.7	3.7	2.3
Four most sensitive n_{jet} event categories	1.9	1.9	0.9

The paper was finally published in the European Physical Journal in May 2017 [61].

Chapter 19

Imperial College account details (01/02/2017)

My details for my Imperial guest account are as follows:

– Email address: e.bhal@imperial.ac.uk

– College Identifier (CID): 942250

– Hostnames: lxoo.hep.ph.ic.ac.uk | lxo1.hep.ph.ic.ac.uk

– Aliases for hosts: imperialoo | imperialo1

– Username: ebhal

– Password: <normal password>

I can just ssh into the remote server like I do with Soolin – `ssh ebhal@lx0<#>.hep.ph.ic.ac.uk` or `ssh imperial0<#>`. These servers are Linux based and run CentOS6. But I can ssh into the Imperial servers from anywhere in the world on any network (check this), so I don't have to be on a specific network like I do for Soolin. More information can be found at https://www.hep.ph.ic.ac.uk/private/computing/.

19.1 lxoo and lxo1 remote server details

I can ssh into both lxoo and lxo1, they both take me to the same place. So I don't need to maintain two setups at IC. They have a grid like Bristol does. However, their batch submission and monitoring software is slightly different. They use Grid Engine (SGE) rather than HTCondor. A rundown

of the batch system is located at https://www.hep.ph.ic.ac.uk/private/computing/sge.shtml. The main takeaways are that the command for job submission is

```
qsub -q hep.q <submission script>
```

I can also specify a maximum time for a job by adding the option `-l h_rt=<hours>:<minutes>:<seconds>`. The maximum, and also default, is 48 hours. Typing `man qsub` and `man sge_intro` displays the manual and other commands, respectively. I can check on jobs by typing

```
qstat
```

I can delete jobs by typing

```
qdel <job ID>
```

where `job ID` is the number in the first column of output when typing `qstat`. Or I can type

```
qdel -u ebhal
```

to kill all my jobs. I can also get more information about a job with

```
qstat -j <job ID>
```

Chapter 20

Service work: Jet Energy Corrections in the Level-1 Trigger (02/02/2017)

Service work (or EPR – Experimental Physics Responsibility) entails some contribution to the experiment a student is working on, over a period of six months, in order to get onto the authors list for papers published by the collaboration. In CMS (at Bristol), I'll be working on the Level-1 Trigger, which is our main contribution to the experiment. The aspect of the trigger I'll be working on is Jet Energy Corrections (JECs) in the Hadron Calorimeter (HCAL, Layer-2). In essence, we (myself and Joe Taylor) are trying to correct the Level-1 jets for various losses, which affect the turnoff curves, rate and efficiency of the trigger. The losses are both p_T- and η-dependent. Because we want the trigger performance to be uniform across the detector, we need some "ideal" or "reference" jet to compare to a given Level 1 jet. There are programs which can produce these reference jets, and we can use them to correct for any given Level-1 jet we detect. This can also include studying the response of the trigger, the resolution in position and energy, and possibly other quantities. These corrections are implemented on a jet-by-jet basis, which means there's no automated method for doing this and why people need to be continually working on it.

The JECs are at the end of a long chain of studies and corrections: ECAL/HCAL Trigger primitives (TPs) → Regions (RCT/Stage 1) / Towers (Layer 1) → Jet finding (GCT/Layer 2) → L1JEC → Global Trigger (GT).

By visiting https://icms-epr.cern.ch/ I can "pledge" a certain amount of time for a specific task as part of my EPR. Because the "L1 Trigger / prompt analysis / CaloL2 Jets/Sums" category is usually oversubscribed, I can pledge for the "L1 Trigger / software / calo trigger offline software devel.

& maint. : Layer-2 Jets+Sums" category.

I've been sent documents giving an explainer on JECs. The older instructions, written by Robin Aggleton, provide more of a physics-based overview as to what's going on: L1JEC explainer – Robin. The newer instructions, written by Joe Taylor, have more relevance to the code itself and how to run everything: Level-1 Trigger Jet Energy Corrections – Joe, 01/02/2017.

From looking at Joe's explainer, I've started setting up the necessary software on Soolin. The commands that are repeatable, i.e., the ones I need to enter each time I want to use the software, is in a shell script **cmssw_crab_JEC.sh** in my home directory on Soolin. Then I just need to `source` it to initialise the software. Joe's slides tell me to create my own offline remote for the Level-1 Trigger repository and push any changes there.

```
source /cvmfs/cms.cern.ch/cmsset_default.sh

cd /users/eb16003/JEC_Software
# The CMSSW version I need will probably change regularly,
   so make sure I'm initialising the correct version
# The same applies to the tag in the repo (current
   integration tag is v91.16)
cd CMSSW_9_0_0_pre2/src
cmsenv

# Source the CRAB environment
source /cvmfs/cms.cern.ch/crab3/crab.sh
# Start a proxy of a grid certificate
voms-proxy-init --voms cms --valid 168:00

# For sending jobs to DICE
cd ~/htcondenser_legacy
source setup.sh

cd /users/eb16003/JEC_Software/CMSSW_9_0_0_pre2/src/
   L1Trigger
```

Listing 20.1: My bash script for sourcing the relevant files and initialising the environment for the Level-1 Trigger code and CRAB. File name: cmssw_crab_JEC.sh (v2).

There's software called CRAB (CMS Remote Analysis Builder) which submits jobs

to their server and grid. This requires my grid certificate, and the grid pass is the same as I used for the RA1 tree production. More information on CRAB can be found at https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideCrab. When running jobs with CRAB, I can check on the progress either by typing `crab status` or clicking the link http://dashb-cms-job.cern.ch/dashboard/templates/task-analysis/#user=default&refresh=0&table=Mains&p=1&records=25&activemenu=2&pattern=&task=&from=&till=&timerange=lastWeek.

To create a remote, first fork the corresponding repository on GitHub. Then on the command line, clone the repository in some working directory using `git clone <remote URL> <destination path>`. I can then add my own remote by typing `git remote add <remote name> git@github.com:eshwen/<same repo>.git`. So I can make changes to files (as long as I've switched to my branch), then add and commit them. When I want to push them, type `git push <remote name> <branch I'm working on>`. Other people can fetch my remote by adding it in the same way I did, then typing `git fetch <remote name>`.

So my remote (in the style <remote name> <remote URL>) is

eshwen git@github.com:eshwen/L1JetEnergyCorrections.git (fetch)

eshwen git@github.com:eshwen/L1JetEnergyCorrections.git (push)

Then I can edit files, and add and commit them. I can work on the "master" branch because it's my remote. I can push my changes by typing

```
git push eshwen master
```

If I make significant changes, let Joe know so he can make a pull request and merge them with his repository. And when editing any files, make sure to recompile just to be safe.

20.1 First round of JECs (end date – 17/03/2017)

My first task was to conduct closure tests for the next set of Layer 1 corrections. This required the dataset **/QCD_Pt-15to3000_TuneCUETP8M1_Flat_13TeV_pythia8/RunIISpring16DR80-FlatPU20to70HcalNZSRAW_withHLT_80X_mcRun2_asymptotic_v14-v1/GEN-SIM-RAW**, with the software **CMSSW_9_0_0_pre2** and the integration tag **l1t-integration-v91.16**. I could follow Joe's explainer from the start, just replacing the old CMSSW version and integration tag with the new one. All the relevant code was in **$CMSSW_BASE/src/**. I had to edit **L1Trigger/L1TCalorimeter/python/caloStage2Params_2017_v1_1_cfi.py**, making sure

L72: `caloStage2Params.jetCalibrationType = cms.string("None")` was *commented* out, and

L74: `caloStage2Params.jetCalibrationType = cms.string("functionErf11PtParams16EtaBins'` was *uncommented*

That *turns on* the JECs. Then I had to edit L45 of **./hackConditions_cff.py** to point to the Params file. The final edit was to update the variable `job_append` in **L1Trigger/L1JetEnergyCorrections/crab/crab3_stage2.py**. I just needed to replace the date, CMSSW version, integration tag, and git commit hash with their current iterations (as well as replacing "noJec" with "withJEC"). I can get the commit hash by typing `git log`. The first entry will be the most recent update, and I only need the first 6 or 7 characters from that commit hash to include in `job_append`. So the variable now looked like

```
job_append = "
    crab_qcdSpring16FlatPU20to70genSimRaw_qcdSpring16_genEmu_21Feb2017_902
    "
```

After testing locally, I could submit the jobs to the grid by typing

```
python crab3_stage2.py
```

which would create the ntuples *with corrections*. Once finished, the next step was to copy them from **/hdfs/dpm/phy.bris.ac.uk/home/cms/store/user/ebhal/<dataset>_<ID>/oooo/** to **/hdfs/L1JEC/CMSSW_9_0_0_pre2/<string from** `job_append`**>/initialNtuples/**. Because it is a hadoop distributed file system, the terminal commands are slightly different. A cheat sheet is here: https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/FileSystemShell.html. In full, the command is

```
hadoop fs -cp <absolute path to input minus the "/hdfs"> <
    absolute path to output minus the "/hdfs">
```

Once the ntuples had been copied over, the next stage in the work flow was matching these corrected L1 jets with the MC genJets. This was done by navigating to **L1JetEnergyCorrections/bin/HTCondor/** and editing the `NTUPLE_DIRS` variable in **submit_matcher_dag.py** to point to the ntuples I just created. Then I just had to run

```
./submit_matcher_dag.py
```

which submits a "mother" or "controller" job to the grid, which further spawns a sub-job for each ntuple I'm running over. Using DAG is different to normal batch submission because these jobs need to be executed in a specific sequence, which isn't possible with regular batch submission. When the jobs have been submitted, a line will print to the screen of the form

```
All statuses: DAGstatus.py  /storage/eb16003/L1JEC/
   CMSSW_9_0_0_pre2/L1JetEnergyCorrections/jobs/pairs/<date
   >/matcher_102253_pCB.status
```

So I can check the status of the jobs by typing

```
python DAGstatus.py  /storage/eb16003/L1JEC/CMSSW_9_0_0_pre2
   /L1JetEnergyCorrections/jobs/pairs/<date>/
   matcher_102253_pCB.status
```

If some jobs fail, I can resubmit them via

```
condor_submit_dag /storage/eb16003/L1JEC/CMSSW_9_0_0_pre2/
   L1JetEnergyCorrections/jobs/pairs/<date>/
   matcher_102253_pCB.dag
```

(i.e., running `condor_submit_dag` and pointing to the .dag file) and repeat until all the jobs have succeeded. This is just like with flat tree production: I could resubmit failed jobs, check the output, and then repeat.

When submitting matching jobs, they all seemed to fail at every attempt. After a long period of being stagnant, we found the issue. The problem was due to the htcondenser library. It was a newer version that seemed to be incompatible with the L1JEC code. So I cloned an older version using

```
git clone git@github.com:joseph-taylor/htcondenser_legacy.
   git ~/htcondenser_legacy
```

and modified my script (Listing 20.1) to reflect the change. I also moved all the JEC software from **/storage/** to **/users/**. Note that, regardless of where the software is located, output will always be written to **/storage/**. Resubmitting the jobs after these changes fixed whatever the problem was.

Once all the jobs were successful a "pairs" ROOT file was created, and stored one directory above the ntuples (and in /pairs/). Now the calibrations that were applied needed to be checked. These jobs are submitted via **submit_checkCalib_dag.py**. I just needed to point the variable `PAIRS_FILES` to this newly created "pairs" ROOT file. I could check on them using DAGstatus the same way as with the matching, and resubmit failed jobs in the same way. The only differences are that these dag and status files are stored in **/storage/eb16003/L1JEC/CMSSW_9_0_0_pre2/L1JetEnergyCorrections/jobs/check/**, and there are three sets of dag and status files (one for each pileup bin, as defined in **L1JetEnergyCorrections/bin/binning.py**).

Once all these jobs had succeeded, the final task was to produce nice looking plots to present to the TSG (Trigger Studies Group). If I went to **L1JetEnergyCorrections/bin/**, I could run

```
python showoffPlots.py --checkcal <path to checked \ROOT
   file> --oDir <output directory>
```

where the folder containing all the "checked" ROOT files would be one directory above the ntuples (and in /check/), and the output directory for this set of plots is **/storage/eb16003/JEC_Plots/<date>/PU<pileup bin>**. Note that I needed to run this script on specific ROOT files in that folder. Most of the files are called something like **check_intialNtuples_ak4_ref10to5000_l10to5000_drop25_<index of eta bin>_etaBinsSel16_PU<pileup bin>_maxPt1022.root**. There are also files with the labels "central" and "forward" in their names, referring to different eta ranges (central = $|\eta| < 3$, forward = $|\eta| > 3$). But we wanted all of them, so we were looking for the ROOT files with no index or central/forward label. And because we had three pileup bins, I only needed to run the script for the three matching files. The only problem I ran into was related to the eta ranges. In the script **showoffPlots.py**, the variables `eta_min` and `eta_max` were set to 0,3 for the central range and 3,5 for the forward range. But in the histograms that are extracted, the ranges are actually 0,2.964 and 2.964,5.191. So I just needed to change the variables at L1040, and also the folder names at L332–334. Then running the command to make plots should give me ∼600 files for each pileup bin. The plots (17-03-2017) are stored in **../Service Work - Jet Energy Corrections in the Level-1 Trigger/** relative to my lab book folder. The relevant identification is

- CMSSW version: 9.0.0.pre2
- Integration tag: `l1t-integration-v91.16`
- Dataset: /QCD_Pt-15to3000_TuneCUETP8M1_Flat_13TeV_pythia8/RunIISpring16DR80-FlatPU20to70HcalNZSRAW_withHLT_80X_mcRun2_asymptotic_v14-v1/GEN-SIM-RAW

20.2 First round of JECs with proper corrections (end date – 02/04/2017)

Unfortunately, the layer 1 people botched the calibrations on their side of things so I have to redo the calibrations from the start (as of 22/03/2017). I'll still be using the same version of CMSSW, but I'll be using the new integration tag **l1t-integration-v92.7**, and the Params file **caloStage2Params_2017_v1_4_inconsistent_cfi.py**.

The procedure was similar to before: I created the initial ntuples with no corrections and ran the matcher, Joe derived the calibrations, then I had to do the closure tests. However, this time the corrections were not stored in a functional form, but in a LUT (lookup table) and then sent to Aaron

Bundock. So I had to get the corrections from Aaron and make sure that the correct LUT was pointed to in the code. To get these, I had to navigate to **$CMSSW_BASE/src/** and type

```
git remote add bundocka git@github.com:bundocka/cmssw.git
git fetch bundocka
git cherry-pick 34c62be
```

and make sure that L76 in **L1Trigger/L1TCalorimeter/python/caloStage2Params_2017_v1_4_inconsistent_cfi.py** – `caloStage2Params.jetCalibrationType = cms.string("LUT")` – was uncommented, and L74 was commented out. Then I followed the procedure in the previous subsection to conduct the rest of the closure tests: creating the ntuples with corrections, running the matcher, checking the calibrations, and making the plots. Like before, the plots are stored in **../Service Work - Jet Energy Corrections in the Level-1 Trigger/** relative to my lab book directory, with the (02-04/2017) tag. The identification for this round of JECs is

– CMSSW version: 9.0.0.pre2

– Integration tag: `l1t-integration-v92.7`

– Dataset: /QCD_Pt-15to3000_TuneCUETP8M1_Flat_13TeV_pythia8/RunIISpring16DR80-FlatPU20to70HcalNZSRAW_withHLT_80X_mcRun2_asymptotic_v14-v1/GEN-SIM-RAW

20.3 Useful commands for hdfs

The **/hdfs** mount is where all the JEC ntuples and data are stored. If I botch something when making them or need to delete ntuples for whatever reason (e.g., a round of JECs that become irrelevant due to the Layer-1 screw ups, or deleting the grid output after I've transferred it elsewhere to free up space on **/hdfs**), I can use

```
hadoop fs -rm -r -skipTrash <absolute path, minus the "/hdfs
    ">
```

if it's in **/hdfs/L1JEC//**. Or, if it's on **/hdfs/cms/store/phy.bris.ac.uk/home/cms/store/user/ebhal/**, i.e., where the grid output is stored, I need to use the grid command

```
gfal-rm -r gsiftp://lcgse01.phy.bris.ac.uk/<absolute path,
    minus the "/hdfs">
```

It's best to do this in a clean shell (but with a grid certificate proxy initialised) as CMSSW environments can cause weird errors when running the command.

20.4 Second round of JECs (end date – 20-06-2017)

This round of JECs is important as the LHC restart is imminent (we have stable beams but not at a luminosity high enough that the detectors need to be fully operational quite yet), and we need the trigger calibrations and corrections completed swiftly. The identification for this round is

- CMSSW version: 9.2.0
- Integration tag: `l1t-integration-v95.13`
- Dataset: /QCD_Pt-15to3000_TuneCUETP8M1_Flat_13TeV_pythia8/PhaseISpring17DR-FlatPU0to70NZS_90X_upgrade2017_realistic_v20-v1/GEN-SIM-RAW

20.4.1 Setting up the environment, making the config and running the ntuples without corrections

I needed to re-clone CMSSW and add the integration tag for the new versions with

```
cd JEC_Software/
ls
cmsrel CMSSW_9_2_0
ls
cd CMSSW_9_2_0/
ls
cd src/
cmsenv
git cms-init
git remote add cms-l1t-offline git@github.com:cms-l1t-offline/cmssw.git
git fetch cms-l1t-offline
git cms-merge-topic -u cms-l1t-offline:l1t-integration-v95.13
git cms-addpkg L1Trigger/L1TCommon
git cms-addpkg L1Trigger/L1TMuon
git clone https://github.com/cms-l1t-offline/L1Trigger-L1TMuon.git L1Trigger/L1TMuon/data
scram b -j 8
git clone git@github.com:eshwen/L1JetEnergyCorrections.git L1Trigger/L1JetEnergyCorrections
```

```
scram b -j 8
```

The params file I had to edit was **caloStage2Params_2017_v1_9_inconsistent_cfi.py** in **L1Trigger/L1TCalorimeter/python/**, setting `jetCalibrationType` to `cms.string("None")`. I then added it to the import in **hackConditions_cff.py**. I also added the new dataset/MC (as labelled above) to **L1JetEnergyCorrections/python/mc_samples.py** in the same format as the previous datasets. If I query the dataset on DAS, I can check the number of events and the root files. The next task was to create the CMSSW config file that makes the ntuples derived from the MC. The one I've been using up to the point (i.e., for the previous MC) is **L1JetEnergyCorrections/python/l1NtupleMcMaker2017_RAW2DIGI.py**. But I can make a new one specifically for the new MC with the command

```
cmsDriver.py -s RAW2DIGI --python_filename=
   l1NtupleMcMaker2017_RAW2DIGI_reEmu_HCAL_TPs.py -n 100 --
   no_output --no_exec --era=Run2_2017 --mc --conditions=92
   X_upgrade2017_TSG_For90XSamples_V1 --customise=L1Trigger/
   Configuration/customiseReEmul.
   L1TReEmulMCFrom90xRAWSimHcalTP --customise=L1Trigger/
   L1TNtuples/customiseL1Ntuple.L1NtupleRAWEMUGEN_MC --
   customise=L1Trigger/Configuration/customiseSettings.
   L1TSettingsToCaloStage2Params_2017_v1_9_inconsistent --
   filein=/store/mc/PhaseISpring17DR/QCD_Pt-15
   to3000_TuneCUETP8M1_Flat_13TeV_pythia8/GEN-SIM-RAW/
   FlatPU0to70NZS_90X_upgrade2017_realistic_v20-v1
   /120000/003FF53C-8232-E711-9340-7CD30ACE160C.root
```

and adding this to the line after `process.GlobalTag = <blah>` in this new config:

```
process.GlobalTag.toGet = cms.VPSet(
    cms.PSet(record = cms.string("HcalLUTCorrsRcd"),
        tag = cms.string("HcalLUTCorrs_2017plan1_v2.0_mc"),
        connect = cms.string("frontier://FrontierProd/
   CMS_CONDITIONS")
    )
)
```

After testing it, I had to edit the **crab/crab3_stage2.py** file to include the updated CMSSW config I just created, the new dataset, and the `job_append` string. Then I submitted the jobs to make

the ntuples without corrections. During this stage, a few jobs failed. If this happens, I can resubmit the failed jobs with

```
crab resubmit -d <path to CRAB project, normally in
    L1JetEnergyCorrections/crab/ directory>
```

Once finished, I copied the output to **L1JEC/CMSSW_9_2_0/**. I also had to remember to make the directories to store everything as they're not created on the fly if they don't currently exist. And I only had to copy the root files from the CRAB output, none of the logs or anything.

20.4.2 Matching, and deriving calibrations

After this was the jet matching. I added the path to the ntuples in the `NTUPLE_DIRS` list in **L1JetEnergyCorrections/bin/HTCondor/submit_matcher_dag.py** and then ran that script. Once completed, I could derive the calibrations. This involved adding the path to the matching output to the list `PAIRS_FILES` in **submit_runCalibration_dag.py** and changing the pileup bins in **../binning.py** to `pu_bins = [[40, 50]]`. I could then run the calibration script.

After that had finished, I had root files in the directory where all the ntuples are stored (up one directory from that, in **output/**). I needed to copy the file that contained the information from all eta bins (i.e., didn't have an eta index in the file name) to my **/users/** directory. Then, I had to run **runCalibration.py** from **L1JetEnergyCorrections/bin/** with the command

```
python runCalibration.py <path to input root file, copied
    over from /hdfs> <path and name of new root file> --redo-
    correction-fit --inherit-params --stage2
```

which fits the correction curves. Now, with this new file I had to "massage" the correction curves so they are more accurate. To do this, I can use Joe's **tuneFits.h** ROOT class. If I open a ROOT session in the directory that contains the new root file with correction curves and type

```
.L $CMSSW_BASE/src/L1Trigger/L1JetEnergyCorrections/bin/
    local_L1JEC_scripts/tuneFits.h+
```

this loads the class. As there are 16 eta bins, their indices go from 0 to 15 (lowest number bin has the index 0). Then I need to type

```
tuneFits t<index> = tuneFits("<root file>", <index>)
```

This opens the graph of the points and the correction curve overlaid in a TBrowser. The terminal should print something like "Fitting between the ranges: <x_min> and <x_max>" which give the lower and upper limits of where the correction curve applies. Typing

`t<index>.redoFit(<delta x_min>, <delta x_max>)`

means I can adjust the bounds of the correction curve. I can just edit `<delta x_min>`, which increases `<x_min>` by that value, and edit `<delta x_max>`, which increases `<x_max>` by that value. I essentially want to cover as much of the x-range as possible whilst still maintaining a good fit. Most of the time, I don't need to meticulously mess with `<delta x_max>`, but want to capture the flatness in the tail up to the point where the error bars get large. But for `<delta x_min>`, I want to change it such that the start of the correction curve is as close as possible to the maximum point on the graph. Once the curve looks good, I can save it with

`t<index>.save()`

and move on to the next eta bin. I need to repeat this for all eta bins, then that's the calibrations completed.

Note that if I save one of the edited curves but then want to go back and redo it, I can't. So I would have to start over again. One tip is to tune `<delta x_max>` first, then tune `<delta x_min>`. Doing it the opposite way will offset the slope around `<delta x_min>`. Also, because the canvas size is set automatically, you can lose resolution in a curve if there are some abnormally large error bars. To zoom in, I can drag the cursor between the minimum and maximum value I want to see for a given axis, e.g., if I want my y bounds to be 0 and 10, I drag the cursor from $y = 0$ to $y = 10$. If I do fuck up one or two of the curves, but the rest are fine, they all still need to be re-done. But in a ROOT session, I can use Ctrl-R to reverse search and find the most recent `<delta x_min>` and `<delta x_max>` values for each `t<index>`.

20.4.3 Making the LUTs

Then I needed to make the lookup tables (or LUTs). Joe's previous instructions didn't contain this step, so he sent me an updated version: Level-1 Trigger Jet Energy Corrections – Joe, 26/05/2017. To do this, I had to go to **L1JetEnergyCorrections/bin/local_L1JEC_scripts/** and edit **writeParamsForLUT.cxx**. I needed to change `inputFile` to point to the new massaged root file, and `outputFile` to the output path (for simplicity, keep it in the same folder as the input file, with the same file name, except with a .txt extension). Then I can run the macro/script with

```
root -q -b -l $CMSSW_BASE/src/L1Trigger/
   L1JetEnergyCorrections/bin/local_L1JEC_scripts/
   writeParamsForLUT.cxx
```

Then go to **L1JetEnergyCorrections/bin/**. I need a file called **lut_pt_compress.txt** which can be found in the L1Trigger-L1TCalorimeter repository by the cms-l1t-offline organisation: https://github.com/cms-l1t-offline/L1Trigger-L1TCalorimeter. So copy that file and place

it in the current directory. I also need to use an environment that contains the matplotlib Python library. To do this, I had to type

```
conda create -n matplotlib_eb16003 matplotlib
```

in a new Soolin session (so I wasn't in any environment). Then to initialise it – again, in a clean shell, but with X11 forwarding – I could type

```
source activate matplotlib_eb16003
source /software/root/v5.34.25/bin/thisroot.sh
```

I only have to run the creation once, but I need to initialise it each time I want to use the library. Then to deactivate it, I can either log out or just type

```
source deactivate
```

Once activated, I could go back to **L1JetEnergyCorrections/bin/** and make the LUTs by typing

```
python correction_LUT_plot.py <path to output text file from
    previous step> <output path>/lut_HR.txt --stage2 --plots
    --text --ptCompressionFile lut_pt_compress.txt
```

I should get a few text files, which are the LUTs, and a load of plots in pdf files. As a sanity check, I should look at these plots (luckily the TBrowser in ROOT can view them). The ones we're interested in are named **pt_pre_vs_post_#.pdf**, and the red lines should basically match the blue lines.

The final steps with the LUTs were to convert them into different formats. I need the files **lut_HR_add_mult.txt** and **lut_HR_eta.txt**, which are generated by the previous command. I should also copy the **lut_pt_compress.txt** file to the same directory and rename it to **lut_HR_pt.txt**. Then I could run the commands

```
python mif_maker.py <path>/lut_HR_pt.txt lut_pt.mif
```

and

```
python programmable_lut_maker.py <path>/lut_HR_pt.txt lut_pt
    .xml
```

to get firmware and XML LUTs, respectively . I then repeat those two commands for **lut_HR_add_mult.txt** and **lut_HR_eta.txt**. Finally, I need to check these look good, and then send all the txt, mif, and xml files to Aaron to verify them.

20.4.4 Closure tests

After sending the LUTs to Aaron, he added them to a new integration tag that I had to pull. By going to **$CMSSW_BASE/src/** and undoing my edits to the params file and hackConditions, I did

```
git clone git@github.com:bundocka/cmssw.git
git remote add bundocka git@github.com:bundocka/cmssw.git
git fetch bundocka
git pull bundocka 2017v2JEC
git checkout 2017v2JEC
scram b -j 8
```

I added the new params file that contained the LUTs, **caloStage2Params_2017_v1_9_inconsistent_newJEC_cfi.py**, to **hackConditions_cff.py** and then added its details to **L1Trigger/Configuration/python/customiseSettings.py**. I then remade the CMSSW config from **L1JetEnergyCorrections/python/** with

```
cmsDriver.py -s RAW2DIGI --python_filename=
   l1NtupleMcMaker2017_RAW2DIGI_v2_closureTest.py -n 100 --
   no_output --no_exec --era=Run2_2017 --mc --conditions=92
   X_upgrade2017_TSG_For90XSamples_V1 --customise=L1Trigger/
   Configuration/customiseReEmul.
   L1TReEmulMCFrom90xRAWSimHcalTP --customise=L1Trigger/
   L1TNtuples/customiseL1Ntuple.L1NtupleRAWEMUGEN_MC --
   customise=L1Trigger/Configuration/customiseSettings.
   L1TSettingsToCaloStage2Params_2017_v1_9_inconsistent_newJEC
    --filein=/store/mc/PhaseISpring17DR/QCD_Pt-15
   to3000_TuneCUETP8M1_Flat_13TeV_pythia8/GEN-SIM-RAW/
   FlatPU0to70NZS_90X_upgrade2017_realistic_v20-v1
   /120000/003FF53C-8232-E711-9340-7CD30ACE160C.root
```

Then, as usual, I ran a test, added the new config and updated `job_append` in **crab3_stage2.py**, and ran it to generate the ntuples with corrections. Once finished, I ran the matching again. When that succeeded, I edited **binning.py** to make sure the other pileup bins were uncommented when it came to checking the calibrations and making plots. I also edited **submit_checkCalib_dag.py** and pointed the variable `PAIRS_FILES` to the pairs file that was just created. I could then run that script to check that the calibrations had been applied properly. The final stage was then producing plots for all pileup bins with

```
python showoffPlots.py --checkcal <path to output of
   checkCalib> --oDir <output directory>
```

and uploading them to the L1 Jets & Sums notebook on Evernote. I also have a copy of the plots in my Service Work directory. Note that to make plots with uncorrected jets, e.g., for comparisons,

I just need to run **submit_checkCalib_dag.py** and **showoffPlots.py** on the uncorrected ntuples. Unfortunately, the Layer-1 people have fucked up again! They didn't account for the 0.7 HF scale factor in the lowest p_T bin for the HF TPs. And so this round of JECs is invalid. However, I'll be able to follow basically the same procedure as this round, just using the updated Layer-1 corrections and some variant of the cmsDriver commands I used above.

20.5 Second round of JECs with proper corrections (end date – 07/11/2017)

With the new Layer 1 corrections, I could conduct the JECs again. This time, the important information is

– CMSSW version: 9.2.8

– Integration tag: `l1t-integration-v96.49`

– Dataset: /QCD_Pt-15to3000_TuneCUETP8M1_Flat_13TeV_pythia8/RunIISummer17DRStdmix-NZSFlatPU28to62_92X_upgrade2017_realistic_v10-v1/GEN-SIM-RAW

and I could clone CMSSW and set up the environment in the same way as before. I edited the params file to change the `jetCalibrationType` to `None`, added the file to **hackConditions_cff.py**, and added the new MC to **mc_samples.py**. To make the new CMSSW config file, I used the command

```
cmsDriver.py l1Ntuple -s RAW2DIGI --python_filename=
   l1NtupleMcMaker2017_RAW2DIGI_v3.py -n 420 --no_output --
   era=Run2_2017 --mc --conditions=92
   X_upgrade2017_realistic_v7 --customise=L1Trigger/
   Configuration/customiseReEmul.
   L1TReEmulMCFrom90xRAWSimHcalTP --customise=L1Trigger/
   L1TNtuples/customiseL1Ntuple.L1NtupleRAWEMUGEN_MC --
   customise=L1Trigger/Configuration/customiseSettings.
   L1TSettingsToCaloStage2Params_2017_v1_10_mode_inconsistent
    --filein=/store/mc/RunIISummer17DRStdmix/QCD_Pt-15
   to3000_TuneCUETP8M1_Flat_13TeV_pythia8/GEN-SIM-RAW/
   NZSFlatPU28to62_92X_upgrade2017_realistic_v10-v1
   /10000/00052042-ED9E-E711-A959-FA163E22945C.root
```

After testing the config with the `cmsRun` command, I updated the CRAB script to change `job_append`, add the new MC to `datasets`, and add the new CMSSW config to `PY_CONFIG`. I could then submit the jobs for the ntuples without corrections.

Once the initial ntuples were created and transferred to **/hdfs/L1JEC/**, I ran the matching. With no issues there, I could run the jobs to derive the calibrations. I just changed the pileup range from 40-50 to 50-60. After the jobs finished, I tried to run **runCalibration.py** to make the correction curves on the output root file. But, like before (which I forgot to document), I got a weird error of the form

```
TypeError: buffer is too small for requested array
```

Joe and I didn't figure out exactly what the problem was, but it was likely due to a Python/numpy problem in the 92X releases of CMSSW. So if, instead, I did `cmsenv` from my CMSSW_9_0_0_pre2 release, but then ran **runCalibration.py** from my 9_2_8 release, it would work. However, because of the few points in one of the p_{T} bins for $4.191 < |\eta| < 5.191$, we had to add a hack in the script:

```
if absetamin == 4.191:
    print "* WARNING: about to apply a JOE_HACK *"
    print "* messing with fit limits for 4.191<|eta|<5.191 *
  "
    max_ind = 17
    fit_max = xarr[max_ind]
    print max_ind
    print fit_max
    fit_min = 40.0
    min_ind = 0
```

I pushed that change (as well as the new pileup range) to my GitHub repo. Now that I had the correction curves, I had to massage them so the fits were good. Then, I could make the LUTs and give them to Aaron to incorporate them into the trigger code. He added them to his fork of CMSSW that I could just fetch and cherry pick to get the updated LUTs and params files in my working directory. I made sure the JECs were on and that **hackConditions_cff.py** pointed to it. Then I ran `cmsDriver` to get the new CMSSW config:

```
cmsDriver.py l1Ntuple -s RAW2DIGI --python_filename=
  l1NtupleMcMaker2017_RAW2DIGI_v3_closureTest.py -n 420 --
  no_output --era=Run2_2017 --mc --conditions=92
  X_upgrade2017_realistic_v7 --customise=L1Trigger/
  Configuration/customiseReEmul.
```

```
L1TReEmulMCFrom90xRAWSimHcalTP --customise=L1Trigger/
L1TNtuples/customiseL1Ntuple.L1NtupleRAWEMUGEN_MC --
customise=L1Trigger/Configuration/customiseSettings.
L1TSettingsToCaloStage2Params_2017_v1_10_mode_inconsistent
 --filein=/store/mc/RunIISummer17DRStdmix/QCD_Pt-15
to3000_TuneCUETP8M1_Flat_13TeV_pythia8/GEN-SIM-RAW/
NZSFlatPU28to62_92X_upgrade2017_realistic_v10-v1
/10000/00052042-ED9E-E711-A959-FA163E22945C.root
```

and made sure to update **crab3_stage2.py** to incorporate that and the new `job_append`. After running the script to make the ntuples with corrections, I transferred them to the correct area on **/hdfs** and ran the matching (adding the path to the new ntuples in the script). Once that had finished, I could check the calibrations. This involved adding the path to the new pairs root file in the script, and adding the 40-50 PU bin in **binning.py**. After running the script, I made the plots for the 40-50 and 50-60 pileup bins and added them to an Evernote note in the L1 Jets & Sums notebook. The plots are also in my Service Work directory on my iCloud Drive.

20.5.1 Follow ups

I was asked by Aaron to present on the JECs at a L1 DPG meeting. The main touchstones I had to add were an outline of the method and datasets/repos involved, plots of the correction curves in different regions of the detector, turn-on curves comparing old vs. new calibrations, and scatter plots comparing jet energies before and after calibrations. My slides are here: L1 Jet Energy Corrections (13-11-2017).

I was also asked to redo the JECs, using the "mean" params file (**caloStage2Params_2017_v1_10_mean_inconsistent.py**) that contained the Layer-1 calibrations, derived differently. Response curves, similar to the jet response curves we plot when deriving the JECs, are plotted for different objects in particular p_{T} and $|\eta|$ bins. Then, either the mean or mode response is used for the calibration of that object in the bin. The mean response had been used until fairly recently. But the potentially long tails in the plots can skew the values, and make the calibrations worse. So using the mode response instead gives the benefit of insensitivity to the tails. We could compare the performance of "mode" and "mean", relaying the better version to the DPG people so they know which to give us in the future. The workflow was the same as previously, just substituting the "mode" params for the "mean". The only other difference was that I added the LUTs to my CMSSW workspace myself. I had to put the **lut_HR_add_mult.txt** file I made in **L1Trigger/L1TCalorimeter/data/**. Then, in the "mean" params file, add the path to this LUT in

the variable `caloStage2Params.jetCalibrationLUTFile` (~L120). Then, I could make the CMSSW config, remake the ntuples, etc. to get the plots. From the results, it seems "mode" was better, and was what we requested for calibrations in 2018.

Olivier asked me to plot the "jet resolution" for the uncalibrated jets as a function of $|\eta^{\text{Ref}}|$, and compare the old params (with old MC) and new params (most recent MC, both "mean" and "mode"). He defined the jet resolution as

$$\sigma\left(\frac{p_{\text{T}}^{\text{L1}}}{p_{\text{T}}^{\text{Ref}}}\right) / \text{mean}\left(\frac{p_{\text{T}}^{\text{L1}}}{p_{\text{T}}^{\text{Ref}}}\right) \tag{20.1}$$

where the jets were binned in $|\eta^{\text{Ref}}|$. This wasn't too difficult, as the p_{T} ratio is in the pairs file made during the JECs, in the leaf "rsp". Note that some people call the standard deviation the "RMS" by mistake. In ROOT, there's a function called `GetRMS` which actually just calculates the standard deviation. The script I wrote is here:

```
from ROOT import TFile, TGraphErrors, TMultiGraph, TCanvas,
    TLegend
from array import array
from operator import truediv
from math import sqrt
from progressbar import ProgressBar, Percentage, Bar, ETA #
    requires progressbar package from pip

def initialisePlots(baseDir, multiGraph, MarkerColour,
    legend):

    file = TFile(baseDir+"/
    pairs_initialNtuples_ak4_ref10to5000_l10to5000_dr0p25.
    root")
    tree = file.Get("valid")
    nEntries = tree.GetEntries()

    entriesToRun = 5000000 # Number of entries in the tree
    to loop over
    if entriesToRun > nEntries:
        raise ValueError("entriesToRun: %i, should be less
    than nEntries: %i", entriesToRun, nEntries)
```

```
17      # Create arrays that store eta bins and half their
    widths (for symmetric error bars)
        etaRefArr = array("f", [-0.125, 0.125, 0.375, 0.625,
    0.875, 1.125, 1.375, 1.625, 1.875, 2.125, 2.375])
19      etaErrBars = array("f", [ 0.5*(etaRefArr[x+1] -
    etaRefArr[x]) for x in xrange( len(etaRefArr[1:]) ) ])

21      # Create blank arrays to hold jet information and number
        of entries
        ptRatio = array("f", ( [0.] * len(etaRefArr[1:]) ) )
23      ptRatioSq = array("f", ( [0.] * len(etaRefArr[1:]) ) )
        entriesPerBin = array("f", ( [0.] * len(etaRefArr[1:]) )
        )
25
        # Initialise progress bar
27      widgets = [Percentage(), Bar('>'), ETA()]
        pbar = ProgressBar(widgets = widgets, maxval =
    entriesToRun).start()
29
        for i in xrange(entriesToRun):
31          tree.GetEntry(i)
            # Account for jet saturation at pt = 1023.5 GeV
33          if tree.pt > 1023.0:
                continue
35          # Fill arrays with eta bins and jet information
            for j in xrange( len(etaRefArr) - 1 ):
37              if ( abs(tree.etaRef) > (etaRefArr[j] +
    etaErrBars[j]) and
                     abs(tree.etaRef) < (etaRefArr[j+1] +
    etaErrBars[j]) ):
39                  ptRatio[j] += tree.rsp
                    ptRatioSq[j] += tree.rsp ** 2
41                  entriesPerBin[j] += 1
                    break
```

```
43          pbar.update(i+1)

45      pbar.finish()

47      # Array of the mean jet response per eta bin
        ptRatioMean = array( "f", map(truediv, ptRatio,
    entriesPerBin) )
49      holdDiff = array("f", ( [0.] * len(etaRefArr[1:]) ) )
        pbar2 = ProgressBar(widgets = widgets, maxval =
    entriesToRun).start()
51
        for i in xrange(entriesToRun):
53          tree.GetEntry(i)
            if tree.pt > 1023.0:
55              continue
            for j in xrange( len(etaRefArr) - 1 ):
57              if abs(tree.etaRef) > etaRefArr[j] and abs(tree.
    etaRef) < etaRefArr[j+1]:
                    holdDiff[j] += (tree.rsp - ptRatioMean[j]) *
    * 2
59                  break
            pbar2.update(i+1)
61
        pbar2.finish()
63
        # Standard deviation of the repsonse in each |eta| bin
65      stdDev = array( "f", ( sqrt(y) for y in map(truediv,
    holdDiff, entriesPerBin) ) )
        # Standard deviation of response per eta bin, divided by
         the mean response in that bin
67      finalY = array ( "f", map(truediv, stdDev, ptRatioMean)
    )

69      myGraph = TGraphErrors(len(etaRefArr[1:]), etaRefArr
    [1:], finalY, etaErrBars)
```

```
    myGraph.SetMarkerStyle(3)
    myGraph.SetMarkerSize(1.5)
    myGraph.SetMarkerColor(MarkerColour)

    multiGraph.Add(myGraph, "p")
    if "withJEC" in baseDir:
        calib = "calibrated"
    else:
        calib = "uncalibrated"

    if baseDir == baseDirOld:
        legend.AddEntry(myGraph, "#splitline{Old params, %s.}{Events: %i}" % (calib, entriesToRun), "p")
    elif baseDir == baseDirNewMode:
        legend.AddEntry(myGraph, "#splitline{New params (mode), %s.}{Events: %i}" % (calib, entriesToRun), "p")
    elif baseDir == baseDirNewMean:
        legend.AddEntry(myGraph, "#splitline{New params (mean), %s.}{Events: %i}" % (calib, entriesToRun), "p")

    return multiGraph

multiGraph = TMultiGraph()
myLeg = TLegend(0.65, 0.7, 0.9, 0.9)

# After calibrations
#baseDirOld = "/hdfs/L1JEC/CMSSW_9_2_0/crab_qcdSpring17FlatPU0to70genSimRaw_qcdSpring17_genEmu_19Jun2017_920v/pairs/"
#baseDirNewMode = "/hdfs/L1JEC/CMSSW_9_2_8/crab_qcdSummer17FlatPU28to62genSimRaw_qcdSummer17_genEmu_03Nov2017_928/pairs/"
#baseDirNewMean = "/hdfs/L1JEC/CMSSW_9_2_8/crab_qcdSummer17FlatPU28to62genSimRaw_qcdSummer17_genEmu_14Nov2017_928/pairs/"
```

```
97 # Before calibrations
   baseDirOld = "/hdfs/L1JEC/CMSSW_9_2_0/
      crab_qcdSpring17FlatPU0to70genSimRaw_qcdSpring17_genEmu_13Jun2017_920v
      /pairs/"
99 baseDirNewMode = "/hdfs/L1JEC/CMSSW_9_2_8/
      crab_qcdSummer17FlatPU28to62genSimRaw_qcdSummer17_genEmu_27Oct2017_928
      /pairs/"
   baseDirNewMean = "/hdfs/L1JEC/CMSSW_9_2_8/
      crab_qcdSummer17FlatPU28to62genSimRaw_qcdSummer17_genEmu_11Nov2017_928
      /pairs/"
101
   initialisePlots(baseDir = baseDirOld, multiGraph =
      multiGraph, MarkerColour = 2, legend = myLeg) # colour =
      red
103 initialisePlots(baseDir = baseDirNewMode, multiGraph =
      multiGraph, MarkerColour = 4, legend = myLeg) # colour =
      blue
   initialisePlots(baseDir = baseDirNewMean, multiGraph =
      multiGraph, MarkerColour = 3, legend = myLeg) # colour =
      green
105
   canv = TCanvas("canv", "canv", 600, 600)
107 canv.SetGrid()
   multiGraph.Draw("a")
109 multiGraph.SetTitle("Jet Energy Resolution; |#eta^{Ref}|; #
      sigma(pt_{jet}^{L1}/pt_{jet}^{Ref}) / mean (pt_{jet}^{L1
      }/pt_{jet}^{Ref})")
   canv.SetLeftMargin(0.15)
111 multiGraph.GetYaxis().SetTitleOffset(1.7)
   myLeg.Draw()
113 canv.SaveAs("jet_energy_resolution.pdf")
```

Listing 20.2: Plot the jet resolution as a function of η. File name: plot_jet_resolution.py.

The progress bar is from a Python package called "progressbar". I can install it with `pip install -user progressbar` locally and on remote servers. The graph my script produces is here:

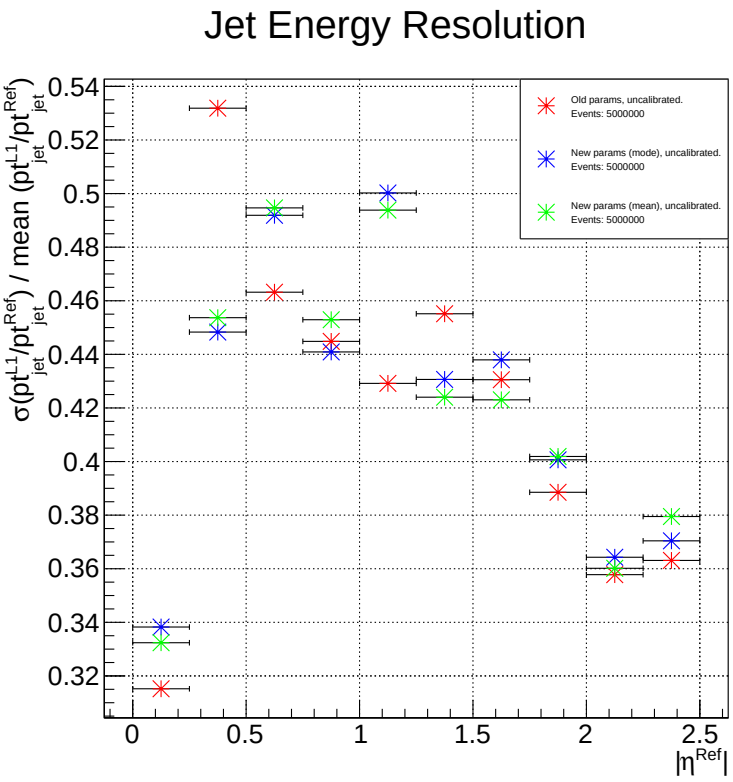

Figure 20.1: The jet energy resolution for the old params (red) and new params (blue) for different eta bins.

20.6 CMS Week presentation on the Jet + MET status (05/12/2017)

I've been asked to give a presentation during CMS Week about the 2017 status and 2018 plans of the jets and MET operations conducted by those in Calo Layer-2. The slides are linked here: L1 Jet MET talk (05-12-2017).pdf.

20.7 Third round of JECs (end date – 17/04/2018)

As we've recently started the 2018 run of the LHC, we need calibrations to be conducted promptly so that the LUTs with scale factors can be deployed to the firmware, and data can be swiftly reconstructed and validated. The components I need are as follows:

– CMSSW version: 10.0.3

– Integration tag: `add100XLayer1SF` (encompasses tags `v97.20 - 97.22` rebased from CMSSW_10_0_0)

– Dataset: /QCD_Pt-15to3000_TuneCP5_Flat_13TeV_pythia8/RunIISpring18DR-NZSPU0to70_100X_upgrade2018_realistic_v10-v1/GEN-SIM-RAW

I could set up the environments in a similar way to before, but with some additions from various pull requests, bug fixes and rebases:

```
cmsrel CMSSW_10_0_3
cd CMSSW_10_0_3/src
cmsenv
git cms-init
git cms-addpkg L1Trigger/Configuration
git cms-addpkg L1Trigger/L1TCalorimeter
git cms-addpkg L1Trigger/L1TCommon
git cms-addpkg L1Trigger/L1TMuon
git clone https://github.com/cms-l1t-offline/L1Trigger-
   L1TMuon.git L1Trigger/L1TMuon/data
git clone git@github.com:eshwen/L1JetEnergyCorrections.git
   L1Trigger/L1JetEnergyCorrections
git remote add bundocka git@github.com:bundocka/cmssw.git
git cms-merge-topic bundocka:add100XLayer1SF
git clone https://github.com/cms-l1t-offline/L1Trigger-
   L1TCalorimeter.git L1Trigger/L1TCalorimeter/data
scram b -j 8
```

I changed the jet calibration type in **caloParams_2018_v1_1_inconsistent_cfi.py** and imported the file in **hackConditions_cff.py**. Then, I navigated to **L1JetEnergyCorrections/python/** and used the `cmsDriver.py` command

```
cmsDriver.py l1Ntuple -s RAW2DIGI --era=Run2_2018 --mc --
   python_filename=l1NtupleMcMaker2018_RAW2DIGI_v1.py --
   no_output -n 202 --conditions=100
   X_upgrade2018_realistic_v11 --customise=L1Trigger/
   Configuration/customiseReEmul.L1TReEmulMCFromRAWSimHcalTP
    --customise=L1Trigger/L1TNtuples/customiseL1Ntuple.
   L1NtupleRAWEMUGEN_MC --customise=L1Trigger/Configuration/
   customiseSettings.
```

```
L1TSettingsToCaloParams_2018_v1_1_inconsistent --
custom_conditions=HcalChannelQuality_2018_v3.0_mc,
HcalChannelQualityRcd,frontier://FrontierProd/
CMS_CONDITIONS --filein=/store/mc/RunIISpring18DR/QCD_Pt
-15to3000_TuneCP5_Flat_13TeV_pythia8/GEN-SIM-RAW/
NZSPU0to70_100X_upgrade2018_realistic_v10-v1/100000/00818
B45-1522-E811-910B-1866DAEA7E64.root
```

to create the CMSSW config for the CRAB jobs. After testing with `cmsRun`, I then added the new dataset to **mc_samples.py** and linked to it in **crab3_stage2.py**, as well as updating `job_append` and adding the CMSSW config to `PY_CONFIG`. After initialising the CRAB environment, I submitted the initial ntuples with

```
python crab3_stage2.py
```

I have also been asked to perform the JECs with ECAL zero suppression, which involved the same steps and workflow, but using the calo params file **caloParams_2018_v1_1_ECALZS_inconsistent_cfi.py**. The cmsDriver command was identical, apart from the output file name (in which I appended "ECAL_ZS"), and the attribute of `L1Trigger/Configuration/customiseSettings` which I set to `L1TSettingsToCaloParams_2018_v1_1_ECALZS_inconsistent`.

When tuning the fits for the curves made with the normal params, they were pretty terrible and wouldn't tune properly to capture the peak and tail of each curve. Joe suggested changing the values used in the fitting function in **runCalibration.py**. For a previous round of JECs (where the initial fits were decent), I found the root file and an accompanying LUT text file containing the parameters used in the fit. So I copied one of those lists and changed the one in the file to

```
STAGE2_DEFAULT_PARAMS_JETMETERR = [1.86431, -1.34016e+06,
   -2.85506e-08, 20.2633, -6.40935e-07, -1.54, 1.06511]
```

Then, I reran **runCalibration.py** to get the root file with better fits I was able to tune. As with the CMSSW_9_2_X JECs, I got the weird error `buffer is too small for requested array` when running **runCalibration.py**. Again, I had to source my CMSSW_9_0_0_pre2 release and run in that environment. I also had to apply another "Joe hack" for $2.964 < |\eta| < 3.489$.

Once the calibrations were complete and the LUTs were made, I needed to perform the closure tests. To remake the ntuples with the calibrations, I needed to copy my **lut_HR_add_mult.txt**, **lut_HR_pt.txt** and textbflut_HR_eta.txt to **L1Trigger/L1TCalorimeter/data/** (giving the files some kind of unique marker to avoid confusion) and add the paths to those files in the calo params file. The variable `caloStage2Params.jetCalibrationLUTFile` requires the path to

lut_HR_add_mult.txt, then `jetCompressPtLUTFile` and `jetCompressEtaLUTFile` require the pt and eta LUTs, respectively. Then, I needed to change `jetCalibrationType` to turn the JECs on. The cmsDriver command was the same as when generating the config for the initial ntuples, just with a different `python_filename`. The same recipe could be applied to JECs derived with the ECAL ZS params. I presented an update at the DPG meeting: L1 JEC comparison (2017 vs early 2018).pdf.

20.7.1 Follow ups

As outlined in the presentation, there was a problem with HF jet resolution among some other stuff. I was asked to make some scatter plots comparing different jet collections (GenJet vs offline CaloJet, Gen vs PFJet, PF vs Calo, PF vs L1Jet, and Calo vs L1Jet) in HF and in several E_{T} ranges. This was so we could get a rough idea of how these different offline or "true" measurements of jet p_{T} are related to each other and L1. Luckily, the L1JetEnergyCorrections repo has some support for this. In **submit_matcher_dag.py**, around L75, I could just change the `SAMPLE` variable and it would intelligently pick the correct C++ macro to run the matching out of the available options. And I could run it on the corrected or uncorrected ntuples.

When comparing PF and GenJets, I had to edit some of the code in **RunMatcherStage2PFGen.cpp**, replacing some out-of-date stuff and making it similar to **RunMatcherStage2L1Gen.cpp**. Namely, I had to update the names of the trees that house the properties of some of the collections and fix some variable names. Then, I had to write cpp files for each jet collection comparison in the same vein. All my edits will be in the commit logs for the repo. I also had to remember to compile everything after editing the C++ code and adding the new files to **BuildFile.xml** in the same directory, which also pointed out errors I needed to fix.

I could run the matcher like normal after the edit, as well as the calibration check (but making sure to rename the original pairs and checked root files so they weren't overwritten). Then, in **showoffPlots.py**, I just needed to make sure `l1_str` and `ref_str` matched the collections I was comparing. I could also set the E_{T} ranges in the relevant functions.

Originally, the PF and offline Calo objects (reco) weren't included in the ntuples, so Aaron remade them using the GEN-SIM-RAW and AOD formats of the QCD dataset.

20.8 LHCP

I was asked to present a poster at the LHC Physics (LHCP) conference in Bologna in June 2018. The poster title was titled "The CMS Level-1 jet and energy sum trigger for the LHC Run II". This mainly

involved presenting the algorithms used for JetMET in Run-2 and the performance of the triggers in 2016 and 2017. My final version of the poster is linked here: LHCP poster JetMET 2018 v4.pdf. After the conference, I submitted the poster to the proceedings.

Chapter 21

Using SSHFS to mount remote servers (01/03/2017)

SSHFS (SSH File System) is a client that allows you to mount folders from remote servers onto a computer as if they were a disk. This allows you to edit your files located on the sever using software on your computer. So I can use Visual Studio Code instead of emacs, and other GUI programs. It's immensely helpful and makes everything about working on a remote server much easier, and it also isn't too hard to set up. I can mount specific folders or directories, make changes, and it is instantly synced to the server (because mounting the directory only links you to it, it isn't copied over and then synced when changes are made like with iCloud Drive). So to use SSHFS, I need to open my terminal and install the following packages with Homebrew:

```
brew install automake
brew install libtool
brew install autoconf
brew install pkg-config
brew install glib
brew install Caskroom/cask/osxfuse
# restart computer, then log back into terminal
brew tap homebrew/homebrew-fuse
brew install sshfs
```

Then, locally, I should make the directory **~/sshfs/soolin** (because at the moment I'll just be using SSHFS with Soolin). As normal, I would have to either be on the university's network or connected via a VPN to access Soolin. To connect to and mount the directory that's located on the

remote server, type

```
sshfs <username>@<hostname>:<remote path> <local path to
    mount point> -o reconnect,allow_recursion,local,
    allow_other,follow_symlinks,volname=<local volume name>
```

A shortcut to the mount will also be displayed on my Desktop like when normal drives are connected. As an example, If I want to mount my **/storage/eb16003/** directory, I can type

```
sshfs soolin:/storage/eb16003/ ~/sshfs/soolin/ -o reconnect,
    allow_recursion,local,allow_other,follow_symlinks,volname
    =storage.eb16003
```

I can use the alias I have for the host, as shown above. So I can type `sshfs soolin` rather than `sshfs eb16003@soolin.dice.priv`. Using the alias also allows me to circumvent the password prompt. In some cases when trying to mount or unmount, the *absolute* local path is needed. If I get weird errors, try using it.

When I want to unmount the directory, type

```
umount -f <local path to mount point>
```

So in the example above, I would type `umount -f ~/sshfs/soolin` to unmount my **/storage/eb16003/** directory. I've included the mounting and unmounting commands in shell scripts (**soolin_mount.sh** and **soolin_unmount.sh**, respectively) located in **~/.sshfs/**, so I can just source them to mount and unmount directories. And I would only need to edit the remote path in **soolin_mount.sh** if I wanted to mount a different directory.

I will likely only need to edit files located on **/storage** or **/users**. But if I'm jumping back and forth to different directories on Soolin, it would be simpler to just use the terminal like normal. I could have the file system mounted to edit files, and also have the terminal open to navigate between the directories that aren't mounted.

I've also added scripts to mount and unmount my home directory on Imperial's remote server, as well as CERN's lxplus. These are in the same folder as the Soolin scripts, are named **imperial_mount.sh** and **imperial_unmount.sh** for Imperial, and **lxplus_mount.sh** and **lxplus_unmount.sh** for lxplus.

Chapter 22

Physics PGR conference and Annual Progress Monitoring (20/04/2017)

Every year, a postgraduate researchers conference is held where students demonstrate what they've been doing so far. First year PGRs are tasked to make a poster detailing their projects, stand next to it for certain amount of time, and explain things and answer questions to people who view it. This year's conference is on 10th May. My poster (actual size is A0) is displayed below.

22.1 Annual Progress Monitoring (07/06/2017)

Annual Progress Monitoring is used to ensure that PhD students have made sufficient progress to move forward to the following year. For first year students, a report has to be written detailing the progress made over the year, as well as future plans. A presentation to the HEP group must also be made (in my case). My report and presentation are linked here: First Year Report and First Year Presentation.

From the viva/interview, there were several comments and aspects I need to take into account.

- I need to know the entire process of DM production from the protons (and partons) within the beam, to how they interact and produce the particles (including sparticles), to the specifics of the analysis.
- That we use *transverse* energy and momentum because – for, e.g., SUSY production – quark-antiquark annihilation is the predominant collision. Then most of decay products are jets because QCD is the favoured interaction; it's called the "strong" force for a reason. And you

don't know the exact longitudinal momentum of the partons, but we know that the transverse momentum before the collision (which has to equal the momentum after) is zero because the protons are travelling completely longitudinally. The beams have equal and opposite momentum but that doesn't hold for the individual partons within each proton in the beam. Quarks tend to have a greater proportion of the momentum over the antiquarks (because protons are predominantly matter, not antimatter), which the Parton Distribution Functions show. The normalisation of PDFs require that the valence quantum numbers for a particle are reproduced. So for a proton, the integral over the total momentum of the difference in probability of a u carrying a certain momentum and a $\bar{u}$ carrying an equal momentum yields 2 (the number of valence us in the proton). So, if $f_q(x)$ is the probability of a quark flavour q carrying a momentum fraction x of the proton's longitudinal momentum,

$$\int_0^1 dx(f_u(x) - f_{\bar{u}}) = 2 \tag{22.1}$$

as described in slide 6 of https://www2.physics.ox.ac.uk/sites/default/files/2014-03-31/qcdgrad_rojo_oxford_tt14_5_dis_pdf_19197.pdf or section 2 of [62].

The Higgs is normally produced from gluon-gluon fusion (with a triangle of, normally, top quarks that decays into the Higgs). But that doesn't work for SUSY particles because the gluons don't usually have enough energy to produce these particles, so it's $q\bar{q}$ for the most part.

– The unique thing about RA1 is the use of α_{T}. It cuts QCD heavily by constraining jet imbalance, so removes a lot of potential "fake MET" from misreconstructed jets. The figure below (from [45]) shows that QCD-driven final states drop sharply at around $\alpha_{\mathrm{T}} = 0.5$, and so we tend to make cuts a little higher than this value to reduce the QCD background to a much lower level. This allows a cleaner search for SUSY by using MET as a variable. We're sensitive to strongly-coupled SUSY. The use of $\Delta\phi^*_{\mathrm{min}}$ greatly reduces QCD as well, with the reference above containing a corresponding plot of the events vs $\Delta\phi^*_{\mathrm{min}}$.

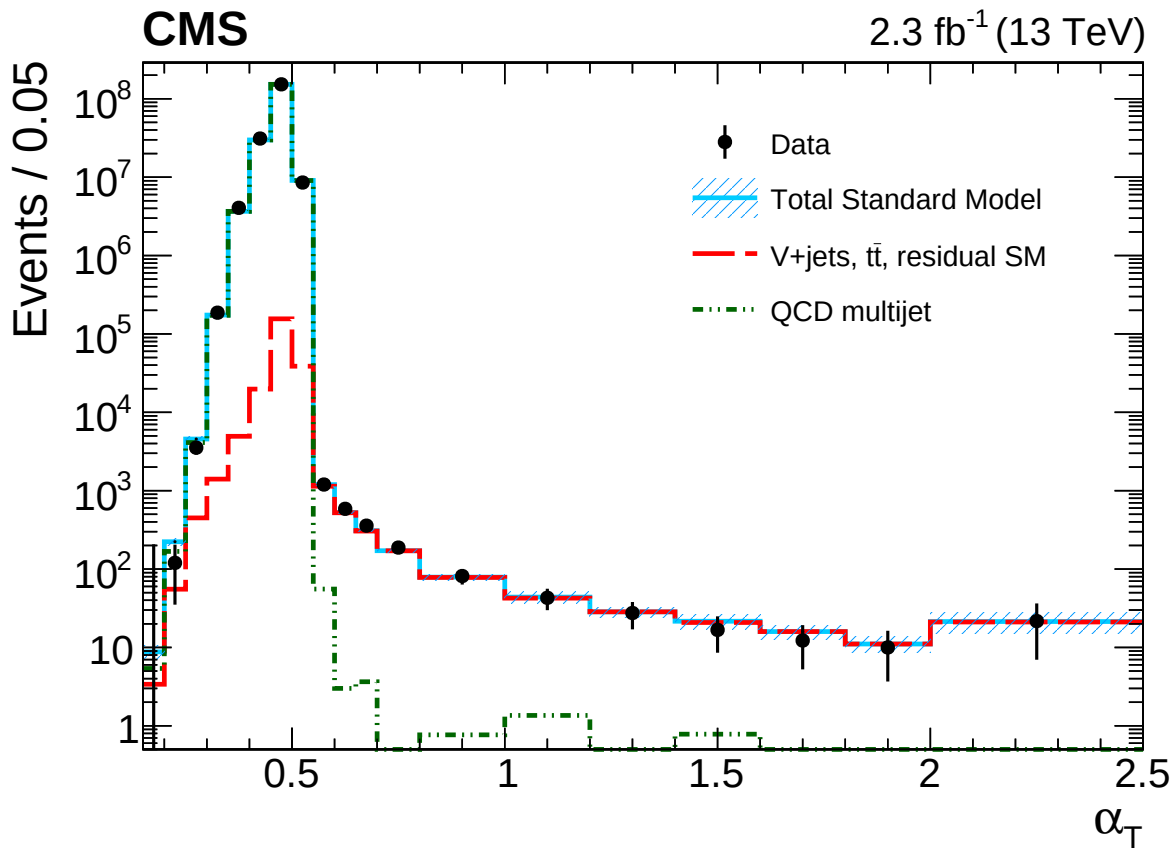

Figure 22.1: The α_T distribution observed in data for events that satisfy the selection criteria defined in the text. The statistical uncertainties for the multijet and SM expectations are represented by the hatched areas (visible only for statistically limited bins). The final bin of the distribution contains the overflow events.

- RA1 looks at all hadronic final states because of the SUSY models we use. The gluinos and squarks decay into quarks, hence lots of jets and the need for α_T. We use "natural" SUSY models because of the minimal fine-tuning of the bare Higgs mass that only need the gluino, third-generation squarks (stop and sbottom) and a Higgs-like LSP around the electroweak scale. These models are motivated by the discovery of a light Higgs boson.
- Need to research how the reference jets (GenJets) for JECs are made. Broadly, some people make unbiased Monte Carlo jets (i.e., before trigger cuts) that the L1 jets can be calibrated to. After some research, I've found some good descriptions in [63].
- That I should get more involved with the Trigger in the future. I'll pass on the JEC stuff to a younger student (like Joe has done to me) and get more involved in the development and stuff. It's also helpful that Bristol is so involved in the Trigger.
- My language is a little "loose" in the report. Need to take into account that I'm in a field where

statements need to be very precise, as people scrutinise them, and English may not be the native language of some of the readers, so can misinterpret what I mean. So, things like the first sentence can be tightened up.

– At some point, I need to talk to Bjoern and Henning about how their supervisory roles are going to change because of Bjoern leaving. Presumably, Henning will become more involved with what I'm doing while Bjoern steps back a bit.

– Overall, Dave (Newbold) and Jonas were quite happy with how I've progressed so far, and have "high hopes for someone of your ability", and am "someone who can make a career out of this". I've been doing "pretty bloody well" according to Dave. They're happy that I've gotten quite involved in the analysis side of things and with the trigger.

If I want to read the interviewer and supervisor's comments (as well as my own), I can do so at https://dlm.chm.bris.ac.uk/apm.

Dark Matter Searches at CMS at $\sqrt{s}$ = 13 TeV

University of BRISTOL

Eshwen Bhal

Supervisor: Bjoern Penning

Dark matter searches will be conducted with the CMS experiment at the LHC using Run-2 data at a centre-of-mass energy of 13 TeV. Supersymmetry is currently the most popular theory that accommodates dark matter candidates, and will be the main focus of this PhD. The majority of the work conducted will be analysis-based and may include interdisciplinary aspects such as astrophysical dark matter searches. Projected results from this undertaking would involve setting world-leading limits on the masses of dark matter and its mediator in different models, and possibly discovering other properties like its production frequency at the LHC.

Introduction

The Universe is filled with a non-baryonic form of matter labelled "dark" matter. This substance does not interact with any of the four fundamental forces, apart from gravity. Whilst evidence and observations have been astrophysical in nature (Figs. 1 and 2), understanding the properties of dark matter shifts the responsibility to particle physics.

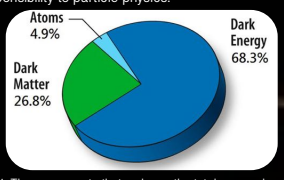

Fig. 1: The components that make up the total energy density of the Universe. These values are taken from the Planck satellite.

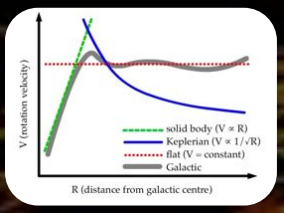

Fig. 2: An observed galactic rotation curve (grey) with standard kinematic curves. A source of invisible matter must dominate to explain the roughly flat rotation curves seen in astronomy.

Experimental setup:

CERN's Large Hadron Collider (LHC) has allowed us to probe high energies and create exotic particles that existed when the Universe was hot; the temperature was high enough to produce these particles – including dark matter – in abundance. Once it cooled, these particles were no longer spontaneously created: a thermal freeze out occurred. [1] The Compact Muon Solenoid (CMS) experiment at the LHC aims to detect the signatures of dark matter production from various theories, such as Supersymmetry. With a projected 150 fb^{-1} of data from Run-2 at a centre-of-mass energy of 13 TeV (tera-electron volts), we may learn much more about dark matter.

Theory

Supersymmetry (SUSY) is the leading candidate for physics beyond the Standard Model of Particle Physics. [2] It introduced a spin symmetry that predicts a fermionic superpartner for each boson, and vice versa. If the lightest supersymmetric particle (LSP) is stable and electrically neutral, it would provide a promising dark matter candidate.

Fig. 3: A Feynman diagram depicting dark matter (DM) pair production from Standard Model (SM) particles. Within the circle are the subprocesses that produce dark matter.

The conservation of *R*-parity [2] means that the decay cascades of SUSY particles produced in a collider ends with the LSP, and Standard Model particles with each decay. These are normally energetic hadrons, which then create showers of new hadrons from pair production that are detected as "jets". Due to dark matter LSPs being undetectable, the reconstructed event from the detector would contain "missing" transverse energy (MET) which would be required to satisfy energy and momentum conservation. The predominant sources of MET in collider events are from neutrinos, misreconstructed particles, and possibly new particles.

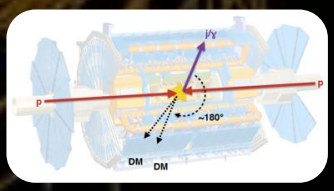

Fig. 4: An example of a *pp* collision at an accelerator producing dark matter (reconstructed as MET) and recoiling SM particles.

So the characteristics of SUSY in a collider would be high MET from the LSPs, and several hadronic jets. This is something CMS is looking for.

Proposed method

I am working with CMS in inclusive searches for physics beyond the Standard Model (SUSY being one aspect). These, in principle, require three sets of data: the data from *pp* collisions at the LHC; the expected signal from particle decays due to new physics; and estimates of the background events from Standard Model processes (to distinguish the signal from the background).

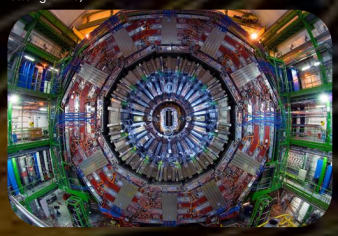

Fig. 5: The CMS detector at the LHC.

Much of the work will be analysis, either sifting through the LHC data or generating the simulated signal/background. By the end of this PhD, the entire Run-2 dataset will be available, providing an enormous number of collider events.

Other sources of knowledge could also be used. Astrophysical observations have shown several independent channels that prove its existence [3], and there are more experiments to explore them further. These can be combined with our efforts to give us much greater scope in the hunt for dark matter.

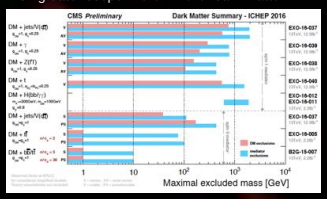

Fig. 6: Limit plots of the dark matter mass and that of its mediator from several analyses. [4] Masses below these values are excluded. Specific models and decay channels were used, and these are currently the world-leading mass measurements.

Projected results

There are many aspects in which our understanding of dark matter can be improved. Over the course of this PhD, my intention is to be able to constrain some of its characteristics. With the full Run-2 dataset, it will be possible to set the world's best limits on the masses of dark matter particles in different theory frameworks and from different decay modes. These could be pushed higher than those set by Figs. 6 and 7 by orders of magnitude. I may also be able to determine how often it is produced at the LHC, the favoured decay modes that produce it, and devising methods to accurately detect these modes. Any progress in these areas will aid in future endeavours to further understand dark matter.

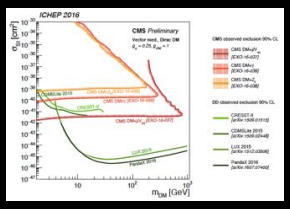

Fig. 7: Limit plots of the spin-independent production cross section (effectively the likelihood to produce the particle) as a function of dark matter mass from several experiments. [4]

[1] Michael E. Peskin. "Dark matter and particle physics", in: *J. Phys. Soc. Jap.* 76 (2007).
[2] Stephen P. Martin, "A Supersymmetry Primer", in: *Adv. Ser. Direct. High Energy Phys. 18* (1998).
[3] Hitoshi Murayama. "Physics Beyond the Standard Model and Dark Matter", arXiv: 0704.2276 [hep-ph] (2007).
[4] CMS Collaboration, "Dark Matter Summary Plots from CMS for ICHEP 2016".

Figure 22.2: PGR conference poster.

Chapter 23

Signal model analyses for SUS-16-038 (21/04/2017)

My next task is to learn about the signal models used in RA1's 2016 paper (SUS-16-038 – reinterpretation of SUS-16-016 analysis with full 2016 dataset), which encompasses the SUSY SMS models, and HF DM (heavy flavour dark matter) models. The HF DM models are so called because the final state includes a heavy quark, i.e., a t or b. They're usually labelled in the fashion "DMttP" for a $t\bar{t}$ final state and pseudoscalar mediator. For private samples (not needed for this), there will be a "p" prefix, e.g., "pDMttP". I need to make a list of all of these models, whether there are trees already – and where they are – and which trees haven't been made yet. Then I can create the trees (I'll need to know what cuts, etc. to apply) for the remaining models and conduct some analysis; for the moment this will comprise checking the acceptance, systematics, efficiencies, etc. Beyond that I'll need to derive some results, but the above is a lot of work for the immediate future, so I can worry about that later on.

The paper is currently being written, so isn't available yet. From my perspective, I can use information from the SUS-16-016 Analysis Note (AN-16-161, http://cms.cern.ch/iCMS/jsp/db_notes/noteInfo.jsp?cmsnoteid=CMS%20AN-2016/161) to learn about how the analysis is actually completed. The relevant information is in sections 16 "Interpretation in Simplified SUSY models" and 17 "Interpretation in Dark Matter models". These indicate the models and samples used, as well as technicalities and the context surrounding the signal we're analysing. The Analysis Note for SUS-16-038, being written in tandem with the paper, is here: http://cms.cern.ch/iCMS/jsp/db_notes/noteInfo.jsp?cmsnoteid=CMS%20AN-2017/122.

I can start by initialising a CMSSW environment and fetching the relevant repos:

```
source /cvmfs/cms.cern.ch/cmsset_default.sh
cd /storage/eb16003/CMScmg
cmsrel CMSSW_8_0_25
cd CMSSW_8_0_25/src
cmsenv
export CMSSW_GIT_REFERENCE=/software/SUSY/RA1/cmg-cmssw-bare
git cms-init
git remote add ra1-private git@github.com:CMSRA1/cmg-cmssw-
   private.git
git remote add cmg-central https://github.com/CERN-PH-CMG/
   cmg-cmssw.git
git fetch ra1-private
echo .gitignore >> .git/info/sparse-checkout
echo PhysicsTools/Heppy/ >> .git/info/sparse-checkout
echo PhysicsTools/HeppyCore/ >> .git/info/sparse-checkout
echo EgammaAnalysis/ElectronTools/ >> .git/info/sparse-
   checkout
echo RecoEgamma/ElectronIdentification/ >> .git/info/sparse-
   checkout
echo RecoEgamma/PhotonIdentification/ >> .git/info/sparse-
   checkout
# The following echoes are specific to SMS tree production
echo RecoBTag/Configuration/ >> .git/info/sparse-checkout
echo RecoBTag/DeepFlavour/ >> .git/info/sparse-checkout
echo RecoBTag/LWTNN/ >> .git/info/sparse-checkout
echo RecoBTag/SecondaryVertex/ >> .git/info/sparse-checkout
echo PhysicsTools/PatAlgos/ >> .git/info/sparse-checkout
echo DataFormats/BTauReco/ >> .git/info/sparse-checkout
git config core.sparsecheckout true
git checkout heppy_80X_ra1-0.7.x-Moriond17Prod
echo CMGTools/ >> .git/info/exclude
git clone -o ra1-private git@github.com:CMSRA1/cmgtools-lite
   -private.git CMGTools
cd CMGTools
git remote add cmg-central git@github.com:CERN-PH-CMG/
```

```
   cmgtools-lite.git
git checkout 80X-ra1-0.7.x
git submodule init
git submodule update
git checkout 80X-ra1-0.7.x-Moriond17Prod
cd $CMSSW_BASE/src
scram b -j 9
```

Then I should have all the code necessary for tree production. The default branch I'll be on in **$CMSSW_BASE/src/** will be `heppy_80X_ra1-0.7.x`. In **./CMGTools/** there are two branches: `80X-ra1-0.7.x` and `80X-ra1-0.7.x-Moriond17Prod`. The former contains lists of the SMS and DM samples from Spring16, but need to be updated to Summer16 where applicable. The latter has been used to produce the DM samples from Summer16 (see **CMGTools/RA1/python/components/components_MC_DM_Summer16.py**).

I also cloned a fresh copy of AlphaTools for analysing the trees:

```
cd /storage/eb16003/CMScmg
git clone -o alphatools git@github.com:CMSRA1/AlphaTools.git
    --recursive
cd AlphaTools/analysis
source setup.sh
cd ..
git pull alphatools v1.10.x
git submodule update
git submodule init
```

and just need to source **setup.sh** in **AlphaTools/analysis** to start using it. In parallel, I created the same set up on IC's lxoo. The commands are basically the same as above, but L6 is instead `export CMSSW_GIT_REFERENCE=/vols/cms/RA1/cmg-cmssw-bare` The equivalent of /storage/eb16003/CMScmg/ on Soolin is **/home/hep/ebhal/CMScmg_imperial/** on lxoo. I also had to set up my grid certificate on this server.

I've made scripts that source and initialise everything: **~/cmssw_2016signalmodels.sh** on Soolin; and **~/cmssw_2016signalmodels_imperial.sh** at Imperial. However, I shouldn't initialise everything together because each framework uses different environments and environment variables, which can screw up whatever I'm doing. I've only collected the commands in a single script for reference purposes. In practice, I'll just comment out the lines for the software I don't want to load.

23.1 Making the DM and SUSY SMS trees

For making SMS model trees I'll need the config file **run_AtLogicNoSelection_MCMiniAODv2_SUSY_SMS_FastSi** and for DM trees I'll need **run_AtLogic_MCSummer16_DM_cfg.py**, both located in **CMGTools/RA1/cfg**. So I just need to update the components/models to the latest versions, make the component lists to be read by the configs, and then run the tree production for the remaining models.

At IC, the path **/vols/cms/RA1/80X/MC/20170210_SUSY_DM_PreAppr/** contains the trees of the currently-produced signal models. The following models are contained in these subdirectories:

- **AtLogic_MCSummer16_DM/**: HF DM scalar and psuedoscalar models (Summer16, OBSELETE).
- **AtLogicNoSelection_MCMiniAODv2_SUSY_SMS_FastSim/**: SUSY SMS T1bbbb, T2cc, and T2tt models (Spring16, OBSELETE).

For now, I just need to produce the trees for the T2bb SMS model from Spring16. For the paper, we would (in theory) need all the models to be from Summer16, but the SUSY model makers decided it wasn't worth updating the Spring16 models because the changes wouldn't be significant. Making the trees for T2bb was fairly simple. In the cfg file I just had to comment out everything except `componentList.append(cmps.SMS_T2bb_madgraphMLM)` in the function `componentList = [ ]`. Then I recompiled, committed everything and tagged the code (which is **80X_MC_20170426_S01** for the T2bb Spring16 trees). To submit the jobs, the command was

```
heppyBatchAlphaT.py -o $OUTDIR -c
    AtLogicNoSelection_MCMiniAODv2_SUSY_SMS_FastSim
```

Then I extracted the output, resubmitted failed jobs, then combined them when everything succeeded. At Bristol, the trees are located at

- Bristol: **/hdfs/SUSY/RA1/80X/MC/20170426_S01** (T2bb Spring16, OBSELETE).
- Imperial: **/vols/cms/RA1/80X/MC/20170426_S01** (T2bb Spring16, OBSELETE).

23.2 Remaking the SMS and DM trees

However, there was a problem with the trees that I encountered when trying to analyse them. In the SMS trees, all of the SUSY particle masses were "lumped" together, and so the mass points weren't saved in the trees (like m_{SUSY} and m_{LSP} for events). So Shane submitted a "pull request" on GitHub – basically committing changes, but allowing them to be reviewed before being merged into a branch/repo, which is generally good practice, especially when modifying code in important

branches – which includes the SUSY particle masses and isolated track variables. Then I just needed to merge it, and in my workspace on Soolin/lxoo, I just needed to do `git pull ra1-private 80X-ra1-0.7.x-Moriond17Prod` to get the updated code. After that, I needed to remake all the SMS trees with these inclusions. In the config file, I just had to make sure all of the components were uncommented, then I could submit jobs and combine the output, etc. to make the trees. After another pull request from Shane to fix the DM tree production, I could submit the DM jobs as well. A few DM jobs were consistently failing due to an empty **pycfg.py** file. As with tree production for the cut flow tables, I could copy the filled **pycfg.py** from a successful job into the directory of a failed one (as long as it is from the same sample), and then resubmit them. The tag of the code for both SMS and DM tree production is **80X_MC_20170510_S01**. The new trees are located at

- Bristol (SUSY SMS): **/hdfs//SUSY/RA1/80X/MC/20170510_SMS-Spring16_DM-Summer16/AtLogicNoSelection_MCMiniAODv2_SUSY_SMS_FastSim/** (T1bbbb, T2bb, T2cc, T2tt; Spring16).
- Bristol (DM): **/hdfs/SUSY/RA1/80X/MC/20170510_SMS-Spring16_DM-Summer16/AtLogic_MCSummer16_DM/** (DMttP, DMttS; Summer16).
- Imperial (SUSY SMS): **/vols/cms/RA1/80X/MC/20170510_SMS-Spring16_DM-Summer16/AtLogicNoSelection_MCMiniAODv2_SUSY_SMS_FastSim/** (T1bbbb, T2bb, T2cc, T2tt; Spring16).
- Imperial (DM): **/vols/cms/RA1/80X/MC/20170510_SMS-Spring16_DM-Summer16/AtLogic_MCSummer16_DM/** (DMttP, DMttS; Summer16).

The mass points were still merged into one tree per model for production, so they needed to be split such that there was a tree for each mass point. This was done using the TreeSplitterSusy macro in AlphaTools. For whatever reason, this wasn't working with my set up so Shane split the trees. These new "split" trees are stored in

- **/vols/cms/RA1/80X/MC/20170510_SMS-Spring16_DM-Summer16/AtLogicNoSelection_MCMiniAODv2_SUSY_SMS_FastSim/SMS_splitMasses/** (T1bbbb, T2bb, T2cc, T2tt Spring16 split trees).

23.3 Getting the StatsInput for the SMS models (Spring16 datasets)

I've set up AlphaTools (v1.10.x branch) on Imperial's remote server (lx00/lx01). The main code is located in **~/CMScmg_imperial/AlphaTools/analysis/**, which also has the alias **$ALPHATOOLS-**

DIR. Using this, I can find the systematic uncertainties for the SMS models. First I need to change some stuff in **Configuration/config_cfi_2016.py**. At L23 (the dict `basedirSignalModelsDict`), for each SMS model I needed to add its name as the dictionary key, with path to the trees as its value. The variable `basedir = "/vols/cms/RA1/80X/"`, so I can set the key-value pair for, e.g., T1bbbb as

```
"T1bbbb":basedir+"MC/20170510_SMS-Spring16_DM-Summer16/
    AtLogicNoSelection_MCMiniAODv2_SUSY_SMS_FastSim/
    SMS_splitMasses/",
```

Because all of the split trees are in the same directory, the dictionary value for each model is the same. After that was done, going to **Analyzers/StatsInput/**, the file **StatsAnalyzer_cfg.py** is the config that makes histograms of the yields under all systematic variations (btag SFs, JECs, etc.). It is these variations that are taken as the systematic uncertainties. I shouldn't need to mess with the config, so I could just type

```
python StatsAnalyzer_cfg.py --outDir <output directory> --
    pset signalmodel --signalModel SMS_<name of model> --
    nCores <number of cores to use> --dataMC mc
```

As some of these models can take ages to run over, I can comment out L232 and L266 in **Core/Process/makeProcess.py** so the control regions aren't run over, speeding everything up.

After running over all of the processes, I may get several output root files. The file of interest is **Signal_SignalModels.root**, which contains histograms of the yields (nominal and with systematic variations) for each process. I need to make sure I run over *all* the samples for a given model so they are all in the same directory. This is important for the efficiencies. Although, sometimes the outputs don't combine properly (or take ages to do so), so I'll get individual root files for each sample in each category but they didn't combine into the <blah>_SignalModels.root. To fix this, I could just source **hadd.sh** in the base output directory and comment out the files that have already combined successfully. Or I can do it via batch in **StatsInput/** with

```
python haddSignalModel.py -i <output directory from
    StatsAnalyzer> -f
```

And I should make sure all the output is on /vols so I don't clog up my allocated space in my user's directory. This is now stored in **/vols/cms/RA1/80X/MC/20170510_SMS-Spring16_DM-Summer16/AtLogicNoSelection_MCMiniAODv2_SUSY_SMS_FastSim/SMS_StatsAnalyzer/**.

23.4 Determining the efficiencies for the SMS models (Spring16 datasets)

The efficiencies for the SMS models are defined as the signal acceptance $\times$ efficiency ($\mathcal{A}\varepsilon$), as shown in Table 5 of [45]. In the auxiliary material for the paper, Figure 2 right shows a plane for the T1qqqq model with different colours representing the efficiency. So the mass of the parent SUSY particle is displayed against the LSP mass, and the colour gradient shows the efficiency for the mass parameter space. I will need to reproduce this type of plot for each of the Spring16 SMS models.

There's a framework called AlphaStats that formats datacards and runs statistical results for SUSY and DM models. I cloned the repo (https://github.com/CMSRA1/AlphaStats) on lxoo in **~/CMScmg_imperial/** with the commands

```
git clone -o alphastats git@github.com:CMSRA1/AlphaStats.git
    --recursive
cd AlphaStats
source setup.sh
git pull alphastats v1.6.x
git submodule update
git submodule init
```

To initialise it I just need to source the setup script, which also creates the alias directory **$ALPHASTATS=/home/hep/ebhal/CMScmg_imperial/AlphaStats**. I'm using the branch *v1.6.x-eshDev20170601*, based off v1.6.x. The output of StatsAnalyzer from AlphaTools is needed to get the efficiencies. If I edit **configStat.py**, I need to change `inputDirSignal` to the path of the output from StatsAnalyzer (i.e., the folder that contains all of the systematics for a given model), and make sure the model(s) are specified in the `signalModels` list (i.e, *only* the model(s) I want to run over for this run). Also, make sure I have `dmModels = [ ]`, everything but `"Signal"` in the `inputArguments.signalFiles[signalModel]` dict commented out (as I only have the signal component from StatsAnalyzer), and the only `sigHists` that's uncommented is `inputArguments.sigHists = "*"`. Running, via batch,

```
python batchSubmitOptimise.py -o <output directory> -f --
   options "-d --getDataLumi --greenBand --signalMCStat" --
   submit
```

should give me Higgs datacards and shape root files for each sample, separated by H_T, b-categories, and jet categories. I then need to combine them using the script **makeCardsAndWs.py**. There's no need to edit anything so I can just type

```
python makeCardsAndWs.py -i <output directory from
   optimiseBinning> -c "nB"
```

making sure the `-c "nB"` flag is used to combine the H_T bins and b-categories, but not the jet categories. Next, I call **runCombineTask.py** to sort the categories by sensitivity with the command

```
python runCombineTask.py -i <output directory from
   optimiseBinning> -t ASCLS_UL_PRIOR --what expected
```

Once that has run, I need to sort the n_{jet} categories by sensitivity. This is done with

```
python sortCategories.py -i <output directory from
   optimiseBinning> -m ul -c "nB"
```

This should give me a file for each sample called **datacards_for_combination_<sample>_mht_sorted.txt**, which contains a list of the combined Higgs datacard files, sorted by sensitivity in the n_{jet} dimension. These files are then used as input for a version of the script **makeJetRankingPlot.py**, which I then modified and stripped down to include only the signal stuff, and reflect the changes between the 2015 and 2016 analyses (such as updating the n_{jet} categories). This script, **makeJetRankingPlot_2016_SMS.py**. makes efficiency plots for the SMS models (and can be built upon to include DM models). I just need to set the path for `inputFileStatsAnalyzer` to the **Signal_SignalModels.root** file in the output from StatsAnalyzer. Then I can just type

```
python makeJetRankingPlot_2016_SMS.py -i <output directory
   from optimiseBinning> -o <output directory for efficiency
    plots>
```

A link to the file is here: makeJetRankingPlot_2016_SMS.py (v1). I made a pull request so the script is now part of the 1.6.x branch in AlphaStats. Shane and the other guys finished the efficiencies and systematics whilst I was away at Glastonbury, and the plots are now in the paper.

23.5 Getting the StatsInput for the DM models (Summer16 datasets)

Finding the systematics for the HF DM models were very similar to the procedure for the SMS models. I needed to add the locations of the trees to `basedirSignalModelsDict` in **Configuration/config_cfi_2016.py**. The keys for the models were "DMttP" and "DMttS", in accordance with the definitions in L159 of **Core/Process/makeProcess.py**. I could then run StatsAnalyzer with the command

```
python StatsAnalyzer_cfg.py --outDir <output directory> --
   pset signalmodel --signalModel <key from config_cfi_2016,
    no other prefix/suffix> --nCores <number of cores> --
   dataMC mc
```

I did get an error when trying this. In **StatsAnalyzer_cfg.py**, the systematics are defined in a list called `systematics` at L39. At L58, if the model is DM, some of the systematics are supposed to be removed. However, they were already commented out in the `systematics` list, so I could just comment out the removal commands at L59-60. Once that was fixed, I could run and then hadd the output without any problems.

23.6 Determining the efficiencies for the DM models (Summer16 datasets)

As with the systematics, running making the efficiency plots was quite similar to the procedure for the SMS models. In **configStat.py**, I had to include the paths to the StatsAnalyzer output for the DM models (this time assigning them to `inputDirDM` instead of `inputDirSignal`), make sure I had the correct models in the list `dmModels`, and that `inputArguments.sigHists = "*"`. One extra edit was changing the dictionary `inputArguments.signalFiles[dmModel]`. One of the key-value pairs pointed to "Signal_MC.root", which is the StatsAnalyzer output for SM MC, so I just had to change it to "Signal_SignalModels.root" and uncomment the other lines in the dictionary.

After those edits, I could just follow the procedure like normal: optimiseBinning → makeCardsAndWs → runCombineTask → sortCategories → makeJetRankingPlot_2016.

The only changes were that I needed the DM cross sections for each sample, which Shane provided as a Python dict in a script (**xsectionDM.py**). So I just had to import that in makeJetRankingPlot_2016 and make sure that the variables read it in correctly, in the format required to implement it properly. I made sure to add the DM support properly, and as such renamed it to **makeJetRankingPlot_2016_SMS_DM.py** .The new version of the script is here: makeJetRankingPlot_2016_SMS_DM.py (v2).

Chapter 24

Making DMSimp dark matter trees (16/05/2017)

I've been instructed to make DMSimp dark matter trees. These are for scalar, pseudoscalar, vector, and axial vector models. I've been sent Python scripts with the lists of the samples in each model. From these, I need to convert them into a format suitable for CMGTools to read and then make the trees. The original models are defined at

- Axial vector: https://github.com/rgerosa/AnalysisCode/blob/Raffaele_8024_X/MonoXAnalysis/test/makeAnalysisTrees/crab/SampleList_MC_80X_Summer16/SampleList_MC_Axial_DMSimp.py
- Vector: https://github.com/rgerosa/AnalysisCode/blob/Raffaele_8024_X/MonoXAnalysis/test/makeAnalysisTrees/crab/SampleList_MC_80X_Summer16/SampleList_MC_Vector_DMSimp.py
- Pseudoscalar: https://github.com/rgerosa/AnalysisCode/blob/Raffaele_8024_X/MonoXAnalysis/test/makeAnalysisTrees/crab/SampleList_MC_80X_Summer16/SampleList_MC_PseudoScalar_DMSimp.py
- Scalar: https://github.com/rgerosa/AnalysisCode/blob/Raffaele_8024_X/MonoXAnalysis/test/makeAnalysisTrees/crab/SampleList_MC_80X_Summer16/SampleList_MC_Scalar_DMSimp.py

These are from the Summer16 batch, as can be inferred from the paths above. The number assigned to `crossSection` is the one computed through the "genAnalyzer" on the sample itself.

Using CMSSW_8_0_25, I went into **CMGTools/**, then created the branch *80X-ra1-0.7.x-Moriond17ProdEshDevDMSimpProd20170516* to store my edits. Moving to **RA1/cfg/**, I created a config file called **run_AtLogic_MCSummer16_DMSimp_cfg.py**. This config uses **run_AtLogic_MCSummer16_DM_cfg.py** as a template, but is tailored to the samples I received for the models above. In **RA1/python/components/** I made a file to store the samples: **components_MC_DMSimp_Summer16.py**.

The config and component files are here: run_AtLogic_MCSummer16_DMSimp_cfg.py (v1); components_MC_DMSimp_Summer16.py (v1).

I've been getting problems when testing. The error message being "Unknown string id in LHE weights". When opening the script that gives the message (**$CMSSW_BASE/python/PhysicsTools/Heppy/analyzers/gen/LHEWeightAnalyzer.py**), there's a comment stating that only certain formats pertaining to the weight are allowed. In the config, I've tried removing the LHE weight instances but the same error pops up. So I copied one of the root files used when testing – by going to the DAS, typing in the dataset, finding the individual root files and then using the `xrdcp` command within CMSSW to copy it – to see if I could find the format used to define the weights.

24.1 Making private dark Higgs trees (08/08/2017)

In addition to the DMSimp trees, I've been asked to run over some privately-produced dark Higgs miniAODs made by Sam Baxter at DESY. The first step is to make the flat trees with Heppy/CMGTools and then run over those within the RA1 framework. I can use a similar procedure to previous tree productions: make a config file that DICE can use to produce the trees, and make a components file that stores the miniAOD information. I used the same branch as the DMSimp production. The config file I made is **run_AtLogic_MC_DarkHiggs_SamBaxter_cfg.py** and the components file is **components_MC_DarkHiggs.py**. The issue is that the samples are private, so can't be run over in the same way as public samples. Shane has a function `makeMyPrivateMCComponentFromIC` which can run over private samples produced at Imperial by using an xrootd "mapping" that's site-specific. As the samples are stored at DESY, I either need the mapping for DESY, or request that a copy of the samples be stored at Bristol/Imperial so that I can run over them. After talking to Sam, I've found that, for DESY, it's `root://dcache-cms-rootd.desy.de/pnfs/desy.de/cms/tier2/`.

Chapter 25

LTA at CERN (18/05/2017)

One of the parks of my PhD is a long term posting (LTA) at CERN. Usually, students go for at least a year, and we have funding for a placement of up to 18 months. I'll probably apply for 12 months, and then possibly extend it if I want to stay (as long as I give the UK Liaison Office, UKLO, at least 3 months notice). Being funded by the STFC, I can only start my LTA on specific dates (listed here: https://www.ppd.stfc.ac.uk/Pages/For-Students-Only.aspx). The 10th October is the most ideal one, so I'll have to find accommodation in Bristol until the end of September/beginning of October. Once Bjoern gives the go-ahead, I need to complete the form on the website requesting accommodation (which is all paid for by the STFC). I should also let Briony Spraggon know so she can notify the STFC of the dates for insurance purposes, and to start preparing the finances.

On the accommodation application form, I've written the start date as 10th October 2017 and the end date as 21st September 2018 (the latest date available on the web page). I have emailed Briony with the dates so she can sort out finances, and Martisse has also been included so that she can put me in contact with the removal's company (Luker Brothers).

My address is

```
63 Rue du Commandant Blaison
Bat 3, ground floor
01630 St Genis Pouilly
```

France

Chapter 26

Using HEPData to archive published material

(14/06/2017)

A recent requirement for CMS analyses is for plots and tables from published papers to be uploaded to HEPData, https://hepdata.net/. The main purpose of this is for archiving and long-term data preservation. The UI on the website stores all data in a table, which can then be visualised on a plot. This means that actual tables from papers can be visualised, and people can see the numerical values associated with plots. (It's a little confusing because on HEPData, all data is regarded as a table, whether it comes from a table or figure in the associated paper.) In this format, tables and plots are described by individual YAML files, and are then collated along with captions and relevant metadata in a **submission.yaml** file.

It was left to me and Ben to convert all the tables and plots from the SUS-15-005 paper into YAML files and then upload it to HEPData. An entry had been created for the paper: https://www.hepdata.net/record/77606. An overview of the submission process could be found here, along with some examples: https://github.com/HEPData/hepdata-submission.

For RA1, we use GitHub to manage the source code of our papers. The repo is AlphaTDR2. So the first step was cloning it, and the branch we used to complete this, to a directory on Soolin (I just used **/storage/eb16003/CMScmg/**). The branch was called "AN-15-005_hepData", and we developed the code in **AlphaTDR2/papers/SUS-15-005/trunk/HepData**. As the examples weren't super helpful so we found some entries for similar papers on HEPData and could download the YAML files to see how the code translated to the final tables. Another helpful piece of documentation was Jae_HepData_SUSYWorkshop_11Apr2017. When we wrote stuff, we could test it using

the instructions at https://github.com/HEPData/hepdata-validator. Instead of using the command prompt every time, I just used those commands to make scripts that test the submission file and each data file, respectively. Ben and I have made a wiki page on GitHub for instructions: https://github.com/CMSRA1/RA1OPS/wiki/HepData-Submission.

Some important syntax subtleties to note:

- Always include a space after a colon operator, e.g., `name: '<blah>'`
- Always use single quotes to enclose a string, never double quotes
- If the number of entries in each column of a table don't match, it won't be flagged as an error in the validator tool. But when uploading the submission to the sandbox, the screen will just say "Loading Data" forever.

There's a "sandbox" area (https://www.hepdata.net/record/sandbox) where anyone can upload and preview files, which is useful for testing and seeing whether everything is laid out properly before the final submission.

For some of the figures, where there are several thousand values, the public page for SUS-15-005 linked a root file that contained these. As the values would be a pain to extract directly from the root file, I tried using the "root_numpy" Python package to port the TTree to a NumPy array. But because the values weren't stored in a tree – they were in a TH2D within a TDirectory – it made things difficult. So Tai sent me a script, which I then modified, that could pull the values out of a TH2D. I used it for the covariance matrix (Additional Figure 22). Then I could use the following formula to calculate the values for the correlation matrix (Additional Figure 23):

$$\mathrm{corr}_{ij} = \frac{\mathrm{cov}_{ij}}{\sqrt{\mathrm{cov}_{ii} \times \mathrm{cov}_{jj}}} \tag{26.1}$$

The script (for future reference) is here:

```
import os, sys, re, logging
import ROOT

# Convert a TH2D's contents into an ntuple, whose info is
    written to a file

rFile = ROOT.TFile('aggregated_results_crfit.root')
tDirectory = rFile.Get('shapes_fit_b')
hist = tDirectory.Get('total_covar')
```

```
ret = [ ]

outFile = open('total_covar.txt', 'w')

xaxis = hist.GetXaxis()
yaxis = hist.GetYaxis()
for j in range(yaxis.GetNbins() + 1):
    for i in range(xaxis.GetNbins() + 1):
        if hist.GetBinContent(i, j) == 0: continue

        # https://github.com/CMSRA1/AlphaTools/blob/v1.10.x/analysis/Analyzers/StatsInput/Modules/StatsInputAnalyzer.py#L55
        row =  (
            yaxis.GetBinLowEdge(j),
            xaxis.GetBinLowEdge(i),
            float("%.5f" % hist.GetBinContent(i, j) ),
        )
        ret.append(row)

outFile.write( str(ret) )
outFile.close()
```

Listing 26.1: A script to extract values from a 2D ROOT histogram (TH2D) and output the values into a text file. File name: convert_TH2D.py.

After Ben and I iterated several times, the submission was finally approved. It can be accessed at https://www.hepdata.net/record/77606.

Chapter 27

Checking $H_{\mathrm{T}}^{\mathrm{miss}}$ tails for data in the event display

(11/07/2017)

For pre-approval for SUS-16-038, I've been asked to look at the $H_{\mathrm{T}}^{\mathrm{miss}}$ tails in the event display at high H_{T}, LT, for data. I can use Fireworks (see 12) as the event display visualiser. It's a case of identifying the excess events and inspecting them to see if there are any problems (reconstruction, etc.). By looking at the mountain range plot for the fully-unblinded 2016 data (the most current one is below), the bins containing "excess" events are indicated as those with large pulls (on the Pull axis, the units are in sigmas). So, to start with, I should be looking in the bins with a pull of 3σ or more: $n_{\mathrm{jet}} = 3$, $n_{\mathrm{b-jet}} = 3$, $H_{\mathrm{T}} > 600$ GeV; and $n_{\mathrm{jet}} \geq 6$, $n_{\mathrm{b-jet}} \geq 3$, $600 < H_{\mathrm{T}} < 900$ GeV.

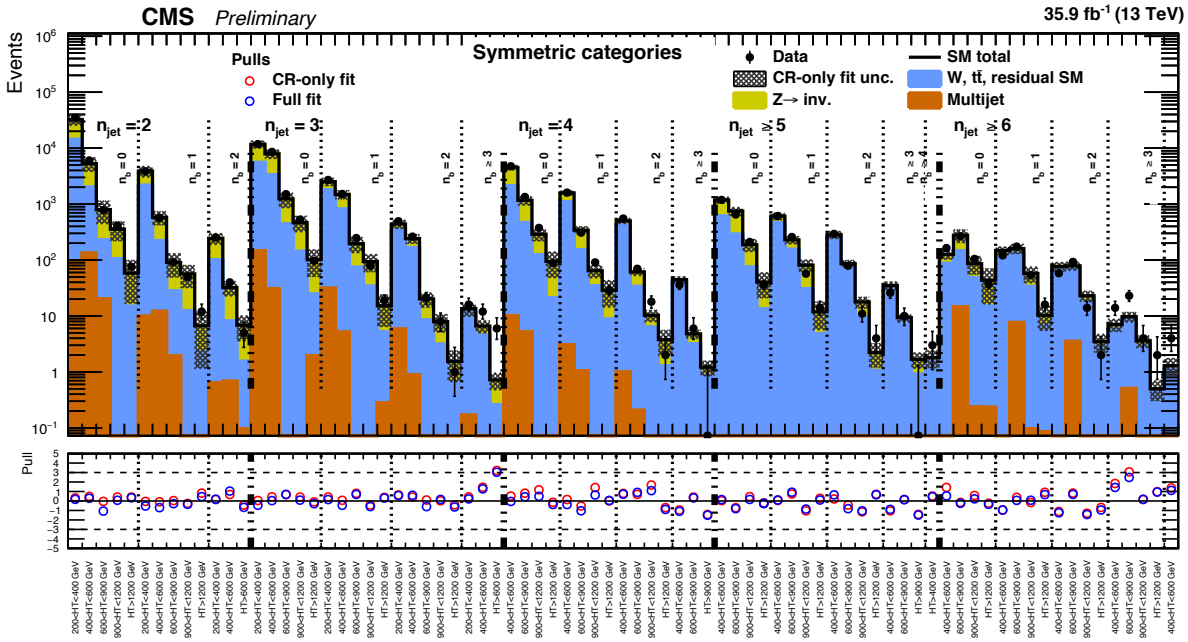

Figure 27.1: Mountain range plot showing the signal region events binned by (symmetric) n_{jet}, $n_{\text{b-jet}}$ and H_T with the fully unblinded 2016 data at 35.9 fb^{-1}.

The above plot is taken from http://www.hep.ph.ic.ac.uk/~klo2/RA1/BTagFormula/FullFit/Log/20170630/Unblinding36fb2017_v14_formulaSyst/postFitNorm/summaryPlot_Symmetric_prefit_overlay_fit_b.pdf. I can apply cuts to select these bins and print out the run, lumi and event numbers. I can then use Lucien's script (https://github.com/lucien1011/LittleFWLite/blob/master/SkimEDM/skimMiniAOD_EvtListAndPD_cfg.py) to skim over the miniAOD, consolidate those events in a new root file, and then analyse them with Fireworks.

If I clone the repository (https://github.com/lucien1011/LittleFWLite, master branch) I can skim over the miniAOD, provided I supply the right input. The script **Utils/TextFileHandler.py** handles that. I need to add the run, lumi and event numbers (in that order) for each event to a text file, with one event per row. The config details the location of this text file. Because of the nature of the code, I need to clone at Imperial. I've forked it so I can do my own development and not mess up the original, so the command to clone it is

```
git clone git@github.com:eshwen/LittleFWLite.git
```

As I'm using my own fork of the repo, I can push any changes to `origin master`.

27.1 Retrieving the event information

To get the event information, I can use AlphaTools at Imperial. On the v1.10.x branch, in **Analyzers/EventDisplay/**, there's a script called **Displayer_HighNb_cfg.py** which can create an RA1-style event display and a list of the events, after cuts, with their relevant information. I just needed to change the cuts to suit my needs, as well as the output file names and directory. It is already configured to run over 2016 data. I also had to edit L129 and L176 of **./Modules/Displayer.py**, changing `objTag = ""`. The leaf "pseudoJetFlag" is not in the newer trees, so I get errors if I leave those references in. Once I had made those changes, I could just run the script. The output folder contains text and pdf files for each data set that was run over. The text files contain the run number, lumi section and event number for each event that I need for the next step. The pdfs contain the RA1-style displays – with one pdf per data set – that show the event information, α_T distributions, the properties and tracks of the hardest objects, and some calorimeter information.

The RA1 displays are quite helpful. Fireworks doesn't distinguish between regular jets and b-jets (and we need to know which is which). So I added some code in **Displayer.py** to change the colour of the b-jets in the circular plot and the square $\eta - \phi$ plot, as well as add a legend entry for them. But it turns out that the b-tag working points we were using were out of date. They are stored in a dict in **$ALPHATOOLSDIR/Data/BTagWorkingPoints.py**, and the `outString` string in the `printJets` function of **Displayer.py** calls the values in this dict. The portion of the string `bCategory("CSV",jet.btagCSV)` appends L, M or T (whether it uses the loose, medium or tight working point) to the bit in quotes, and then pulls the corresponding value from the dict. So I just had to change it to `bCategory("CSVv2IVF",jet.btagCSV)`. Then I could add `if` statements in the jet loops of **Displayer.py** to draw the b-jets that satisfied the medium working point (`phyobj.btagCSV > 0.8484`) in dark red.

27.2 Skimming the miniAODs

Now I can use LittleFWLite to skim over the miniAODs in the data set, and output the selected events into a new root file. If I go to the directory (**~/LittleFWLite/**), I need to `source setup.sh` and also do `cmsenv` in a recent CMSSW release – I used 8.0.26 – to source the main FWLite and FWCore modules.

If I go to the CMSSW GitHub repo (https://github.com/cms-sw/cmssw), there are lots of modules in different directories. If I follow the Flat Tree Production instructions, there's a line with `cat » .git/info/sparse-checkout «EOF`, and different modules below. I could add FWCore in that list of modules, then escape the list by typing `EOF`. Then I can continue with the instructions and compile, etc., so the modules I listed are included in the CMSSW directory I'm using. This is handy if I ever need specific CMSSW modules that aren't part of the standard workflow.

In the script **SkimEDM/skimMiniAOD_EvtListAndPD_cfg.py**, I needed to make sure I passed `runAAA=True` to the `getFilesFromPD` function in the **Utils/DBHandler.py** script. To run over every miniAOD file in the data set, I could leave the `True` alone in `options.register('allFiles'`. However, this can take ages to run and isn't worth it unless I want to process a lot of events. So it's easier to just change it to `False`. This way, only files that contain the event information I supply are run over. Then I had to make sure the data set name , text file path, and output file path were correct. The syntax for the primary data set is quite specific. As the code uses the DAS to search for the data set and its files, it needs to be entered in the same format as a DAS query. I should look up the data set name on DAS by typing the process and run era, then using *s for the rest of the query. I could then run the config with

```
cmsRun skimMiniAOD_EvtListAndPD_cfg.py
```

I should get the log on-screen indicating the root files being opened and the events being processed.

27.3 Using Fireworks to check the event displays

Once I had the ROOT file of the events I needed, I could use Fireworks to look at the event displays. For stand-alone tarballs, the commands to download and install it are different for the different versions (see 12). However, it is fully integrated within CMSSW from CMSSW_3_X_X. As I was using CMSSW_8_0_26, I just needed to `cmsenv`, then I could run it with

```
cmsShow <path to root file>
```

If on a remote server, I need to connect with `ssh -X` or `-Y` for X11 forwarding. I initially got some errors when trying to load the program, but entering the command `defaults write org.macosforge.xquartz.X11 enable_iglx -bool true` into the terminal, locally, then restarting my Mac fixed the problem.

I can switch between events in a ROOT file by pressing the big green arrows. I can add annotations to physics objects – to show their properties – by right-clicking over the object and pressing "Add Annotation". I can also drag the annotation to move it, and a grey line connects it to its object to avoid

confusion with other, possibly similar objects. I can also save files, the main display being labelled a $\rho - \phi$ plot with File→Export Main View Image.

I couldn't find any obvious anomalies in the event displays. There were several jets, as expexted, high $E_{\mathrm{T}}^{\mathrm{miss}}$, and the occasional high-p_{T} muon. One of the annotated displays is below.

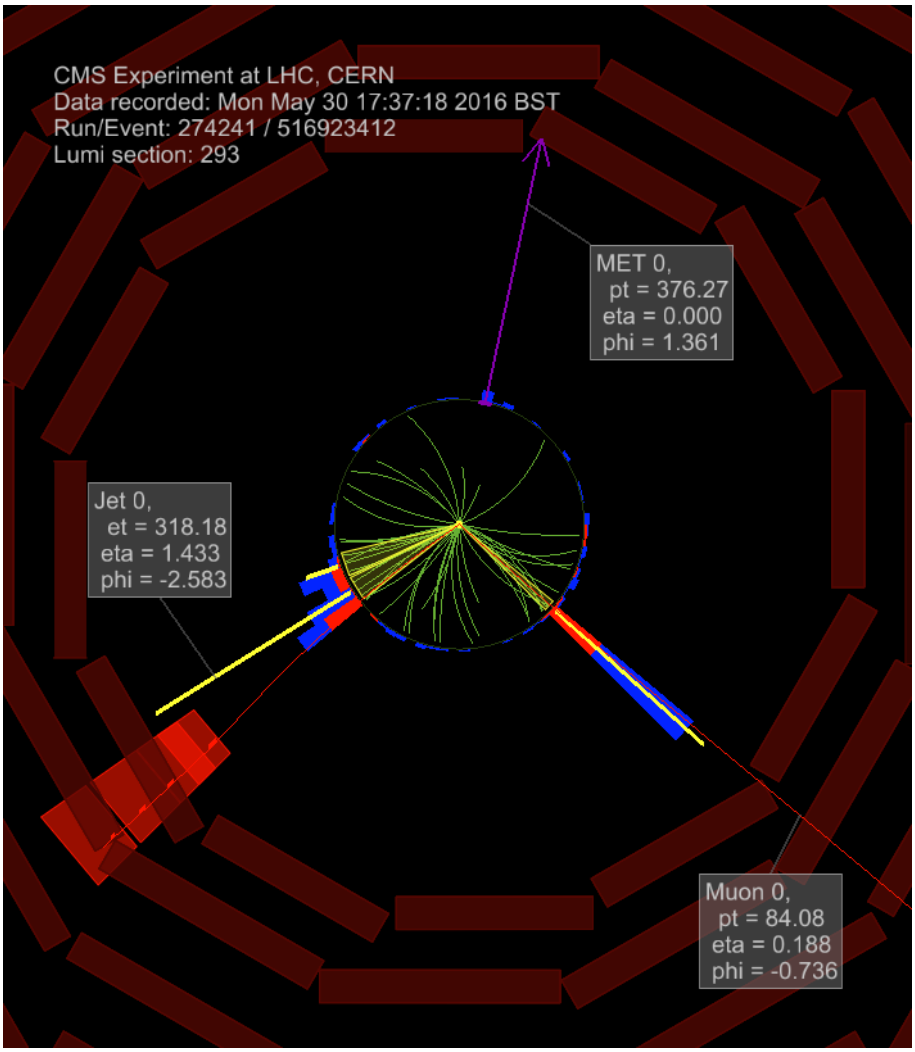

Figure 27.2: The event display from an "excess" event in the /HTMHT/Run2016B-03Feb2017_ver2-v2/MINIAOD primary data set. The leading objects are annotated, as well as the event information.

27.4 Conveying the results on Imperial's web pages

Using some software developed by Ben and other IC people, I can use HTML and web-based languages to upload plots, etc., in a clean layout on Imperial's website. By logging onto lxoo or lxo1, I need to do

```
mkdir ~/public_html # Folder name must be this
mkdir -p CMS/RA1/AN-17-122/EventDisplays
cd CMS/RA1/AN-17-122/EventDisplays
git clone https://github.com/ic-coders-club/plotify.git
```

Then I will have a web page that I can view on a browser, of the form **www.hep.ph.ic.ac.uk/~<user>/<path below home dir minus "public_html/">**. From the above commands, it is www.hep.ph.ic.ac.uk/~ebhal/CMS/RA1/AN-17-122/EventDisplays. After this, I copied the Fireworks and RA1 event displays to **plotify/plots/**. The pdfs needed to be converted to png. To do this, I can just type

```
for file in */*pdf; do convert -density 150 $file -trim ${
    file/pdf/png}; done
```

Next I had to add a **config.js** file, which contained code to point to the images to display, as well as how to organise and divide them up. Then I added an **index.html** file, which took care of the layout of the web page and the sub-headings, etc. The **style.css** file generated the colours and font. The link to all the event displays is at http://www.hep.ph.ic.ac.uk/~ebhal/CMS/RA1/AN-17-122/EventDisplays/plotify/. The Fireworks and RA1 displays are shown, with one event per page. As the Fireworks displays don't distinguish between regular and b-jets, I can match the circular plot in the RA1 display to the jets in Fireworks.

27.5 Remaking the RA1 trees to include all muons in the event displays

By default, in the RA1 display none of the muons aren't plotted; presumably because they fail at least one of the acc/ID/isolation/ΔR requirements. Because the code that produces the plots runs over the trees located at Imperial. The cuts that are applied during tree production means that none of these muons are actually stored in the trees. But with the skimmed miniAODs I have of the excess events, we should be able to reproduce the trees of those events, but leave in all the muons. Then I can run over these new, small trees in AlphaTools to produce RA1 displays that contain the muons and their properties.

Using Heppy/CMGTools in CMSSW, I needed to lower the thresholds for muons in **RA1/python/buildSequence.py**. Then add this snippet to L23 in **./analyzers/AtEventAttributesPrep.py** so all muons are included in the trees:

```
and e.relIso < 0.12 and e.miniRelIso < 0.2
```

This is due to the event selection and tree content, and full content are separate. So CMG-Tools needs to cut on right muons and save loose muons. I then needed to make a new function to skim over data locally in **RootTools/python/samples/ComponentCreator.py**, using the function `makeMCComponentFromLocal` as a template. I had to tweak it to omit MC-specific variables and include data-specific ones. The function looked like this:

```
def makeDataComponentFromLocal(self,name,dataset,path,
    pattern=".*root"):
      component = cfg.DataComponent(
          dataset=dataset,
          name = name,
          files = path+"/"+dataset+".root",
          triggers = [],
          intLumi = 1,
      )
      return component
```

Once that was done, I could make a components file containing the skimmed miniAODs, using **RA1/python/components/components_Data_2016.py** as a template. I had to replace the components with my own and change the line

```
components = [kreator.makeDataComponent(**s) for s in
    componentList]
```

to

```
components = [kreator.makeDataComponentFromLocal(**s) for s
    in componentList]
```

To match the input required for the function above, my components looked like this:

```
HTMHT_Run2016H_6j3b = dict(
      name = "HTMHT_Run2016H_6j3b",
      dataset = "skimHTMHT_Run2016H_6j_3b",
      path = "/home/hep/ebhal/EventDisplay_MHT_6j_3b/\
    ROOT_files",
)
```

Finally, I edited the config file **RA1/cfg/run_AtLogic_Data_cfg.py** to include my component file and my function to make the data components from local files. One problem I ran in to was that the config described my components as "data" (instead of MC). Normally, this requires a JSON

associated with the data set. But if I just commented out `jsonAna` in the list `susyCoreSequence` in **TTHAnalysis/python/analyzers/susyCore_modules_cff.py**, I could bypass the JSON stuff as it's not needed in this case. Finally, I could run the config with

```
heppy TEST_Skim run_AtLogic_Data_cfg.py
```

But when I looked at the trees, few muons were present. It seems some were still failing the selection/ID requirements. If I edited **RA1/python/treeContent/baseContent.py** and added the line

"inclusiveMuons" : NTupleCollection("allMuon", slimLeptonType, 50, help="All muons passing very loose selection conditions"),

around L223, some of these failing muons would be included in the trees. The rest can be added if I commented out L179-181 of **PhysicsTools/Heppy/python/analyzers/objects/LeptonAnalyzer.py**, which removes the selection criteria on these inclusive muons. I only need to leave the statement `mu.track().isNonnull()`. The leaf prefix would be "allMuon". So I could see their pT, eta, phi, etc. in the leaves "allMuon_pt", and so on.

Once the trees were made, I had to ensure that AlphaTools ran over them. It's not as simple a process as I'd like. Firstly, I had to add the trees to **$ALPHATOOLSDIR/Configuration/Samples/samples_13TeV_80X.py**. I could add each tree to the list that contained every sample, the syntax being

```
sampleList80X.addSample("HTMHT_Run2016B_3j3b", isData=True,
    parent="HTMHT")
```

Then I needed to add those samples to a "collection", like

```
sampleList80X.addCollection("EshsSkimmedExcessSamples",
                            ["<sample>",
        ])
```

I also needed to change the variable `basedirNominal` in **Configuration/config_cfi_2016_skim.py** to point to the location of my trees. To include the "allMuon" objects (as they are different to regular muons), I had to add an entry for them in **Producers/Config/MakePhysObjects_Setting.py** in the same format as the others. I also needed to add the `event.incMuon` entry to **Producers/MakePhysObjects.py** and add equivalent code to the event display scripts (**DefaultDrawSetting.py** and **Displayer.py**). Finally, I just needed to make sure my collection was referenced in **Analyzers/EventDisplay/Displayer_HighNb_cfg.py**. After looking at them, Rob concluded that we looked into the events in enough detail and that there were no obvious problems. The reply he gave to the conveners was "We have studied 29 events in the bins with the largest excesses (eq3j,eq3b) and (ge6j,eq3b) and find no anomalous behaviour."

Chapter 28

Using NumPy and root_numpy (19/07/2017)

NumPy (http://www.numpy.org/) is a Python package that can handle scientific computing. It's installed by default with pip and recent (70X and later) CMSSW releases. It's part of the SciPy ecosystem that also includes pandas, matplotlib and IPython. There's another package called root_numpy that, among other things, can

– convert ROOT TTrees into NumPy arrays, and vice versa

– make use of matplotlib for plotting

– make use of pandas and its dataframes for data analysis (like AlphaTwirl)

– make use of scikit-learn for machine learning

root_numpy is installed by default only in very recent CMSSW releases (92X and later), but can be installed with pip as well. A good primer for root_numpy is http://scikit-hep.org/root_numpy/index.html.

Chapter 29

Tree production for common analysis and 2017 data (19/09/2017)

In recent months, we have decided to start implementing a "common analysis" with the VBF (and possibly Top) group in CMS. The first step is to produce common flat trees that includes all the information each group needs. This will save processing time (as only one set of trees needs to be made to satisfy all the groups) and storage space. Currently, we are still using Heppy and CMGTools with the RA1 workflow, but accommodating for the selections, variables and triggers, etc. that VBF requires. There has been some debate on the triggers and selections (and their thresholds) because there's a trade-off between inclusivity and size of the trees. If we make minimal selections, there's more potential for different studies and analyses but the trees will be larger. Our code in cmgtools-lite-private also had to be re-based to work with CMSSW_92X to process 2017 data, which Shane has done (albeit with a few bugs).

We will be using a common development branch (VBF_test) to test everything. My first job is to modify the "hadd" functionality in Heppy (**heppy_hadd.py**). Once the trees have been produced with a batch system, rather than just hadding the trees like we normally do and using up more storage, we should make a root file that consists of TChains that point to the batch output files with xrootd. This way, the output trees only need to be stored at one site, and everyone else can access and analyse them remotely with xrootd. In the near future, we're planning to use CRAB for job submission rather than the batch systems at Bristol or Imperial. This has several benefits. But when the time comes, the hadding functionality will need to be modified further as we need to access other non-ROOT files.

First off, I need to source CMSSW, CMGTools and Heppy releases that are up to date. At, e.g., Imperial, I should initialise my grid certificate, then follow the instructions at https://github.com/CMSRA1/RA1OPS/wiki/2a.-UK-Common-Analysis-Flat-Tree-Production-(92X)-[WIP]. The main take away is that I need CMSSW_9_2_4, to check out the branch "heppy_92X_ra1-0.8.x" in **src/** and to check out "VBF_test" after cloning the new CMGTools repo. The full list of commands are

```
cd ~/CMScmg_imperial
cmsrel CMSSW_9_2_4
cd CMSSW_9_2_4/src
cmsenv
export CMSSW_GIT_REFERENCE=/vols/cms/RA1/cmg-cmssw-bare
git cms-init
git remote add ra1-private git@github.com:CMSRA1/cmg-cmssw-private.git
git fetch ra1-private
cat >> .git/info/sparse-checkout <<EOF
.gitignore
PhysicsTools/Heppy
PhysicsTools/HeppyCore
EgammaAnalysis/ElectronTools
RecoEgamma/ElectronIdentification
RecoEgamma/PhotonIdentification
EOF
git config core.sparsecheckout true
git checkout heppy_92X_ra1-0.8.x
echo CMGTools/ >> .git/info/exclude
git clone https://github.com/professor-calculus/cmgtools-lite-private.git CMGTools
cd CMGTools
ls
git checkout VBF_test
git submodule init
git submodule update
```

I could then look at **heppy_hadd.py** in **PhysicsTools/HeppyCore/scripts/**. As this hadding stuff doesn't need to be updated to comply with 92X trees, I can test my changes on 80X trees rather

than waste time making 92X trees to use for testing.

Update: instead of `heppy_hadd`, we're likely going to switch to using the nanoAOD data format and not use Heppy/CMGTools to make trees.

29.1 nanoAOD

Chapter 30

Cut flow tables for SUS-16-038 (20/09/2017)

As with SUS-15-005, I've been tasked with making the cut flow tables for the SUSY benchmark models, this time to add to the Analysis Note for SUS-16-038 (AN-17-122). The cut flow is

```
# Mass point cuts
dict(All = ("ev : ev.nElectronsVeto[0] == 0",
                "ev : ev.nMuonsVeto[0] == 0",
                )), # Lepton vetoes
    "ev : ev.nIsoTracksVeto[0] <= 0",
    "ev : ev.nPhotonsVeto[0] == 0",
    "ev : ev.jetNewID[0] != 0", # Odd jet veto
    "ev : ev.nJet40[0] >= 2",
    "ev : 0.1 <= ev.jet_chHEF[0] <= 0.95",
    "ev : ev.jet_pt[0] > 100",
    "ev : ev.ht40[0] > 200",
    "ev : ev.mht40_pt[0] > 200",
    "ev : ev.nJet40Fwd[0] == 0",
    "ev : ev.MhtOverMet[0] < 1.25",
    dict(Any = (dict(All = ('htbin_200', ('alphaT', dict(v =
    0.65)))),
```

```
                dict(All = ('htbin_250', ('alphaT', dict(v =
    0.60)))),
17              dict(All = ('htbin_300', ('alphaT', dict(v =
    0.55)))),
                dict(All = ('htbin_350', ('alphaT', dict(v =
    0.53)))),
19              dict(All = ('htbin_400', ('alphaT', dict(v =
    0.52)))),
                dict(All = ('htbin_600', ('alphaT', dict(v =
    0.52)))),
21              dict(All = ('htbin_900',))
                )
23          ), # HT-dependent AlphaT cuts
     "ev : ev.biasedDPhi[0] > 0.5",
25   # Most sensitive simplified njet, nb category
     dict(Any = ("ev : ev.HLT_PFHT800[0] == 1",
27              "ev : ev.HLT_PFHT900[0] == 1",
                "ev : ev.HLT_PFMETNoMu90_PFMHTNoMu90_IDTight
    [0] == 1",
29              "ev : ev.
    HLT_PFMETNoMu120_PFMHTNoMu120_IDTight[0] == 1")), # HLTs,
     needed only for LLP models
```

and the models are listed in the subsections below. The "simplified" categories group several n_{jet}, n_b categories together, e.g., `eq01b_eq45j`. Then, in the cut flow I could cut on `"eq0b or eq1b"` and `"eq4j or eq5j"`. So the cut would look like

```
1 dict(All = ( dict(Any = ("ev : ev.nBJet40[0] == 0", "ev : ev
    .nBJet40[0] == 1")),
               dict(Any = ("ev : ev.nJet100[0] == 4", "ev : ev
    .nJet100[0] == 5")),
3 )),
```

although, the ones we use are usually simpler (e.g., `ge6j_le1b`). The first step is to reproduce the trees of these samples from miniAODs without any cuts applied. Then, I can use Tai's cutflowirl repo as before to make the raw cut flow tables. As the format of the trees is the same as in 2015, I don't need to change much within the code; mainly update the event selections and the path to the trees.

As some of the trees are stored at Imperial, I can easily set up the repo there. I made a directory called **CutFlowCode/**, set up CMSSW_8_0_26 and then cloned my fork of cutflowirl.

30.1 Long-lived particles

The most pressing models that need the cut flow tables – because we're presenting on them – are for the SUSY SMS long-lived particles (LLP). These are particles that can decay far into (or outside of) the detector. Their long-lived-ness is usually expressed as a decay length $c\tau$ (in mm), where τ is the mean lifetime of the particle and the c factor is because they are obviously highly-relativistic. Our LLP samples have $c\tau$ values of anywhere between 10^{-3} to 10^{5} mm. The theoretical motivation for this is that Supersymmetry is "split". In this model, the gluino and LSP are the only sparticles light enough to be accessible at the the LHC energy. The rest of the sparticles are decoupled to a much higher energy scale and so can't be produced in the Collider.

Christian produced the trees for these models and confirmed there are no pre-selections, meaning I can use the cutflowirl code to make the cut flow tables immediately. I switched to a new branch "SUSY_LLP_2016" so the code for the previous tables wasn't overwritten. The samples are

– SMS-T1qqqqLL_ctau_0p001_mGluino-1800_mLSP-200_25ns

– SMS-T1qqqqLL_ctau_0p001_mGluino-1000_mLSP-900_25ns

– SMS-T1qqqqLL_ctau_1_mGluino-1800_mLSP-200_25ns

– SMS-T1qqqqLL_ctau_1_mGluino-1000_mLSP-900_25ns

– SMS-T1qqqqLL_ctau_100000_mGluino-1000_mLSP-200_25ns

– SMS-T1qqqqLL_ctau_100000_mGluino-1000_mLSP-900_25ns

and the cut flow is described above. I could then run the script as normal with

```
./cutflowirl/twirl_mktbl.py -c <sample>
```

30.1.1 Problems encountered and re-weighting the trees

Originally, I tried to run over the split trees as they were already separated by mass point, which would make everything quicker. However, I ran into weird errors when attempting it. The problem was that the root file for each split tree actually contained two identical trees, each called "tree". The one with the higher index is the latest revision of the tree. But as the tree name is hardcoded into a variable in the cutflowirl workflow, it may have been confused when it came across two trees with the same name. So I decided to run over the original, unsplit trees instead, located at **/vols/cms/RA1/80X/MC/LongLived/20170904/AtLogic_MC_LL**.

Another problem I ran into was that the end-of-cut-flow efficiencies for my tables were noticeably lower than in the efficiency maps produced by Christian and Lucien. After some debugging and jigging the α_{T} cuts in the cut flow, many of the events that failed were in the $800 < H_{\mathrm{T}} < 900$ region and weren't being run over properly. It turned out to be some outdated information in Tai's atlogic submodule. In **atlogic/Scribblers/htbin.py**, L7 gives the bin boundaries. I had to update the "800" to 900 because there was no $H_{\mathrm{T}} = 800$ boundary any more. This fixed the low-efficiency issue. I then made a PR to include the change whenever the submodule was used in the future.

But even after all that, the efficiencies were still different. This was because the trees I was running over were not weighted, but the events in the efficiency maps were. So I had to use AlphaTools ("v1.10.x_LLValidation" branch) to re-weight the events but not apply any selections. In **Configuration/config_cfi_2016.py**, I had to set the directory of `basedirMC` to the location of the *split* trees (running over them removes one of the two identical trees so cutflowirl can run over the output). Then I could add the following line to **Configuration/Samples/samples_13TeV_80X_T1qqqqLL.py**:

```
sampleList80X_T1qqqqLL.addCollection("SMS_T1qqqqLL_Esh", [
    "SMS-T1qqqqLL_ctau_0p001*1800*-200*",
    "SMS-T1qqqqLL_ctau_0p001*1000*900*",
    "SMS-T1qqqqLL_ctau_1_*1800*-200*",
    "SMS-T1qqqqLL_ctau_1_*1000*900*",
    "SMS-T1qqqqLL_ctau_100000_*1000*200*",
    "SMS-T1qqqqLL_ctau_100000_*1000*900*"],
    mergeFiles=mergeMasses)
```

In **config_cfi_6CF.py**, I had to set the relevant cuts in the functions `makeParameterSet` and `makeSignalParameterSet` to zero and comment out L74-75 (the `topPtAnalyzer` stuff) in in **Skimmers/TreeSkimmer.py**. The config I used was **SkimTreeProducer/SkimTreeProducer_PSet_cfg.py**. I had to edit the sequence in that file, commenting out all of the sequence modules except the following:

```
process.sequence = producerSequence
process.sequence.extend( weightSequence )
process.sequence.append( treeSkimmer )
process.endSequence = defaultEndSequence
```

and putting them in that order. The final things were replacing the `return False` in L58 of **Producers/WeightTriggerSignalBin2016.py** with `return True` so that module didn't remove events from the re-weighted trees, and commenting out `weightLeptonSFEtaPtCRMuons2016` from **weightSequence** in **Sequences/Sequence2016.py** as it gave weird errors. Then I could run it

with

```
python SkimTreeProducer_PSet_cfg.py --mc --era 2016 --pset=
   Signal --customSamples SMS_T1qqqqLL_Esh --outDir <blah>
```

and use the output trees as input for cutflowirl.

30.1.2 Developing cutflowirl

By default, cutflowirl just counts the number of events that pass/fail selections and doesn't include support for weights. I rectified this by making some new classes in atlogic – that are modified versions of **EventSelectionXXXCount.py** – that pass the event information to a modified version of **Count.py**. Then the variables that keep track of number of successful and total events could just be incremented by the value in the weight branch of the tree, rather than 1, i.e., `r[IDX_TOTAL] += 1` would be replaced by `r[IDX_TOTAL] += event.w[0]` to track the total number of events at each stage of the cut flow. But when running, I got errors that the leaf containing the weights didn't exist. After some debugging, I realised that the leaf type for `w` was `Float_t`, which was not supported in the version of AlphaTwirl contained within cutflowirl. So I had to add the line `Float_t = 'f',` to the dict `typedic` in **AlphaTwirl/AlphaTwirl/Events/BranchAddressManager.py**. I also added a `-w/--weights` parser option to **twirl_mktbl.py**. The default is set to `False`, in which case it doesn't use the weight-storing classes. Now, if I want weighted cut flow tables, I can run with

```
./cutflowirl/twirl_mktbl.py -c <component(s)> -w
```

For the AN, we also wanted the jet ID efficiency vs $c\tau$ for each benchmark model. The trees were remade to include this branch, so I had to reweight those and run over them. To implement support, I had to create a new scribbler: **atlogic/Scribblers/jetNewID.py**. As most of the branches for our trees only store one number per event, using the `[0]` element for each branch was fine. But as there are multiple jets in most events, the branch `jet_newId` is an array with multiple elements for each event. As cutflowirl doesn't have support for this, my scribbler loops over each entry in the `jet_newId` array. If any of the entries equal zero (i.e., the jet failed the ID requirements), the whole event fails the cut so doesn't progress in the cut flow. If all the entries pass, a single-element list called 'jetNewID' is created that has a value of 1. Then I can just include it as a normal lambda string in the cut flow, i.e., `"ev : ev.jetNewID[0] != 0"`. Once written, I added the scribbler to **twirl_mktbl.py** in the same way as the other scribblers.

So my fork of cutflowirl is now able to handle weighted events, which is useful for future cut flow tables and efficiencies. The important modified files are twirl_mktbl.py L38 and 123–132, CountWeight.py L28–31, WeightedEventSelectionAllCount.py L48, WeightedEventSelectionAnyCount.py L48, WeightedEventSelectionNotCount.py L38, jetNewID.py.

30.1.3 LLP cut flow tables

The cut flow tables are listed below.

For $c\tau = 0.001$ mm:

Table 30.1: Cut flow table for T1qqqqLL models with $c\tau = 10^{-3}$ mm.

Event selection	Benchmark model (m_{SUSY}, m_{LSP}, $c\tau$)	
	T1qqqqLL ($1800, 200, 10^{-3}$)	T1qqqqLL ($1000, 900, 10^{-3}$)
Before selection	100	100
Event veto for muons and electrons	99	100
Event veto for single isolated tracks	91	89
Event veto for photons	90	89
Event veto for jets failing ID	90	88
$n_{\mathrm{jet}} \geq 2$	90	76
$0.1 < \mathrm{CHF}^{\mathrm{j}_1} < 0.95$	87	72
$p_{\mathrm{T}}^{\mathrm{j}_1} > 100$ GeV	87	44
$H_{\mathrm{T}} > 200$ GeV	87	39
$H_{\mathrm{T}}^{\mathrm{miss}} > 200$ GeV	82	22
Event veto for forward jets ($\lvert\eta\rvert > 2.4$)	77	20
$H_{\mathrm{T}}^{\mathrm{miss}}/E_{\mathrm{T}}^{\mathrm{miss}} < 1.25$	76	19
H_{T}-dependent α_{T} requirements ($H_{\mathrm{T}} < 900$ GeV)	76	12
$\Delta\phi_{\mathrm{min}}^{*} > 0.5$	24	7.9
Trigger efficiency	24	7.8

For $c\tau = 1$ mm:

Table 30.2: Cut flow table for `T1qqqqLL` models with $c\tau = 1$ mm.

Event selection	Benchmark model (m_{SUSY}, m_{LSP}, $c\tau$)	
	T1qqqqLL (1800, 200, 1)	T1qqqqLL (1000, 900, 1)
Before selection	100	100
Event veto for muons and electrons	99	100
Event veto for single isolated tracks	80	90
Event veto for photons	79	89
Event veto for jets failing ID	79	89
$n_{\mathrm{jet}} \geq 2$	79	70
$0.1 < \mathrm{CHF}^{\mathrm{j1}} < 0.95$	74	66
$p_{\mathrm{T}}^{\mathrm{j1}} > 100\,\mathrm{GeV}$	74	36
$H_{\mathrm{T}} > 200\,\mathrm{GeV}$	74	31
$H_{\mathrm{T}}^{\mathrm{miss}} > 200\,\mathrm{GeV}$	68	15
Event veto for forward jets ($\lvert\eta\rvert > 2.4$)	65	14
$H_{\mathrm{T}}^{\mathrm{miss}}/E_{\mathrm{T}}^{\mathrm{miss}} < 1.25$	58	13
H_{T}-dependent α_{T} requirements ($H_{\mathrm{T}} < 900\,\mathrm{GeV}$)	58	8.7
$\Delta\phi_{\mathrm{min}}^{*} > 0.5$	22	5.8
Trigger efficiency	22	5.8

For $c\tau = 10^5$ mm:

Table 30.3: Cut flow table for T1qqqqLL models with $c\tau = 10^5$ mm.

Event selection	Benchmark model (m_{SUSY}, m_{LSP}, $c\tau$)	
	T1qqqqLL (1000, 200, 10^5)	T1qqqqLL (1000, 900, 10^5)
Before selection	100	100
Event veto for muons and electrons	100	100
Event veto for single isolated tracks	97	97
Event veto for photons	97	97
Event veto for jets failing ID	95	96
$n_{\mathrm{jet}} \geq 2$	35	33
$0.1 < \mathrm{CHF}^{\mathrm{j_1}} < 0.95$	29	30
$p_{\mathrm{T}}^{\mathrm{j_1}} > 100\,\mathrm{GeV}$	23	23
$H_{\mathrm{T}} > 200\,\mathrm{GeV}$	21	21
$H_{\mathrm{T}}^{\mathrm{miss}} > 200\,\mathrm{GeV}$	16	16
Event veto for forward jets ($\lvert\eta\rvert > 2.4$)	15	14
$H_{\mathrm{T}}^{\mathrm{miss}}/E_{\mathrm{T}}^{\mathrm{miss}} < 1.25$	14	14
H_{T}-dependent α_{T} requirements ($H_{\mathrm{T}} < 900\,\mathrm{GeV}$)	9.7	9.4
$\Delta\phi^*_{\mathrm{min}} > 0.5$	8.2	8.2
Trigger efficiency	8.2	8.2

I added these to **136_LLP.tex** in AN-17-122 in the AlphaTDR2 repo and then made a PR to merge it into the master branch.

30.2 HLT studies

In addition to the cut flow tables, I've been doing some studies with the HLT, i.e., finding its efficiency for different LLP models and lifetimes, and looking at the luminosity collected by each trigger so that the acceptance $\times$ efficiency values we quote account for them. The main triggers of interest are the H_{T} triggers: `HLT_PFHT800` (low threshold), `HLT_PFHT900` (high threshold); and monojet triggers: `HLT_PFMETNoMu90_PFMHTNoMu90_IDTight` (low), `HLT_PFMETNoMu120_PFMHTNoMu120_IDTight` (high). In signal selection, an OR (like Tai's `Any` class in Python) of these triggers are taken. Early in the 2016 run `HLT_PFHT900` became the low threshold for the H_{T} trigger. Below is some information about the luminosity collected by the different triggers.

Table 30.4: The luminosity collected by each HLT, or combination of Triggers, from the entire 2016 run. Over the course of the run, the threshold for the lowest unprescaled trigger in each category rose. The fraction of luminosity collected by each Trigger whilst it was lowest threshold is included.

Trigger	Luminosity collected (fb^{-1})	Fraction collected as lowest unprescaled trigger
`HLT_IsoMu22 or HLT_IsoTkMu22`	28.6	0.80
`HLT_IsoMu24 or HLT_IsoTkMu24`	35.9	0.20
`HLT_PFHT800`	27.3	0.76
`HLT_PFHT900`	35.9	0.24
`HLT_PFMETNoMu90_PFMHTNoMu90_IDTight`	13.9	0.39
`HLT_PFMETNoMu100_PFMHTNoMu100_IDTight`	17.6	0.10
`HLT_PFMETNoMu110_PFMHTNoMu110_IDTight`	35.3	0.49
`HLT_PFMETNoMu120_PFMHTNoMu120_IDTight`	35.9	0.02

30.3 Tables for the remaining benchmark models

The remaining SUSY models are

- T1bbbb (1900, 100) and (1300, 1100)
- T1qqqq (1700, 100) and (1000, 850)
- T1tttt (1700, 100) and (950, 600)
- T2bb (1000, 100) and (550, 450)
- T2tt (1000, 50), (450, 200) and (250, 150)
- T2cc (500, 480)
- T2qq_8fold (1250, 100) and (700, 600)
- T2qq_1fold (700, 100) and (400, 300)

These trees needed to be remade without any cuts. I'm working on Soolin with CMSSW_8_0_25, with the CMGTools branch "80X-ra1-0.7.x-Moriond17Prod-EshSUSYcutflow2016" and CMSSW branch "heppy_80X_ra1-0.7.x-Moriond17Prod". In **CMGTools/RA1/cfg/**, I edited run_AtLogic_MCMiniAODv2_SUSY_SMS_FastSim_cfg.py and commented out all the conditions in `buildEventSelection_options`. To store the miniAOD samples, I made a components file components_EshSUSYbenchmarks.py and made sure the config called it.

Once the trees were made, I had to transfer them to Imperial so I could split and then reweight them with AlphaTools. I just needed to follow the instructions for the tree splitting, then update the base directory in the config script to point to the split trees, and add them to the script containing the samples. Then I could use cutflowirl to get the cut flow tables.

Table 30.5: Cut flow table for `T1bbbb` and `T1qqqq` benchmark models.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})			
	T1bbbb (1900, 100)	T1bbbb (1300, 1100)	T1qqqq (1700, 100)	T1qqqq (1000, 850)
Before selection	100	100	100	100
Event veto for muons and electrons	99	98	100	100
Event veto for single isolated tracks	96	93	95	92
Event veto for photons	95	92	94	92
Event veto for jets failing ID	95	92	94	91
$n_{\text{jet}} \geq 2$	95	90	94	88
$0.1 < \text{CHF}^{\text{j}_1} < 0.95$	90	87	89	83
$p_{\text{T}}^{\text{j}_1} > 100\,\text{GeV}$	90	75	89	64
$H_{\text{T}} > 200\,\text{GeV}$	90	73	89	60
$H_{\text{T}}^{\text{miss}} > 200\,\text{GeV}$	84	37	83	27
Event veto for forward jets ($\lvert\eta\rvert > 2.4$)	76	33	73	24
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	75	31	72	23
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 900\,\text{GeV}$)	75	25	72	17
$\Delta\phi_{\text{min}}^{*} > 0.5$	23	17	22	11

Table 30.6: Cut flow table for `T1tttt` and `T2bb` benchmark models.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})			
	T1tttt (1700, 100)	T1tttt (950, 600)	T2bb (1000, 100)	T2bb (550, 450)
Before selection	100	100	100	100
Event veto for muons and electrons	43	47	100	99
Event veto for single isolated tracks	34	37	97	95
Event veto for photons	34	37	96	95
Event veto for jets failing ID	34	36	96	95
$n_{\text{jet}} \geq 2$	34	36	95	79
$0.1 < \text{CHF}^{\text{j}_1} < 0.95$	33	35	91	75
$p_{\text{T}}^{\text{j}_1} > 100\,\text{GeV}$	33	29	91	52
$H_{\text{T}} > 200\,\text{GeV}$	33	29	91	45
$H_{\text{T}}^{\text{miss}} > 200\,\text{GeV}$	31	6.7	82	18
Event veto for forward jets ($\lvert\eta\rvert > 2.4$)	27	5.7	70	15
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	27	4.9	70	14
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 900\,\text{GeV}$)	27	4.3	62	9.0
$\Delta\phi_{\text{min}}^{*} > 0.5$	7.3	0.4	40	6.3

Table 30.7: Cut flow table for `T2tt` and `T2cc` benchmark models.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})			
	T2tt (1000, 50)	T2tt (450, 200)	T2tt (250, 150)	T2cc (500, 480)
Before selection	100	100	100	100
Event veto for muons and electrons	65	65	68	100
Event veto for single isolated tracks	58	56	59	96
Event veto for photons	57	55	59	96
Event veto for jets failing ID	57	55	59	96
$n_{\text{jet}} \geq 2$	57	55	40	70
$0.1 < \text{CHF}^{\text{j}_1} < 0.95$	55	52	37	64
$p_{\text{T}}^{\text{j}_1} > 100\,\text{GeV}$	55	43	15	55
$H_{\text{T}} > 200\,\text{GeV}$	55	42	13	49
$H_{\text{T}}^{\text{miss}} > 200\,\text{GeV}$	50	16	2.4	36
Event veto for forward jets ($\lvert\eta\rvert > 2.4$)	43	14	2.0	31
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	42	12	1.6	30
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 900\,\text{GeV}$)	40	9.9	0.8	18
$\Delta\phi_{\text{min}}^{*} > 0.5$	25	5.5	0.3	15

Table 30.8: Cut flow table for `T2qq_8fold` and `T2qq_1fold` benchmark models.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})			
	T2qq_8fold (1250, 100)	T2qq_8fold (700, 600)	T2qq_1fold (700, 100)	T2qq_1fold (400, 300)
Before selection	100	100	100	100
Event veto for muons and electrons	100	100	100	100
Event veto for single isolated tracks	97	96	97	96
Event veto for photons	96	95	96	95
Event veto for jets failing ID	95	95	96	95
$n_{\text{jet}} \geq 2$	95	83	95	80
$0.1 < \text{CHF}^{\text{j}_1} < 0.95$	89	78	90	75
$p_{\text{T}}^{\text{j}_1} > 100\,\text{GeV}$	89	59	90	51
$H_{\text{T}} > 200\,\text{GeV}$	89	52	90	43
$H_{\text{T}}^{\text{miss}} > 200\,\text{GeV}$	84	22	76	14
Event veto for forward jets ($\lvert\eta\rvert > 2.4$)	73	19	65	12
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	73	18	63	11
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 900\,\text{GeV}$)	69	12	50	6.9
$\Delta\phi_{\text{min}}^{*} > 0.5$	44	8.6	35	4.9

As with the LLP models, I added them to the AN and made a PR to include them in the master branch. For the work I've done on SUS-16-038, I've been added to the authors list for the paper, which is quite nice. Note that at the time of writing, I'm not a member of the Collaboration authors list. But an exception was made for Ben and me so that we could be included in the list for the paper.

30.4 Discrepancies between the signal acceptance × efficiency and the cut flows

Soon after making the cut flow tables, we realised that the end-of-cut-flow efficiencies didn't match the signal acceptance × efficiency values in the AN. This is a repeat of 2015, but then we attributed it to weights and scale factors. This time, I ran the cut flows with weights so they should have matched. I conducted various studies trying to debug this. Eventually, I used FAST-RA1 (see Sec. 36) with AlphaTools integrated within to try and further debug what was going on. By removing `skimSequence` from `sequence2016`, the weighted cut flow efficiencies agreed better than before. However, they were still out of our acceptance. Ben managed to check the efficiencies of the skimmers in FAST-RA1 and synchronise the skimmer cut flow with the one Rob and I agreed on for the tables. Whilst the new efficiencies didn't agree *exactly*, they were very close for several models. We didn't end up showing the tables for every benchmark model, just a few for re-interpreters. They are displayed on the public webpage at `http://cms-results.web.cern.ch/cms-results/public-results/publications/SUS-16-038/`. As of May 2018, the paper had also been published in JHEP: [64], and **I am on the authors list for it**.

Chapter 31

Analysing the T2tt-4bd SUSY model (26/10/2017)

I've been tasked with making trees for, and analysing the T2tt-4bd SUSY model. This includes running the limits, systematics and cut flow tables. The model is a bit different because it's a four-body decay mode rather than two-body as with the other models.

The dataset is **/SMS-T2tt_dM-10to80_genHT-160_genMET-80_mWMin-0p1_TuneCUETP8M1_13TeV-madgraphMLM-pythia8/RunIISpring16MiniAODv2-PUSpring16Fast_80X_mcRun2_asymptotic_2016_miniAODv2_v0-v1/MINIAODSIM**. The mass points are generated with m_{Stop} between 250 and 800 GeV and mass splittings ($m_{\mathrm{SUSY}} - m_{\mathrm{LSP}}$) between 10 and 80 GeV for each m_{Stop} point.

31.1 Tree production

I submitted the jobs for tree production via CRAB. With CMSSW_8_0_25, and the branches "heppy_80X_ra1-0.7.x-Moriond17Prod" and "80X-ra1-0.7.x-Moriond17Prod_T2tt-4bd_Crab", I could run it with the command

```
heppyCrabAlphaT.py -c
   AtLogicNoSelection_MCMiniAODv2_SUSY_SMS_FastSim -d RA1 -s
    T2_UK_SGrid_Bristol
```

I could monitor the jobs as usual with `crab status`. But if I typed

```
while true; do crabMonitor.py <CRAB submission path> -r;
   sleep 20m; done
```

it would check the jobs and resubmit the failed ones every 20 minutes. Once the jobs were finished, the next step was combining them. But as these weren't standard batch jobs, the procedure for doing so was different. Each chunk was split into a tree, and then the rest of the information from the analysers was wrapped in a tar ball. So I needed to hadd the root files to get the final tree, and separately untar, then hadd the other information from each chunk. For the root files, trying to hadd them normally didn't work. Both Ben and I got weird errors when trying to do, suggesting there was something wrong with the trees. So Ben wrote a script (chain.py) to `TChain` them together to create a "chained" root file that should mimic a hadded tree. I had to supply arguments such that

```
python chain.py <output root file> <absolute path to input
   trees>
```

But for the rest, I had to use another script from Ben (that I modified a bit) to untar these chunks and format the directory names for `heppy_hadd` to work. It can be found here: copy_untar.py.

Once the output was combined, I transferred it to Imperial and split the trees. One caveat was the final step of actually splitting the trees. ROOT treats TChains and TTrees differently when cloning trees from those types. So, in **cutTree.cpp**, I had to change L44 from

```
TTree *newtree = chain->CopyTree(cut);
```

to

```
TTree *newtree = chain->GetTree()->CopyTree(cut);
```

Then the splitting worked. I renamed the directories with a little hacky script I wrote (needs to be in the base directory of the split trees):

```
import os
import sys

original_dirs = []
original_dirs.extend(os.listdir( os.getcwd()) )
print "Original directories:", original_dirs

for dirs in original_dirs:
    if ".py" in dirs: continue # so script isn't renamed
```

```
    bit_needed = dirs[27:] # Omit everything before the mass
    points
    new_dir = "SMS-T2tt"+bit_needed
    print "Old directory ->", dirs, "to new directory ->",
  new_dir
    os.rename(dirs, new_dir)
```

So that the directories were of the form **SMS-T2tt_mStop-<#>_mLSP-<#>_25ns**. The location of the trees is **/vols/cms/RA1/80X/MC/20171026_T2tt_4bd/**.

31.2 Analysis

There are several components that make up an analysis of any model. For this model, we need

– Feynman diagram

– Limit plane

– Uncertainties from systematic sources for benchmark models

– Cross section limits for benchmark models

– Signal acceptance $\times$ efficiency for benchmark models

– Plot of the most sensitive n_{jet} categories

– Significance scan

– $\sigma/\sigma_{\text{theory}}$ (limits per bin) plots for benchmarks

– Mountain range plots for benchmark models

– Cut flow tables for benchmark models

31.2.1 StatsAnalyzer in AlphaTools

In AlphaTools, I could go to **Configuration/Samples/**, and add the model and directory to **produceSampleFilesSMS.py**. I could run that script without any arguments to generate a file with a list of the samples for the model, and added them to a collection. I needed to import the `sampleList80X_T2tt` function into **samples_13TeV_80X.py** and **Analyzers/StatsInput/produceShellScript.py**. Then, I added the base directory to the split trees in `baseDirSignalModelsDict` in **config_cfi_2016.py**.

Navigating to **Analyzers/NIsrAnalyzer/**, I had to add the name of the model (just "T2tt") to **NIsrAnalyzer_cfg.py** and run it. This gives the ISR weights for each sample in text files. Then, I

could compile them in a pickle file by running the script **compileSignalNormPickle.py**. I initially got an error that the output directory for the pickle didn't exist, but I could just create it. But **Producers/WeightNIsr.py** points to the wrong directory in the following step. So, I needed to set

```
self.normPickleDir=os.environ["ALPHATOOLSDIR"]+"/Data/
    signalNIsrNorms/
```

Finally, I could run StatsAnalyzer on these samples. Going to **Analyzers/StatsInput/**, I ran

```
for var in "" "--genMet" "--jec both"; do python
    produceShellScript.py -o <output dir> ${var} --submit --
    signalModel "T2tt"; done
```

and combine the output with

```
for var in "" "GenMet" "both"; do python haddSignalModel.py
    -i <output dir from previous step>/T2ttOutput${var} -f;
    done
```

That gives me root files with the stats information for the signal and necessary control regions. I moved the directories containing everything to where the trees were stored on **/vols**.

31.2.2 Limits in AlphaStats

Now, I have the input needed for AlphaStats. If I edited **configStatFormula.py**, I edited the variables according to https://github.com/shane-breeze/AlphaStats/wiki/DM-Limits and ran Step 1 (optimiseBinning):

```
python batchSubmitOptimise.py -o $OUTDIR -f --options "--
    shapeSystFromFile --getDataLumi --runFormula  --
    extrapolateZinv --greenBand" --submit
```

where `OUTDIR` is the output directory for the datacards and AlphaStats output. Then I could run Step 2 (makeCardsAndWs) with

```
python makeCardsAndWs.py -i $OUTDIR -c "all" --
    bbbFormulaUssr --bbbOptSig
```

where the `-bbb` stands for "bin-by-bin". At this stage, a file will be generated called **signalModelsNoFail.txt**. It's a good idea to compare that with **signalModels.txt** to see if any mass points failed in one of the previous steps. Especially if the limit plot is going in a publication, it's good to have at least 98% of the mass points succeed, preferably all of them. Step 3 (runCombineTask) could be run with

```
python runCombineTask.py -i $OUTDIR -t ASCLS_UL_PRIOR --what
    both
```

The option `-t ASCLS_UL_PRIOR` is needed when running limits workflow. Once that's done, I can actually run the script to make the limit plots:

```
python PlotScripts/makeFinalPlane.py --scenario both --model
    "T2tt" -i $OUTDIR -o <output dir for plots> --mode ul --
    doubleTranspose --addTheory --remake --remakePickle --
    smooth
```

Then, I'll have several plots in the output directory for both expected and observed limits. However, they don't look particularly nice. Cloning the PlotsSMS repository from CMSRA1, outside of AlphaStats, gives me the code I need to make nice plots. In that repo, I need to go **PlotsSMS/config/SUS16038/** and add the expected and observed root files from the `makeFinalPlane` output directory. I also need to make a config in the same style as the others, and call it **T2-4bd_SUS16038.cfg**. The code defines all the SMS models and uses the part of the string before the underscore to determine which model is being plotted. Within the config, I had to point to the root files and the folder within the files that contained the histograms. Once I've sourced a CMSSW environment, I could run the script with

```
python python/makeSMSplots.py config/SUS16038/T2-4
    bd_SUS16038.cfg <label - normally model name>
```

which should give me nice limit plots. If I wanted to change anything aesthetically, I could either check the function corresponding to the model I was running over in **python/sms.py**, or in **python/smsPlotABS.py** (the `graphWhite` object handles the size of the legend with `SetPoint` 2 and 3 setting its height).

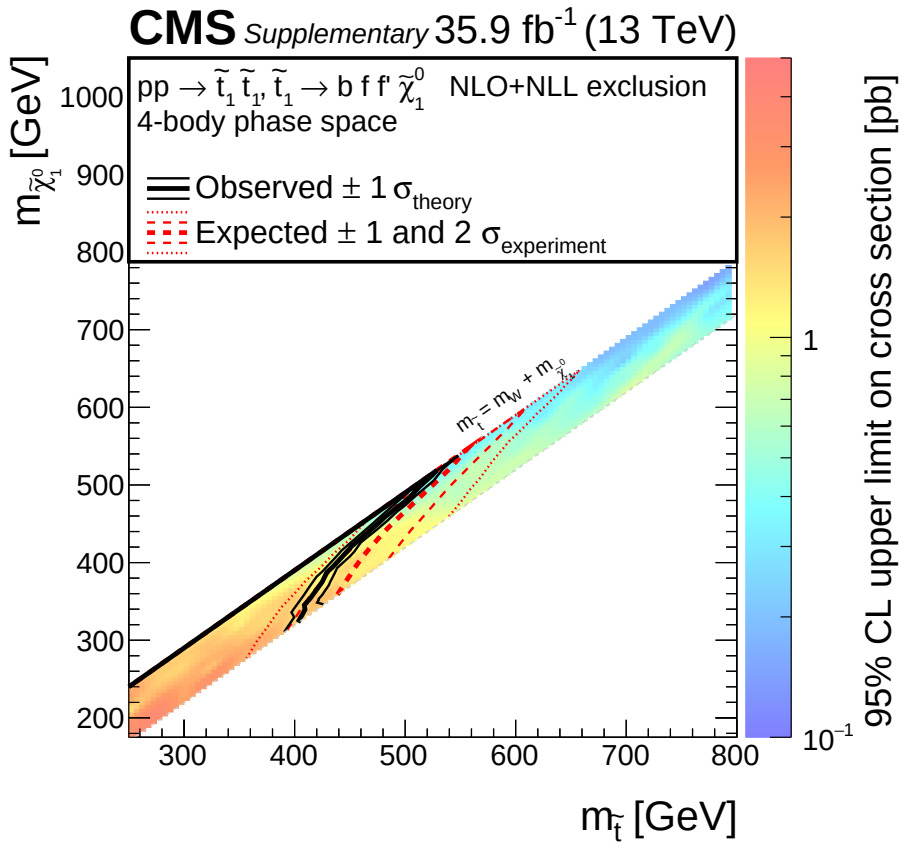

Figure 31.1: T2tt-4bd: Upper limit on the cross section in the mass plane.

Rob asked whether there was any signal contamination from the $\mu+$ jets CR, and it turns out there wasn't. This was a bug that appeared in some of the other SMS models as well, more evident in T1tttt and T2tt as they're more leptonic than the other models. The signal contamination should be most evident for mass splittings of ~ 80 GeV, as the sparticle can decay with an off-shell W boson that decays leptonically. The source of the bug was due to the trigger bit information not being in the trees (which I noticed previously when making the cut flow tables in Sec. 30). However, we wouldn't need to regenerate the trees *with* the trigger bits (as that would take ages), we'd just need to comment out `skimSequence.append(triggerSkimmer2016)` in **Sequences/Sequence2016.py** in AlphaTools. Then, I could just re-run the limits workflow from the StatsAnalyzer stage.

31.2.3 Feynman diagram

I could find the Feynman diagram at https://twiki.cern.ch/twiki/pub/CMS/SUSYSimplifiedModelDiagrams/. It is displayed below.

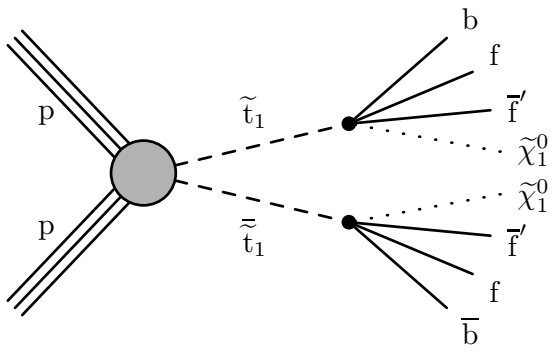

Figure 31.2: Graphical representation of the production and decay of supersymmetric particles in models with the production of third generation squarks (stops).

31.2.4 Systematics for benchmarks

For the systematic uncertainties, I can follow Shane's workflow at https://github.com/shane-breeze/AlphaStats/wiki/Signal-systematic-studies. Basically, I need to look at the systematics that are present in the AN and – apart from MC stat. and luminosity – see if they are in the list `runArguments.signalTemplatesFromFile` in **configStatFormula.py**. Then, I need to make sure those systematics are included in the `systList` list in **Scripts/makeSignalTemplateStudy.py**. I also need to make sure `"mcStat"` is in that list. Then, for AN-17-122, the list should look like

```
systList = [
    "nIsrWeight",
    "jecWeight",
    "puWeight",
    "bsfWeight",
    "bsfLightWeight",
    "bsfCFbWeight",
    "bsfCFlWeight",
    "bsfCFcWeight",
    "triggerWeight",
    "mcStat",
    ]
```

Then, I can run

```
python Scripts/makeSignalTemplateStudy.py -i <datacard
   directory> -o <output directory> --filters <bechmark
   model>
```

A root file containing the systematic variations per njet, nb and HT bin is created in the output directory, and the systematic variations I need to include in the table in the AN are printed to the terminal. Note that the name of the systematic source isn't printed with each range, but they're printed in the order given in `sysList`. The luminosity uncertainty is usually just stated in the AN and is the same for each model, so I can just use that value. The strings `bsfCF+Weight` refer to the FastSim uncertainties. Replacing the "+" with a "b" is the b-tag uncertainty, "c" is for c-tags, and "l" is for light-tags. Then `bsfWeight` is the FullSim b-tag uncertainty and `bsfLightWeight` is the FullSim Mistag uncertainty.

Model	$(m_{\text{Susy}}, m_{\text{LSP}})$	Luminosity	ISR	JEC	PU	b-tag (Fullsim)	Mistag (Fullsim)	b-tag (Fastsim)	c-tag (Fastsim)	light-tag (Fastsim)	Trigger	MC stat.
T2tt-4bd	(450, 400)	2.6%	4-20%	5-12%	7-12%	2-4%	2-3%	2-5%	2-5%	1-7%	2-3%	6-21%

Table 31.1: T2tt-4bd: Representative range taken from the 16% and 84% percentiles of the uncertainty across the analysis bins for each source of signal systematic. One benchmark point is chosen for this model, corresponding to the "compressed" scenario, i.e. with small mass splitting between the mother particle and the LSP.

As a cross-check, I looked at the T2cc systematics as the values should be similar. They were, so I assume I've done everything correctly.

31.2.5 Cross section limits for benchmarks

The expected (μ_{exp}) and observed (μ_{obs}) upper limits on the production cross section are given in a table in the AN. For each of those limits, there are values for "nominal" and "simplified" binning. For nominal, the values should exist in the datacards I made when running the limits. Going into the datacard directory for the benchmark model, I need to look in the file **comb_<model>_mht_card_ASCLS_UL_PRIOR_test.log_<obs/exp>.txt** For the expected limit, there should be a line with `Expected 50.0%: r < <number>`, where `<number>` is what I want. The value used for the cross section limit is always taken from the 50.0% point. For the observed limit, there should be a line with `Observed Limit: r < <number>`.

For simplified, I need to re-run optimiseBinning, makeCardsAndWs and runCombineTask for the benchmark point (putting the output in a new directory for simplicity). But first, in **configStatFormula.py**, I need to set `doBenchmarks` and `doSuperSignal` to `True`. These switch the binning from nominal to aggregated. I also had to add the benchmark mass points to `inputArguments.sigHists` under the `if doBenchMark:` statement. After the three steps have been run, I can follow the same procedure as nominal to get the simplified cross section limits.

Benchmark models		Nominal		Simplified	
$(m_{\text{SUSY}}, m_{\text{LSP}})$ [GeV]		μ_{exp}	μ_{obs}	μ_{exp}	μ_{obs}
`T2tt-4bd`	(450, 400)	0.94	1.89	2.16	3.49

Table 31.2: T2tt-4bd: Expected (μ_{exp}) and observed (μ_{obs}) upper limits on the production cross section, expressed in terms of the signal strength parameter, obtained using both the nominal and simplified binning schema.

31.2.6 Signal acceptance × efficiency

This plot can be obtained in relatively few steps. I can run optimiseBinning in the same way as was used to calculate the limits, making sure the variables in **configStatFormula.py** were also the same (as I might have changed it when running other things). Then, I could run the following steps, referring to the output directory as `OUTDIR` as I did previously. I had to run makeCardsAndWs, combining H_{T} and n_{b} with

```
python makeCardsAndWs.py -i $OUTDIR -c "nB"
```

Then,

```
python runCombineTask.py -i $OUTDIR -t ASCLS_UL_PRIOR --what
    expected
```

I could sort the n_{jet} categories by sensitivity with

```
python sortCategories.py -i $OUTDIR -m ul -c "nB"
```

Finally, I just had to add the StatsAnalyzer output directory (or StatsInput directory) to `inputFileStatsAnalyzer` in **makeJetRankingPlot_2016_SMS.py**, and could make the plots with

```
python makeJetRankingPlot_2016_SMS.py -i $OUTDIR -o <output
   dir for plots>
```

This creates the plots of signal acceptance $\times$ efficiency for the mass plane, and also the plot of the most sensitive n_{jet} categories.

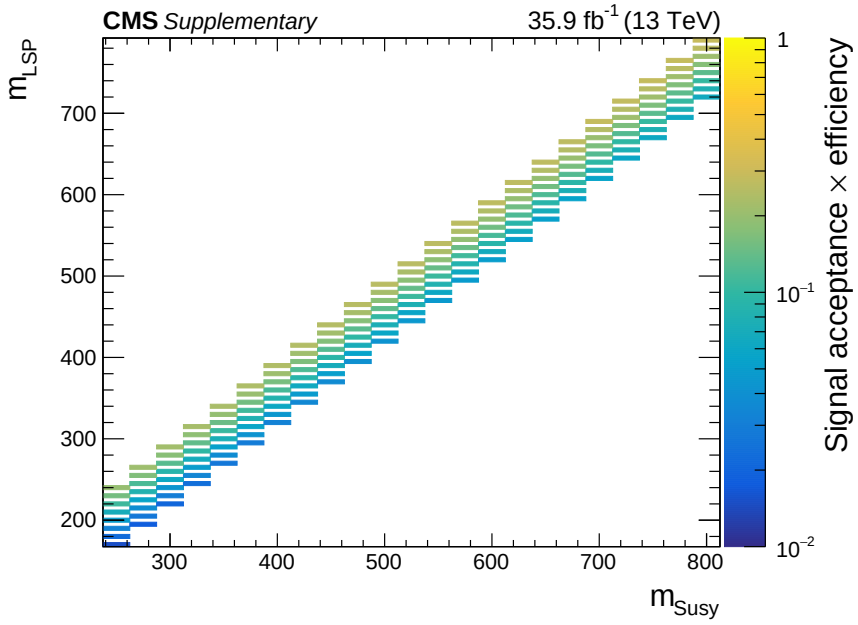

Figure 31.3: T2tt-4bd: ϵ_{sig}.

To extract the value for the benchmark model, need to go to the output directory, into the **eff/**. This should contain a root file with all the information in it. Then, in a ROOT session, I can do

```
TFile f("effHist.root")
f.ls() // To check the histograms
TH2D * h = (TH2D*)f.Get("T2tt_merging_7_cats")
```

```
h->GetBinContent(h->FindBin(450,400))
```

which gives me **0.0935877** for the benchmark model. In a nicer format:

Table 31.3: Signal efficiency for compressed T2tt-4bd model used in the analysis.

Model	$(m_{\text{Susy}}, m_{\text{LSP}})$	Efficiency (total)
T2tt-4bd	(450, 400)	9.4%

31.2.7 Most sensitive n_{jet} categories

This is accomplished when making the signal acceptance $\times$ efficiency plot. In the output directory from makeJetRankingPlot, there are separate plots for each n_{jet} category, and the complete "bit map" plot. That is what needs to be added to the AN.

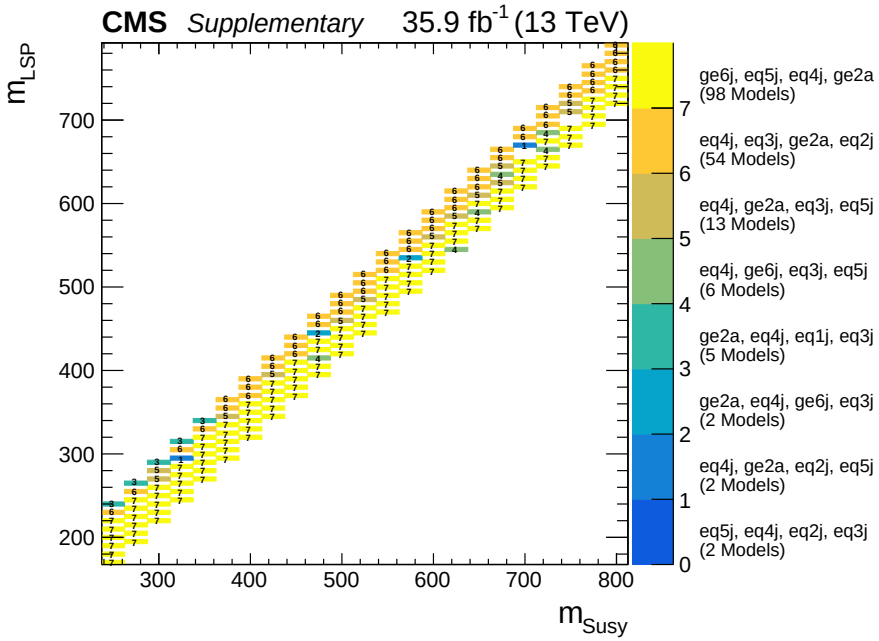

Figure 31.4: T2tt-4bd: Most sensitive categories.

31.2.8 Significance scan

For the significance scan, I want to re-run optimiseBinning and makeCardsAndWs with the same arguments as when running limits, but in a new directory, which I'll label as `OUTDIR`. Then, I need to run runCombineTask with

```
python runCombineTask.py -i $OUTDIR -t SIGNIF --what both
```

making sure `inputArguments.sigHists = "*"` in **configStatFormula.py** (as I might have changed it when running other things). In **PlotScripts/makeFinalPlane.py**, I had to comment out lines 272 (`wMassLine.Draw("same")`) and 274 (`tt.Draw("same")`) so the plot didn't show the dashed line with $m_{\mathrm{SUSY}} - m_{\mathrm{LSP}} = m_{\mathrm{t}}$. Finally, I can run

```
python PlotScripts/makeFinalPlane.py --scenario observed --
    model "T2tt" -i $OUTDIR -o <output dir for plots> --mode
    pv --doubleTranspose --remake --remakePickle --smooth --
    task SIGNIF
```

to get the plots out. I want the interpolated version, which looks like this:

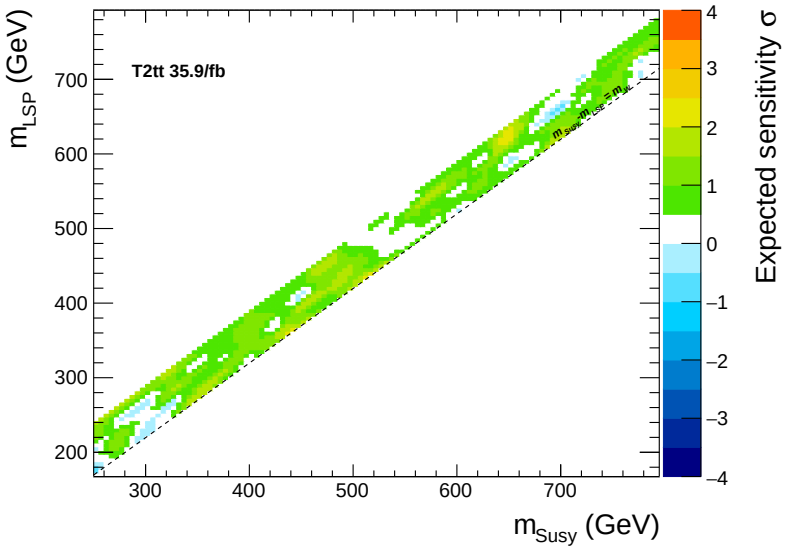

Figure 31.5: T2tt-4bd: Significance scan.

I just had to change the text near the top left from "T2tt" to "T2tt-4bd".

31.2.9 $\sigma/\sigma_{\mathrm{theory}}$ (limits per bin) plots for benchmarks

This is probably the most confusing part of the analysis. First, I had to edit `inputArguments.sigHists` in **configStatFormula.py** to include only the benchmark models (only one in this case). The rest of the environment should be set up as if I'm running limits (i.e., no `doSuperSignal`, etc.). Then, I had to run optimiseBinning on that model twice: once with the standard arguments when running limits; and once with the standard arguments plus the option

-gbForZinvExtrap, which uses the greenband prediction for the $Z \rightarrow$ inv. n_{b} extrapolation. The datacards generated must be in different directories, so I had to set the -o option to different paths. The commands would look like this:

```
python batchSubmitOptimise.py -o <output dir without gb
   label> -f --options "--shapeSystFromFile --getDataLumi --
   runFormula --extrapolateZinv --greenBand" --submit
python batchSubmitOptimise.py -o <output dir with gb label>
   -f --options "--shapeSystFromFile --getDataLumi --
   runFormula --extrapolateZinv --greenBand --
   gbForZinvExtrap" --submit
```

Once done, I had to duplicate each of the output directories so I had four (two with -gbForZinvExtrap and two without). Then, I had to run makeCardsAndWs four times with different options for each. In one of the directories without -gbForZinvExtrap, I had to run with the option -c "all" to combine all bins, and in the other I had to run with -c "nB" to combine everything apart from n_{jet}. Then, in one of the directories with -gbForZinvExtrap, I had to run with -c "ht" to combine only in H_{T}, and in the other directory I had to run with -c "none" to combine nothing. So, the commands would look like this:

```
python makeCardsAndWs.py -i <datacards dir without gb 1> -c
   "all" --bbbFormulaUssr --bbbOptSig
python makeCardsAndWs.py -i <datacards dir without gb 2> -c
   "nB" --bbbFormulaUssr --bbbOptSig
python makeCardsAndWs.py -i <datacards dir with gb 1> -c "ht
   " --bbbFormulaUssr --bbbOptSig
python makeCardsAndWs.py -i <datacards dir with gb 2> -c "
   none" --bbbFormulaUssr --bbbOptSig
```

Once finished, I could run runCombineTask for each directory with the same arguments as limits:

```
python runCombineTask.py -i <datacards dir> -t
   ASCLS_UL_PRIOR --what both
```

After that, I could go to the directory **dataframes/** and run

```
python flatten_valid_bins.py -o ./flattened_bins.txt inputs/
   valid_bins.txt
```

to make a flat list of the valid bins, using **inputs/valid_bins.txt** as a template. Then, I went to **limitsPerBin/** to run **combineLimitsToDataframe.py** to create dataframes. Note that wildcarding is allowed, so I could specify the input directories like so:

```
python combineLimitsToDataframe.py -i /vols/cms/RA1/80X/MC
    /20171026_T2tt_4bd/AlphaStats_Benchmark_* -o <output dir>
     --valid-bins ../flattened_bins.txt
```

X11 forwarding is required for the final step. So make sure to `ssh -Y`, then run **makeLimitPerBinPlots.py** with

```
python makeLimitPerBinPlots.py -i <output dir from previous
    step> -o <output dir for plots>
```

which gives me the following plot for the benchmark model:

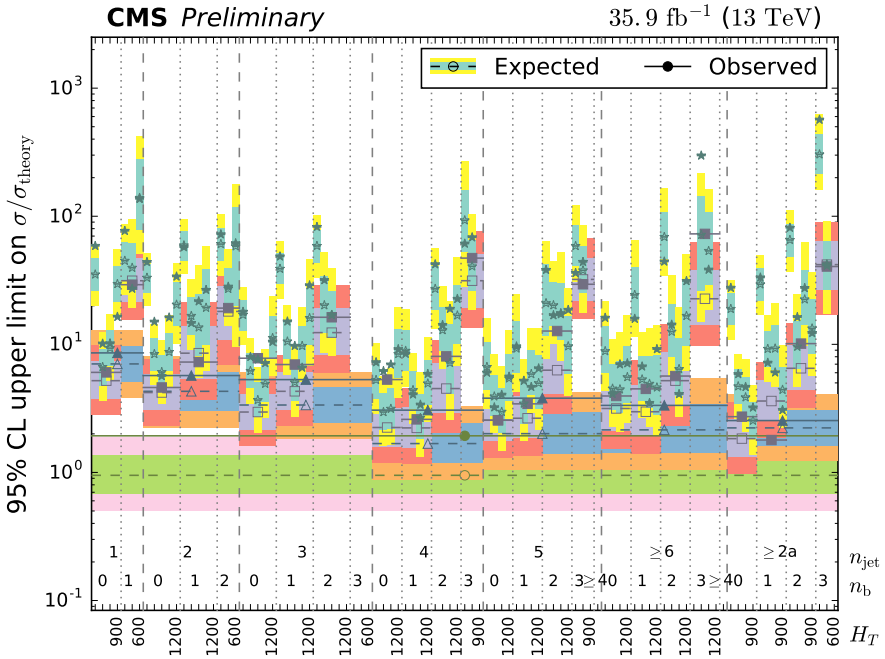

Figure 31.6: 95% CL upper limits on $\sigma/\sigma_{\text{theory}}$ for the (450, 400) T2tt-4bd benchmark model are shown for four different bin combinations: no bins, H_{T}-only, (n_{b}, H_{T}) and all bins overlaid in ascending order. The expected limit, along with the $\pm 1\sigma$ and $\pm 2\sigma$ bands, and the observed limit are displayed for each bin.

31.2.10 Mountain range plots for benchmarks

Ben volunteered to make the mountain range plot for the benchmark model, as the README in AlphaStats was out of date. The plot is here:

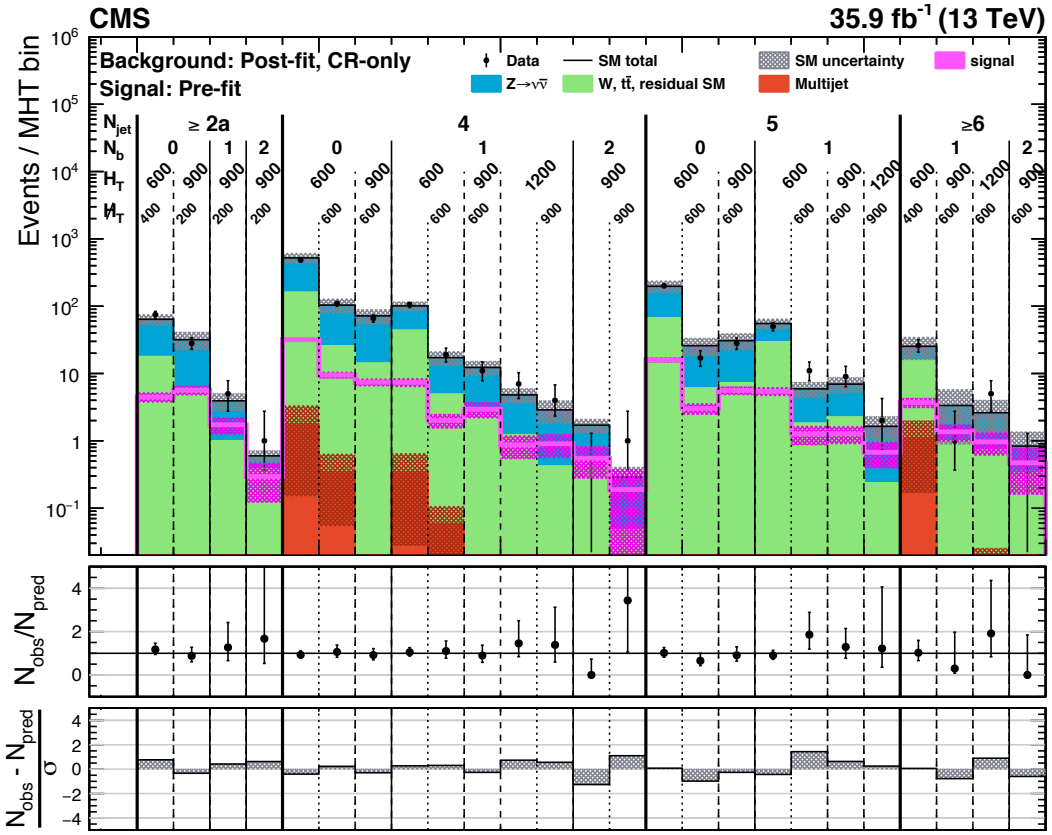

Figure 31.7: Pre-fit T2tt-4bd benchmark model overlay on CR-only post-fit background prediction for the simplified bins. The uncertainty on the signal model counts represents the statistical uncertainty due to the finite size of the of the simulated sample.

31.2.11 Cut flow table for benchmarks

For the cut flows, I used the same workflow as in Section 30 to re-weight the split trees I already had for the benchmark model. I made sure to use the same branch of AlphaTools, just adding the sample to the T1qqqqLL sample file, and point to the ISR pickle I generated earlier for this model. Then, I could just point cutflowirl to the directory of the reweighted tree and run the code as normal to get the table.

Table 31.4: Cut flow table for `T2tt-4bd` model.

Event selection	Benchmark model (m_{SUSY}, m_{LSP})
	T2tt-4bd
	(450, 400)
Before selection	100
Event veto for muons and electrons	79
Event veto for single isolated tracks	71
Event veto for photons	71
Event veto for jets failing ID	71
$n_{\text{jet}} \geq 2$	55
$0.1 < \text{CHF}^{\text{j}_1} < 0.95$	50
$p_{\text{T}}^{\text{j}_1} > 100\,\text{GeV}$	41
$H_{\text{T}} > 200\,\text{GeV}$	37
$H_{\text{T}}^{\text{miss}} > 200\,\text{GeV}$	26
Event veto for forward jets ($\lvert\eta\rvert > 2.4$)	22
$H_{\text{T}}^{\text{miss}}/E_{\text{T}}^{\text{miss}} < 1.25$	20
H_{T}-dependent α_{T} requirements ($H_{\text{T}} < 900\,\text{GeV}$)	11
$\Delta\phi_{\text{min}}^{*} > 0.5$	8.3

31.3 Follow up and approval

Once I added everything to the AN, I could build a draft copy to see how everything looked. If I went to **AlphaTDR2/**, I had to set up the environment with

```
eval `notes/tdr runtime -sh`
```

Then I could build the pdf with

```
cd notes/AN-17-122/trunk
tdr --draft --style=an b AN-17-122
```

Then the path to the pdf will be given.

I gave a presentation to RA1 about the analysis for this model: T2tt-4bd analysis complete (16-11-2017). The main, interesting comments were about the limit per bin plot, the cut flow tables and the most sensitive n_{jet} categories plot. Shane expected the $2b$ categories to drive the sensitivity in the limit per bin plot. But because the mass splitting is small, there's only a small amount of energy to be distributed among the $b$jets, making them soft. With the cut flow tables, the same effect of the

efficiencies not agreeing with the signal acceptance $\times$ efficiency values, applies here. The plot gives an efficiency of 9.4%, while the table gives it as 8.3%. I think this is due to the plot calculating the post-fit efficiency, while my tables calculate the pre-fit (because I only re-weight the trees). Then, in the n_{jet} categories plot, two mass points were missing. In the raw plot, those two mass points were present but the values associated to their most sensitive categories were low, and so weren't included in the final plot. I don't think this is an issue as only 1% of the models are affected.

I had to give an approval talk for the model at a SUSY inclusive meeting. The slides are here: 20180213 T2tt-4bd approval.

31.4 Analysing the chargino-mediated model (T2bW_X05, 03/04/2018)

At the approval talk, one of the suggestions was to instead analyse the chargino-mediated model (sometimes referred to as "T2bW" or "T2bW_X05") rather than the prompt decay model. Finding the Higgs at 125 GeV gives this model a better chance of being found over the prompt decay. In this version, the stop decays into a chargino and b, with the chargino decaying into the LSP and W: $p \to \tilde{t}\bar{\tilde{t}},\ \tilde{t} \to b\tilde{\chi}^{\pm},\ \tilde{\chi}^{\pm} \to W^{*}\tilde{\chi}^{0}_{1}$.

I used CMSSW_8_0_25 with my fork of *cmgtools-lite-private*, the CMGTools branch *80X-ra1-0.7.x-Moriond17Prod-EshSUSYcutflow2016* and added the dataset **/SMS-T2bW_X05_dM-10to80_genHT-160_genMET-80_mWMin-0p1_TuneCUETP8M1_13TeV-madgraphMLM-pythia8/RunIISpring16MiniAODv2-PUSpring16Fast_80X_mcRun2_asymptotic_2016_miniAODv2_v0-v1/MINIAODSIM** to the components. I ran AtLogicNoSelection to make the flat trees, storing them at **/vols/cms/RA1/80X/MC/20180329_S01/** on Imperial and analysed them in virtually the same fashion as the prompt model.

When I split the trees, I had to rename the directories to "SMS-T2bW_X05_<blah>" and make sure the model name "T2bW_X05" was uncommented in the various files in AlphaTools I needed to use. I then had to *manually* make a sample file containing all the directory names because of how the splitting works and the troublesome underscore in the model name. I could import that file in the scripts **samples_13TeV_80X.py** (where I also had to add the line `sampleList80X.addSampleHandler("<collection name>")`) and **Analyzers/StatsInput/produceShellScript.py**. Then, I could add the name of the collection to `psetSignalModels2016.sampleSelection` in **Analyzers/NIsrAnalyzer/NIsrAnalyzer_cfg.py** and run the script to get the ISR weights in a root file. I then had to change the signal model in **compileSignalNormPickle.py** and run that to get the pickle file. Once completed, I had

to add the model and collection name to `sampleListDict` in **produceShellScript.py** and could then run the StatsAnalyzer steps as normal. The subsequent AlphaStats steps were also the same, bearing in mind the model name.

The plots and tables for the analysis are below.

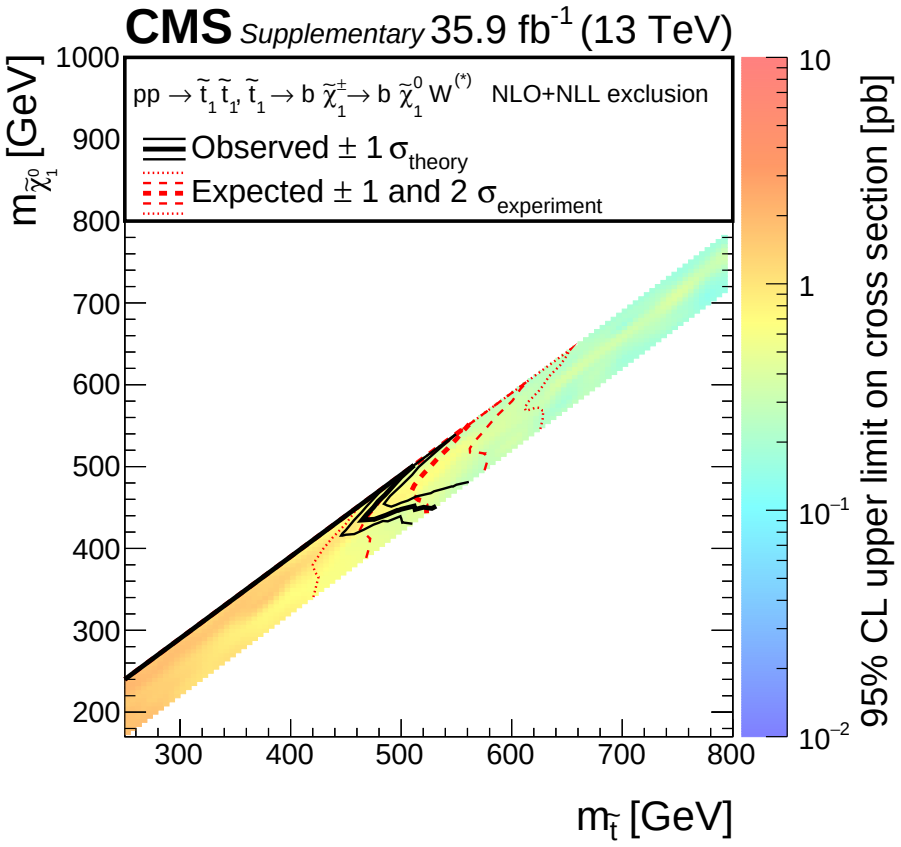

Figure 31.8: T2bW_X05: Upper limit on the cross section in the mass plane.

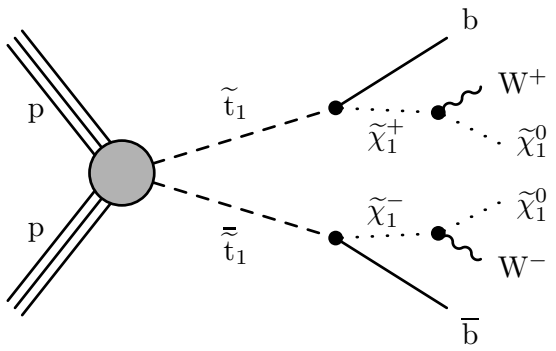

Figure 31.9: Graphical representation of the production and decay of supersymmetric particles in models with the production of third generation squarks (stops).

We chose a benchmark model at (500, 460). Usually, when choosing benchmark points, we want to choose a point near the expected exclusion. From the limit plot, (500, 460) seemed like the best option. Below is the table of systematic uncertainties for the benchmark model.

Model	$(m_{\text{Susy}}, m_{\text{LSP}})$	Luminosity	ISR	JEC	PU	b-tag (Fullsim)	Mistag (Fullsim)	b-tag (Fastsim)	c-tag (Fastsim)	light-tag (Fastsim)	Trigger	MC stat.
T2bW_X05	(500, 460)	2.6%	4-22%	4-13%	8-12%	2-4%	1-2%	2-5%	3-7%	1-8%	2-2%	7-21%

Table 31.5: T2bW_X05: Representative range taken from the 16% and 84% percentiles of the uncertainty across the analysis bins for each source of signal systematic. One benchmark point is chosen for this model, corresponding to the "compressed" scenario, i.e. with small mass splitting between the mother particle and the LSP.

Benchmark models		Nominal		Simplified	
$(m_{\text{SUSY}}, m_{\text{LSP}})$ [GeV]		μ_{exp}	μ_{obs}	μ_{exp}	μ_{obs}
T2bW_X05	(500, 460)	0.9961	1.8094	2.1953	3.4695

Table 31.6: T2bW_X05: Expected (μ_{exp}) and observed (μ_{obs}) upper limits on the production cross section, expressed in terms of the signal strength parameter, obtained using both the nominal and simplified binning schema.

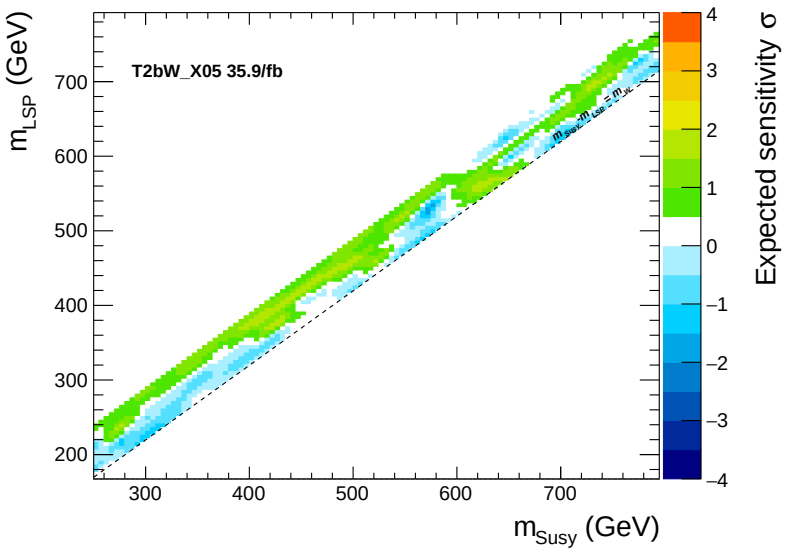

Figure 31.10: T2bW_X05: Significance scan.

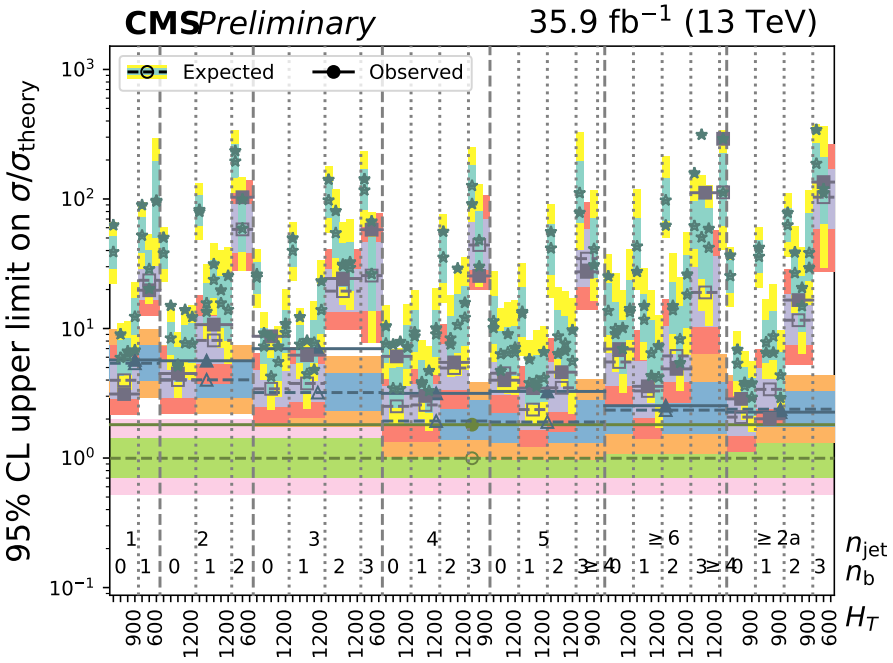

Figure 31.11: 95% CL upper limits on $\sigma/\sigma_{\text{theory}}$ for the (500, 460) T2bW_X05 benchmark model are shown for four different bin combinations: no bins, H_{T}-only, (n_{b}, H_{T}) and all bins overlaid in ascending order. The expected limit, along with the $\pm1\sigma$ and $\pm2\sigma$ bands, and the observed limit are displayed for each bin.

Chapter 32

An introduction to the dark Higgs (17/11/2017)

As we'll likely be doing more dark sector work in RA1's future, I thought I should get caught up on the theory and motivation behind some of these models. One of these is the "dark Higgs", a scalar boson that gives mass to dark sector particles and has a coupling to the standard model Higgs. These models can also include a spin-1 mediator (Z') that can allow dark matter to interact with Standard Model particles. An overview of some dark Higgs models and the feasibility of a search at the LHC is discussed in Ref. [65].

Chapter 33

Service Work: Trigger Shifts at P5 (22/11/2017)

To gain additional EPR credit that gets me closer to authorship, I should sign up for trigger shifts at P5. The url to sign up is https://cmsonline.cern.ch/webcenter/portal/cmsonline/pages_common/shiftlist. Once shifts open up, it's good to grab them as quickly as possible. My shifts are:

- 9th and 10th May 2018 (training shifts, first one with Simone, second with Wei Shi) – 0700 to 1500
- 14th, 15th, 16th May 2018 – 1500 to 2300
- 25th, 26th, 27th May 2018 – 2300 to 0700
- 14th June 2018 – 1500 to 2300
- 22nd, 23rd, 24th June 2018 – 1500 to 2300
- 18th October 2018 – 1500 to 2300
- 26th, 27th, 28th November – 0700 to 1500

I needed to arrange training shifts, take some online courses (at http://sir.cern.ch/) and request CMS Control Room access, all of which are detailed at https://twiki.cern.ch/twiki/bin/viewauth/CMS/TriggerShifterQuickTutorial#Prerequisites.

For getting to P5 and back, there are shuttle buses that leave from CERN and some bus stops in St. Genis. The timetable is detailed at https://smb-dep.web.cern.ch/en/ShuttleService/Circuit3.

A tutorial session was run, detailing the basics of shifts. The slides are here: Trigger shifter tutorial – Introduction, control and configuration and Trigger shifter tutorial – Monitoring, troubleshooting and summary.

33.1 On shift

When on shift, I essentially need to monitor the trigger subsystems and rates, and check everything is okay. When I first get into the Control Room, the previous shifter will tell me about anything noteworthy that happened on their shift. The monitors should already have the relevant pages loaded (like uGT SWATCH cell, L1Page, DQM monitoring, etc.). I should look at the previous shifter's elog on `https://cmsonline.cern.ch/`. I could find the logs by navigating to Elog > Subsystems > Trigger > Trigger. I should also start my own elog for the shift using my template, logging the state of the run and system when my shift starts, as well when new runs start, the beam is dumped, etc (with the context surrounding it). I can check the rates on the page documented in the list below. The plot will show the L1 rate and usually three other triggers (belonging to EG, muons and jets, respectively). Hovering over a curve will give the trigger as a bit number. Then, I can navigate the table below to see which trigger corresponds to the number. At the end of the shift, I should upload my elog.

During stable beams, the prescale column in the uGT SWATCH cell – in uGT SWATCH Cell > Control Panels > uGT Prescales – is arguably the most important thing to keep an eye on. The prescale for a trigger effectively controls the rate. If the prescale for a particular trigger is 1, then an object that would normally fire this trigger will. If the prescale is 2, then half of the trigger firings are recorded. This is useful as there are different triggers for different situations, which therefore require different prescales. Some low-threshold triggers can be useful for calibrations, but can cause very high rates in stable beams, so their prescale is set very high. The prescale columns are usually either bunch-based or lumi-based. When running early in the year and we're ramping up, there's a different amount of bunches at any one time. Whereas later in the year when everything is stable and we run with a set amount of bunches, the prescales should be changed depending on luminosity. The luminosity can be found by checking some of the monitors, and is usually at its highest at the start of a run, and gradually decreases throughout. The prescale column will likely need changing $\mathcal{O}$(1 hour).

The rates, tied to the prescales, are also an important thing to check. During stable beams, the L1 rate should be around 40 MHz (but can reach around 65 MHz at the start of a run as there are more protons in the LHC and so a higher PU) and the HLT rate should be around 1 kHz. If the rates are out of control due to a suspected hot tower, the DQM plots should be checked. These plots are accessible at URL, and hold information from the last 1000 lumi sections. Right-clicking on a plot gives options to adjust the range, or checking for hot towers with `drawoption = hottowers`.

Some useful links are

- Online trigger workbook: https://twiki.cern.ch/twiki/bin/viewauth/CMS/OnlineWBTrigger
- Prescale information (*always load this when I start my shift and when a new run starts as there may be important key/prescale information added*): https://twiki.cern.ch/twiki/bin/view/CMS/OnlineWBL1CollisionPrescales
- Main Level-1 page monitoring: https://l1page.cms/
- uGT SWATCH cell: http://l1ts-ugt.cms:3333/urn:xdaq-application:lid=13/#!/Control%20Panels/1.%20Summary
- Rates page: cmswbm.cms/cmsdb/servlet/TriggerRatesHTML5
- CMS online page: https://cmsonline.cern.ch/webcenter/portal/cmsonline
- Trigger shifter elog page: https://cmsonline.cern.ch/webcenter/portal/cmsonline/pages_common/elog?wc.contentSource=
- Lumi monitoring: https://op-webtools.web.cern.ch/vistar/vistars.php?usr=LHC1
- DAQ status: http://cmsonline.cern.ch/daqStatusSCX/aDAQmon/DAQstatusGre.jpg
- Current rate: es-cdaq.cms/sc/ratemeter.html

I should remember to bring lunch, my laptop (for working when nothing interesting is happening), my laptop charger and a Swiss plug adapter.

33.2 Calo Layer-2 on call

In addition to shifts, I'm expected to sign up as a Layer-2 on-call person. I'm basically given the on-call phone for a shift – each shift being a week long and being expected to do four shifts over the year – and give advice to whomever rings it.

The shifts I've chosen are

- Weeks 17 and 18 – 23rd April to 7th May 2018
- Week 24 – 11th to 18th June 2018
- Week 26 – 25th June to 2nd July 2018
- Week 47 – 19th to 26th November 2018

I'll also have to attend extra meetings, which can be found at https://twiki.cern.ch/twiki/bin/view/CMS/CaloLayer2OnCall#Meetings_to_attend_while_on_call and in my Evernote note.

There are several tutorials and instructions that are helpful when dealing with a call: calo2-oncall-tutorial-dqm.pdf, caloLayer2onCall_onlineSW_02032018.pdf, Tutorial_May_2017_v1.pdf.

I got a call from the L1DOC about an input link error (crcError) on MP5, which is a known problem among Calo Layer-2 and shouldn't be a cause for concern. But Calo Layer-2 was briefly in error (likely from a "blip" where a couple of components were in error, flagging the entire subsystem in that state). A single component warning/error can send the entire subsystem into an "error" state, even though basically everything is fine. The current trigger shifter who noticed the error wrote an e-log detailing it. These will be in the same place as the normal shift logs (see 33.1. I could also write replies to the e-logs to reassure the shifter that there's no real problem so they don't call me or the L1DOC throughout the night. I also posted about the problem on the CaloL2 Ops Skype chat to let everyone know.

33.2.1 Setting up a tunnel to P5

For online DQM (Data Quality Monitoring) and maintenance of the Trigger (like the L1 Page, etc.), I need to set up an ssh tunnel to P5. This is because the relevant webpages and information are only available from within the P5 network. The instructions are at https://twiki.cern.ch/twiki/bin/viewauth/CMS/CaloLayer2OnCall. First, on my computers, I opened **~/.profile** and added this line:

```
alias p5tun='ssh -tN -v -4  -D 55555 ebhal@cmsusr.cern.ch -o
    "ProxyCommand=ssh ebhal@lxplus.cern.ch -W %h:%p"'
```

Then I downloaded the FoxyProxy add-on for Firefox (not Chrome as suggested) and followed the instructions to set it up. Then I followed the link https://gitlab.cern.ch/cactus/cms-ca to install the CMS Level-1 Software Certificate Authority certificate in my browser. I also added my grid certificate (see Sec 18.2) to lxplus. Once those were completed, I can now just type

```
p5tun
```

into my terminal to tunnel to P5. I will be prompted twice for passwords: the first is just my password for CERN's lxplus; the second is the password for my P5/cmsusr/.CMS account, which is my normal password with normal numbers. If I open Firefox, go to https://l1page.cms/ and it loads, it means the tunnel is working. **UPDATE:** newer versions of Firefox aren't very compatible with FoxyProxy so I've also set it up on Chrome on both my laptops.

Chapter 34

Service Work: Jet and $E_{\mathrm{T}}^{\mathrm{miss}}$ studies (22/11/2017)

To become more integrated in the Level-1 Trigger, as well as continuing with the JECs and taking on shifts, I've been asked to take over some of the MET studies Batool has previously done. This should all count towards the EPR I need for authorship, and I can check my progress at https://icms.cern.ch/epr/showMine.

34.1 Understanding high-MET events in Zero Bias

I've been asked to make various plots of events that are firing the `L1_MET100` trigger to see if there are obvious issues with hot towers, ECAL spikes, PU mis-estimation, etc. Aaron wrote a macro that can make some plots needed. I modified it to add extra features, extra plots and tidied up some of the code. Then, I bunged everything into a new GitHub repository for version control and safeguarding – https://github.com/eshwen/High-MET-studies. One of the revisions of the macro is located here:

If I source a CMSSW release that contains the L1Trigger packages (because I need some header files contained within them), i.e., the CMSSW I use for JECs, I can run the macro with

```
root -l -b -q doMETStudy.cxx
```

I initially ran on Soolin as that's where the JEC code was. But if I moved the macro to lxplus and set up the environment, I could use the fact that the ntuples were stored locally on AFS to access

and run over them much quicker (hence the local file paths in the macro). I also realised I could fill histograms with a weight, rather than just incrementing a bin by 1:

Using `h->Fill(quantity, weight)`, I can fill histograms instead of making arrays of different quantities and populating a TGraph. For example, if I wanted to plot the HCAL TP E_{T} against iPhi, I could do `h->Fill(hcaliPhi, hcalTPET)` to increment the iPhi bin by the TP E_{T}.

I made various plots of TP E_{T} against iEta and iPhi for the ECAL, HCAL, towers (sum) and trigger tower (TT) 28. I've made similar plots with $E_{\mathrm{T}}^{\mathrm{T}}$ and $E_{\mathrm{T}}^{\mathrm{miss}}$, as well as plots for individual events to see if the $E_{\mathrm{T}}^{\mathrm{miss}}$ came from large spikes or broad PU jets. I included some plots of the "average" TP E_{T} as well: the total E_{T} in an iEta/iPhi bin divided by its occupancy, as well as the total E_{T} divided by the number of events. These could help identify hot towers, inhomogeneous responses or calibrations, etc. Some of the plots are below, the rest are located in **./sec34/**.

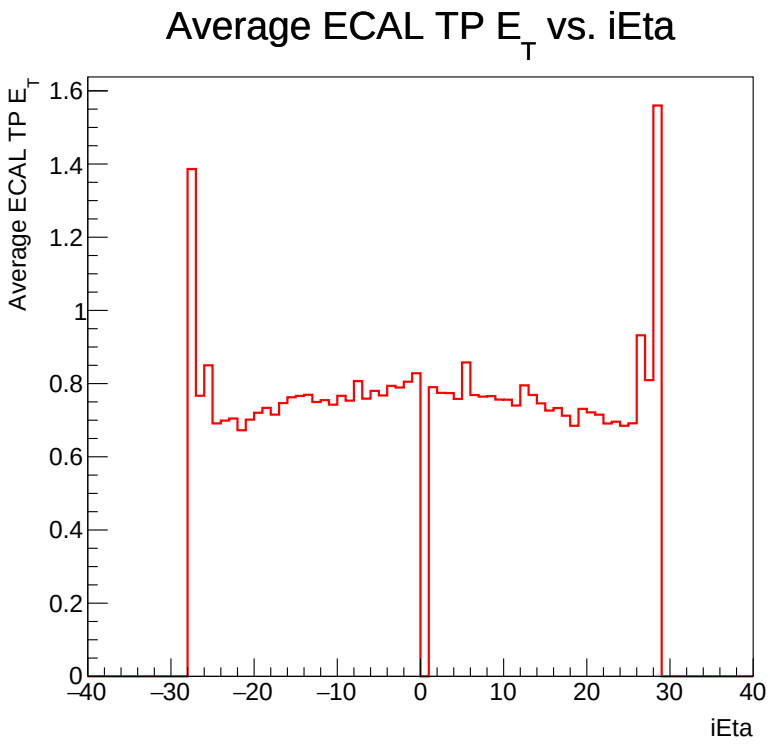

Figure 34.1: The average TP E_{T} (total TP E_{T} in a bin divided by its occupancy) against iEta in the ECAL.

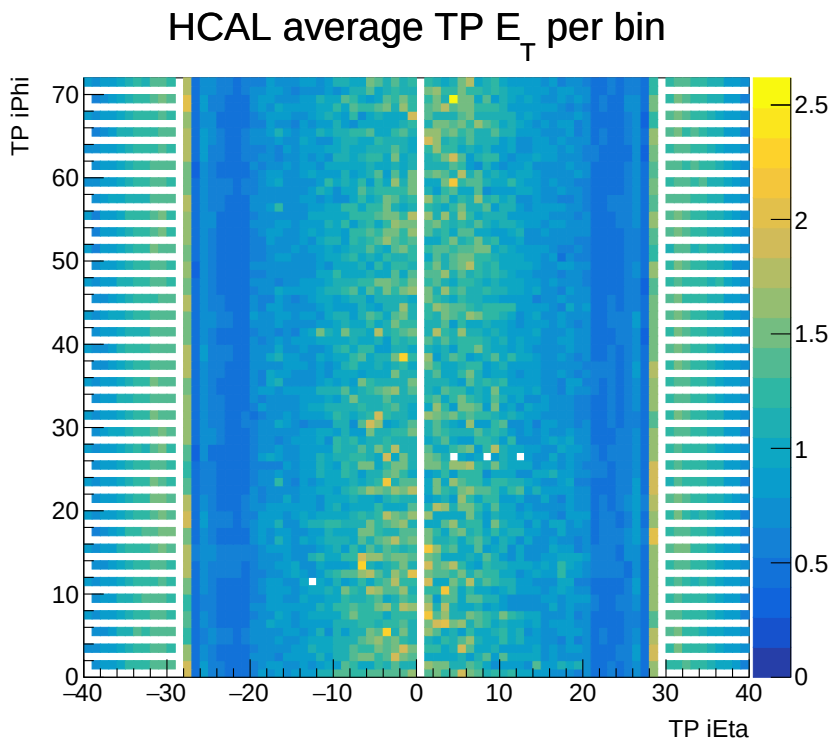

Figure 34.2: The average TP E_{T} against iEta and iPhi in the HCAL.

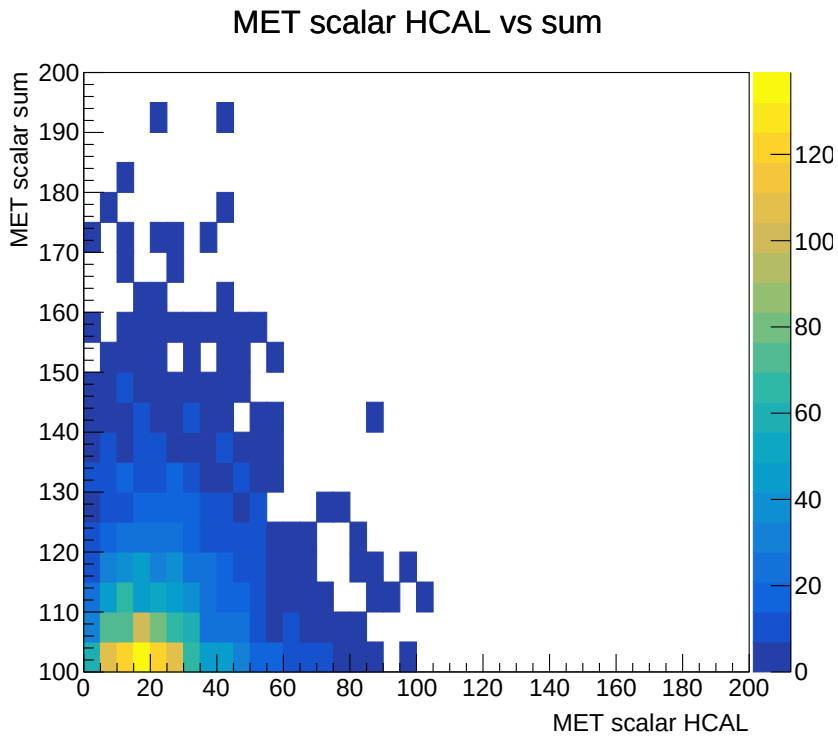

Figure 34.3: The $E_{\mathrm{T}}^{\mathrm{miss}}$ scalar in the HCAL vs the tower (sum).

Some of the plots have been shown in TPG meetings, one of which is here: https://indico.cern.ch/event/698504/contributions/2864568/attachments/1587832/2511546/2018_01_23_L1T_MET_TPs_Esh.pdf.

34.2 Rate vs PU in JetMET for DPS note

Every year, a DPS (Detector Performance Summary) note is produced for each object group in the trigger (muons, jetMET, etc.). For the 2017 performance for JetMET, Aaron and I have been working on the plots to showcase how the trigger has performed for those objects. I've been asked to make the rate vs PU plots with and without the MET PUS (pileup subtraction). Olivier and Chiara (VBF, taus) have code to calculate the rates at https://github.com/camendola/RateStudies, so I followed the README and made the relevant tweaks for JetMET.

The first thing was to get the PU and lumi section for a few runs, which I could do with `brilcalc`. I had to install and initialise it (on lxplus) with

```
export PATH=$HOME/.local/bin:/afs/cern.ch/cms/lumi/brilconda
   -1.1.7/bin:$PATH # Needed each time to initialise
pip install --install-option="--prefix=$HOME/.local" brilws
```

and then I could run it with

```
brilcalc lumi -r <run_no> --byls
```

I should get a table printed to the screen with the relevant info. But I need good lumi sections to work with. If I head to https://cmswbm.cern.ch/cmsdb/servlet/RunSummary and enter the run number, then submit and click on the run number hyperlink, it gives me a summary. Then, if I go to Other services → Lumi Sections, I'll get a green and red grid. By looking in the Physics column, it starts off red and then becomes green. I can take the time at which the column goes green, and then copy the entries from the table I got from running `brilcalc` that are after that time, which have the label "STABLE BEAM". Or I can append the brilcalc command with `> <name>.txt` and remove the entries I don't need from the file created. Then, I need to format the file so the columns are in the order of fill, run, lumi section, avg. PU. Visual Studio Code shortcuts can come in very handy if I want to use a GUI.

I then had to add a list of the ntutples I wanted to run over into a text file, and write a command in **scripts/EvalRatePU.sh** to execute it. The repo wasn't configured to run on Condor, so I had to add support for it in order to run at lxplus.

Note: I couldn't get MET support added in time for the approval of the plots, so they never ended up in the note.

Chapter 35

Semi-visible jets analysis (15/12/2017)

Annapaola and her PhD student Giorgia have been developing the generation of samples and analysis of a dark matter search with semi-visible jets. Ben and I are planning to become involved in this, helping out where we can and hopefully providing some of the framework for my thesis. It is still early days and my first task is to use the code Giorgia has on GitHub to reproduce the sample generation in PYTHIA. The short-term goal is to move the hard scattering aspect and matrix element calculations to MADGRAPH, and showering/hadronisation with PYTHIA to produce samples that can be used in an analysis. MadGraph is better for simulating hard jets, whilst Pythia is better for softer objects.

The following are some slides Giorgia showed at the latest EXO workshop: She postulates that the dark sector consists of at least two "dark quarks" that interact via a new fundamental force. These dark quarks can couple to SM particles via a Z' gauge boson that's leptophobic. The heavier, unstable dark sector particle is created in a collider via $pp \to Z' \to \chi_2\bar{\chi}_2$. The χ_2 then decays into the lighter, stable dark matter χ_1 and jets. The dark matter is recorded as MET, whose direction is roughly collinear with the jets. The rest of the slides discuss the sample generation, event selections, and comparisons to the paper her work has been based off so far: Ref. [59]. Perhaps coincidentally, this is the same paper I read last year. I have a summary in Sec. 17.

A follow-up paper by the same authors was published in 2017: Ref. [66]. This a longer, more-detailed description of the signatures and searches for semi-visible jets. They also include a GitHub repo to the input files needed to generate signal MC events in MadGraph and shower with Pythia: https://github.com/smsharma/SemivisibleJets. They were created with FEYN-

RULES, which can calculate Feynman rules and generate files for new models: http://feynrules.irmp.ucl.ac.be/.

35.1 Summary of model

35.2 Current sample generation in PYTHIA

I forked Giorgia's repo at https://github.com/grauco/SVJ_production and set up a working directory on Soolin (**/storage/eb16003/Semi-visible_jets/**). These consist of three bash scripts that create the Pythia config and batch submission scripts, and then use them to run the sample production on a cluster (where the Tier 3 machine at Zurich is only supported system, at the moment).

The script **set_config.sh** sets up the environments and initialisation, taking care of CMSSW and Pythia in the process. Two versions of CMSSW are needed: 7_1_28 is used for the GEN-SIM production while 8_0_21 is for the generation at reconstruction level. Both include Pythia so there's nothing I need to do for that, and the CMSSW releases are created (if not present already) and then initialised when the script is run. I slightly edited that to clean it up, and started to write a Python version which would be simpler. Then, **set_batch.sh** creates the necessary input files for the job, and **run_bunch_batch.sh** is what the user runs to call **set_batch.sh**.

As this repo is still in its infancy, the files and commands will likely change over time. My fork of the repo (https://github.com/eshwen/SVJ_production) will probably be the most up-to-date, and so the README should be followed for instructions. I've mainly been trying to write a Python version, and add support for running on other sites (such as Imperial and lxplus).

Note: We decided to drop this workflow for sample production in order to switch to MadGraph.

35.3 Producing MadGraph gridpacks for CMSSW insertion

I forked the repo created by the authors of the SVJ papers and used the model files to generate LHE output with MadGraph to test they work properly. The README in my fork (https://github.com/eshwen/SemivisibleJets) details the instructions for doing so.

The important parameters and where they are declared in the MadGraph model files/Pythia parameters are specified below. They can be changed to perform a parameter scan by just editing the variables:

– α_d (running dark coupling, value at 1 TeV) / Λ_d (confinement scale for dark quarks). Specified in Pythia parameters with variable `HiddenValley:Lambda`, where

(35.1) $$\Lambda_d = 1000\ [\text{GeV}] e^{\frac{-2\pi}{\alpha_d b}}$$

and $b = \frac{11}{3}N_c - \frac{2}{3}N_f$ is related to the number of colours and number of flavours [66]. The theorists use an $SU(2)_d$ gauge theory for their dark sector, so I'm using $N_c = 2$. They use a flavour symmetry of $U(1)^{N_f}$ and suggest to set $N_f = 2$, which I also have to specify in `HiddenValley:nFlav` in the Pythia parameters. With these set, the value of α_d should be chosen such that $\Lambda_d \sim M_d$, like in the paper.

– $m_{Z'}$ (s-channel) / m_Φ (t-channel) = 1000 GeV. Specified in MadGraph model file **parameters.py** with variable `MY1` for s-channel as there's only a single mediator; specified by variables `Ms+##` for t-channel because of the bi-fundamental mediators (which are basically identical in implemetation).

– r_{inv} (fraction of stable dark hadrons). Specified in Pythia parameters by adding decay channels for the dark meson, where the branching fraction to dark matter particles is r_{inv} and the branching fraction to SM quarks is $1 - r_{\text{inv}}$.

– g_q (coupling strength between quarks and Z') = 0.25. Specified in MadGraph model file **parameters.py** with variables `gVu##`, `gVd##`. `gAu##`, `gAd##` for s-channel.

– g_d (coupling strength between dark sector particles and Z') = 1. Specified in MadGraph model file **parameters.py** with variables `gVX*`, `gAX*` for s-channel; specified by variables `MWs+##` for t-channel.

– M_d (characteristic mass scale of dark quarks) = 10 GeV. Specified in MadGraph model file **parameters.py** with variables `MX*` for s-channel; specified by variables `Mgv##` for t-channel. By default, these dark quarks are scalar (`spin = 1` in **particles.py**, presumably meaning only one choice of spin). Note that these dark quarks are hadronised in Pythia, where I should specify the resulting dark hadron/meson having a mass of $2M_d$. (See CMSSW gen fragment later for implementation.)

– Particle spin. Specified in **particles.py**. One parameter in the `Particle` tuple that's confusing is `spin`. If `spin = 1` in the entry, the particle is a scalar (spin-0). If `spin = 2`, it's spin-½. Or if it's `spin = 3`, it's spin-1. I assume these correspond to the number of spin projections the particle possesses. For s-channel, the Z' is spin-1 and the dark quarks are spin-0 by default. For t-channel, the Φ mediators are spin-0 and the dark quarks are spin-½.

All particles – from SM as well as new particles for the model – are defined in **particles.py**. Some more details on the different particles (for example, what Xr, Xc, Xd mean) can be found at http://feynrules.irmp.ucl.ac.be/wiki/DMsimp.

The current plan is to use MadGraph with Pythia and a detector simulation to generate some samples, then compare some distributions to the Pythia-only samples Giorgia worked on. We need to make sure the parameters (namely α_{D}, $m_{Z'}$ and r_{inv}) are the same and the distributions match before switching to sole use of MadGraph+Pythia. Then we can create gridpacks and, following approval from the EXO convenors, request central production of the samples covering a range of mass points and coupling strengths, etc. Once we have them, we can begin some analysis.

For central production, a user needs to submit CMSSW config files detailing what to do and how to handle output. If the goal is to start from generator level, it raises a problem. LHE generators like MadGraph can't be called from within CMSSW, usually. The solution is to use gridpacks. Essentially, a gridpack is a tarball of the generator program, and the input files needed for generating the LHE file. In my case, a gridpack would be composed of the MadGraph release I'm using over a semi-visible jets model with a certain combination of parameters, and some metadata for CMSSW.

The first step is to be able to create gridpacks to run MadGraph with the semi-visible jets models and then run those on a grid; I'll probably start off using Condor at lxplus, then try CRAB submission later on. I've followed the instructions at https://twiki.cern.ch/twiki/bin/view/CMS/QuickGuideMadGraph5aMCatNLO.

In terms of input files, I was able to use the proc cards from when I was running MadGraph locally. Then, in the output directories, I pulled the run cards and slightly tweaked those. I also had to zip the model files using `tar -cf` and write the "extramodels" cards that linked to them, then request to upload the zips to the generator web repository on **/afs/cern.ch/cms/generators/www/** on lxplus. When the relevant scripts run, the extramodels file is checked, and the zip file name is looked for in that directory in order to pull the correct model and insert it into the MadGraph release it uses. All the cards and models are stored in the *SemivisibleJets* repo. After cloning the *genproductions* repo (https://github.com/cms-sw/genproductions) outside of my repo with

```
git clone git@github.com:eshwen/genproductions.git
    genproductions -b mg26x
```

that contained the code needed to generate gridpacks with a slightly modified version of MadGraph v2.6.0 (and various fixes I've implemented), I could run the validator tool to check my input cards were okay:

```
cd genproductions/bin/MadGraph5_aMCatNLO/Utilities/
    parsing_code
python parsing.py <path to process card directory/name of
    process card without _proc_card.dat>
```

Once validated, I could run gridpack generation with

```
cd genproductions/bin/MadGraph5_aMCatNLO/
./gridpack_generation.sh <name of process card without
   _proc_card.dat> <relative path to cards directory> <queue
    selection>
```

In the "queue selection" option, I can list all the queues with `bqueues`. Usually, `1nd` (wall time of 1 day), `1nw` (wall time of 1 week) or `grid_cms` will suffice. However, if I want to submit via Condor, I can instead type

```
./submit_condor_gridpack_generation.sh <name of process card
    without _proc_card.dat> <relative path to cards
   directory>
```

The architecture, CMSSW and MadGraph versions to be used are specified in the main script, but can be changed if needed. The name of zipped model files and the input cards must all have the prefix <model name> and the correct suffixes, and so is a bit rigid in that respect. Failures in executing may be down to that, so it's worth double checking. These instructions should also be listed in the README of *SemivisibleJets*.

I came across a couple of problems when trying to run the script. In **gridpack_generation.sh**, I had to change the model finder (∼ L187) to

```
model=`sed 's:#.*$::g' $CARDSDIR/${name}_extramodels.dat |
   grep -E '.zip|.tar|.tgz' $CARDSDIR/${name}_extramodels.
   dat`
```

and comment out the `do` and `done` after it. I also had to change the PDGIDs of some of the dark particles as suggested by the theorists who provided the model. The Hidden Valley (HV) module in Pythia (http://home.thep.lu.se/Pythia/pythia82html/HiddenValleyProcesses.html) specifies new particles that are part of the gauge group introduced. As the PDGIDs of the dark particles in the model are effectively dummy, they need to be changed to correspond to a particle recognised by HV so they can be showered properly. In the *genproductions* repo, I had to specify the following lines in **runcmsgrid_LO.sh** and **runcmsgrid_NLO.sh**:

```
1 echo "******** CHANGING PARTICLE IDS FOR PYTHIA SHOWERING (s
     -channel) ********"
  sed -i 's/5000521/4900101/g' cmsgrid_final.lhe
3
  echo "******** CHANGING PARTICLE IDS FOR PYTHIA SHOWERING (t
     -channel) ********"
```

```
sed -i 's/49001010/4900101/g' cmsgrid_final.lhe
sed -i 's/49001011/4900101/g' cmsgrid_final.lhe
sed -i 's/49001012/4900101/g' cmsgrid_final.lhe
sed -i 's/49001013/4900101/g' cmsgrid_final.lhe
sed -i 's/49001014/4900101/g' cmsgrid_final.lhe
```

Depending on whether I ran MadGraph at LO or NLO, one of those two scripts are copied into the gridpack and run after the LHE file is generated by CMSSW. I couldn't add support to distinguish the two models so fewer PIDs are changed, but they don't overlap between models so it doesn't really matter.

In the output, I can view the Feynman diagrams by looking in **SubProcesses/<subprocess>/**. There are several ps and jpeg files that contain the diagrams.

The main parameters of note when generating these gridpacks are

CMSSW version: 7_1_30

MadGraph version: v2.6.0 (with slight tweaks by CMS Generators group)

PDF used: NNPDF3.0 NLO for 2016 production emulation (with LHAPDF v6.2.1 evaluator, LHAID 292000, see https://lhapdf.hepforge.org/index.html)

Interaction order: Leading Order

35.4 FullSim in CMSSW on gridpacks for Pythia hadronisation and detector simulation

Once the gridpacks have been generated, it is possible to take them through the entire CMSSW chain (from hadronising with Pythia to doing a complete detector simulation with GEANT4 [67] and producing nanoAODs). This is done within CMSSW through the use of `cmsDriver.py` and `cmsRun`. Minimal input and command line arguments are required as CMSSW takes care of a lot of the backend. The first thing to do is create a "GEN fragment", which is a Python script detailing the input parameters and some other useful information to initially feed into CMSSW. As I used an external generator to get my gridpacks, I need to add the code below about the `externalLHEProducer` to the fragment. The first step in the CMSSW chain is to run the gridpack and extract the LHE file. Specifying `generator` as `Pythia8HadronizerFilter` as well allows showering to be done in the same step to get the GEN-SIM root file.

```
import FWCore.ParameterSet.Config as cms
```

```
from Configuration.Generator.Pythia8CommonSettings_cfi
    import *
from Configuration.Generator.Pythia8CUEP8M1Settings_cfi
    import *
from Configuration.Generator.Pythia8aMCatNLOSettings_cfi
    import *

# Needed as I'm using an external generator
externalLHEProducer = cms.EDProducer("ExternalLHEProducer",
     args = cms.vstring('/afs/cern.ch/work/e/ebhal/public/
    DMsimp_SVJ_s_spin1_slc6_amd64_gcc481_CMSSW_7_1_30_tarball
    .tar.xz'),
     nEvents = cms.untracked.uint32(50000),
     numberOfParameters = cms.uint32(1),
     outputFile = cms.string('cmsgrid_final.lhe'),
     scriptName = cms.FileInPath('GeneratorInterface/
    LHEInterface/data/run_generic_tarball_cvmfs.sh')
)

from Configuration.Generator.Pythia8CommonSettings_cfi
    import *
from Configuration.Generator.Pythia8CUEP8M1Settings_cfi
    import *
from Configuration.Generator.Pythia8aMCatNLOSettings_cfi
    import *

generator = cms.EDFilter("Pythia8HadronizerFilter",
     maxEventsToPrint = cms.untracked.int32(1),
     pythiaPylistVerbosity = cms.untracked.int32(1),
     filterEfficiency = cms.untracked.double(1.0),
     pythiaHepMCVerbosity = cms.untracked.bool(False),
     crossSection = cms.untracked.double(<x_sec>),
     comEnergy = cms.double(13000.),

     PythiaParameters = cms.PSet(
```

```
        pythia8CommonSettingsBlock,
29      pythia8CUEP8M1SettingsBlock,
        pythia8aMCatNLOSettingsBlock,
31      JetMatchingParameters = cms.vstring(
            'JetMatching:setMad = off',
33          'JetMatching:scheme = 1',
            'JetMatching:merge = on',
35          'JetMatching:jetAlgorithm = 2',
            'JetMatching:etaJetMax = 5.',
37          'JetMatching:coneRadius = 1.',
            'JetMatching:slowJetPower = 1',
39          'JetMatching:qCut = 100.', #this is the actual
    merging scale
            'JetMatching:nJetMax = 2', #number of partons in
     born matrix element for highest multiplicity
41          'JetMatching:doShowerKt = off', #off for MLM
    matching, turn on for shower-kT matching
            ),
43
        processParameters = cms.vstring(
45          '4900111:m0 = <m_dark_meson>', # Dark meson mass
    . Should be twice the dark quark mass
            '4900211:m0 = <m_dark_stable>', # Stable dark
    particle mass. Can set as dark quark mass - 0.1 geV
47          '4900111: oneChannel = 1 <r_inv> 4900211
    -4900211', # Dark meson decay into stable dark particles
    with branching fraction r_inv
            '4900111: addChannel = 1 <1 - r_inv> 91 1 -1', #
     Dark meson decay into down quarks with branching
    fraction 1 - r_inv
49          #'TimeShower:nPartonsInBorn = 2', #number of
    coloured particles (before resonance decays) in born
    matrix element
            'HiddenValley:ffbar2Zv = on', #it works only in
    the case of narrow width approx
```

```
            'HiddenValley:fragment = on', # enable hidden valley fragmentation
            #'HiddenValley:NBFlavRun = 0', # number of bosonic flavor for running
            #'HiddenValley:NFFlavRun = 2', # number of fermionic flavor for running
            'HiddenValley:alphaOrder = 1', # order at which running coupling runs
            'HiddenValley:Lambda = <Lambda_d>, # parameter used for running coupling
            'HiddenValley:nFlav = <N_f>', # this dictates what kind of hadrons come out of the shower, if nFlav = 2, for example, there will be many different flavor of hadrons
            'HiddenValley:probVector = 0.75', # ratio of number of vector mesons over scalar meson, 3:1 is from naive degrees of freedom counting
            'HiddenValley:pTminFSR = <1.1*Lambda_d>', # cutoff for the showering, should be roughly confinement scale
            ),

        parameterSets = cms.vstring('pythia8CommonSettings',
                                    'pythia8CUEP8M1Settings',
                                    'pythia8aMCatNLOSettings',
                                    'processParameters',
                                    'JetMatchingParameters'
                                    )
    )
)
```

I got the `processParameters` commands (required for the Hidden Valley module in Pythia) from Giorgia's repo and then modified them quite a bit as I understood more about the model. The important things to note are the Pythia imports; the variable `args` which gives the

path to the gridpack tarball; the variable `outputFile` which gives the path to the output LHE file from the generator, which needs to be fixed as the **runcmsgrid.sh** script that's placed in the gridpack requires a specific name; the variable `scriptName` which gives the name of the shell script to run, the options being detailed in https://github.com/cms-sw/cmssw/tree/master/GeneratorInterface/LHEInterface/data. These scripts are part of the base CMSSW, so when a release is sourced, the path provided in `scriptName` should be sufficient to find the relevant file.

In terms of technically implementing r_{inv}, I added a dark meson particle with PDGID 4900111 which should be twice the dark quark mass. I also added a stable dark matter particle 4900211 which has a mass of 0.1 GeV below the dark quark so that it's stable and doesn't get stuck in an endless loop of hadronising and decaying. Then the dark meson can decay either into these stable dark particles (with a branching fraction of r_{inv}) or to SM quarks with the remaining fraction. These stable dark particles can also be replaced by neutrinos, but it doesn't really matter as long as they're invisible and leave $E_{\text{T}}^{\text{miss}}$ behind.

There are a few steps to get from a gridpack to nanoAOD:

1. Gridpack → LHE
2. LHE → GEN-SIM (showering happens here)
3. GEN-SIM → AODSIM (step 1)
4. AODSIM (step 1) → AODSIM (step 2)
5. AODSIM (step 2) → MINIAODSIM
6. MINIAODSIM → NANOAODSIM

Once the GEN fragment had been written, I had to run the commands below and clone specific CMSSW releases for the different steps. The options for the cmsDriver commands were based off the dataset(s)

```
/Axial_MonoJ_NLO_Mphi-100_Mchi-1_gSM-0p25_gDM-1p0_13TeV-
    madgraph/RunIISummer15wmLHEGS-MCRUN2_71_V1-v1/GEN-SIM

/Axial_MonoJ_NLO_Mphi-100_Mchi-1_gSM-0p25_gDM-1p0_13TeV-
    madgraph/RunIISummer16DR80Premix-
    PUMoriond17_80X_mcRun2_asymptotic_2016_TrancheIV_v6-v1/
    AODSIM

/Axial_MonoJ_NLO_Mphi-100_Mchi-1_gSM-0p25_gDM-1p0_13TeV-
    madgraph/RunIISummer16MiniAODv2-
```

```
    PUMoriond17_80X_mcRun2_asymptotic_2016_TrancheIV_v6-v1/
    MINIAODSIM

/Axial_MonoJ_NLO_Mphi-100_Mchi-1_gSM-0p25_gDM-1p0_13TeV-
    madgraph/RunIISummer16NanoAOD-
    PUMoriond17_05Feb2018_94X_mcRun2_asymptotic_v2-v1/*
```

which is emulating 2016 MC (making the `cmsDriver` commands specific to that configuration). By searching the dataset on DAS, I could click on "dbs3 show" for each step in the production chain and copy the "prep_id". Then, I went to McM (the Monte Carlo request Management, https://cms-pdmv.cern.ch/mcm/) → Request → Navigation and put the prep_id in the prepid field. After clicking Search, the dataset should appear, and under Actions, there should be an icon corresponding to "Get setup command". Clicking on that reveals the cmsDriver commands used to generate that step of the production.

35.4.1 Gridpack to LHE-GEN-SIM

```
source /cvmfs/cms.cern.ch/cmsset_default.sh
export SCRAM_ARCH=slc6_amd64_gcc481
cmsrel CMSSW_7_1_30 # Includes Pythia 8.226 for Hidden
    Valley
cd CMSSW_7_1_30/src
cmsenv
mkdir -p Configuration/GenProduction/python
cp <GEN fragment> Configuration/GenProduction/python/
scram b
cmsenv
```

I had to move the GEN fragment to that folder, and compile with `scram b` and `cmsenv` again to make sure the right directories are added to `PYTHONPATH`. Once completed, I could run `cmsDriver.py` from **$CMSSW_BASE/src** to generate the config required by CMSSW:

```
cmsDriver.py Configuration/GenProduction/python/
    SVJ_MadGraph_LHAPDF_13TeV_s_channel_spin1_GEN_frag.py --
    fileout file:SVJ_s_LHE_GEN_SIM.root --mc --eventcontent
    RAWSIM,LHE --customise SLHCUpgradeSimulations/
    Configuration/postLS1Customs.customisePostLS1,
    Configuration/DataProcessing/Utils.addMonitoring --
```

```
datatier GEN-SIM,LHE --conditions MCRUN2_71_V1::All --
beamspot Realistic50ns13TeVCollision -s LHE,GEN,SIM --
magField 38T_PostLS1 --python_filename SVJ_s_LHE_GEN_SIM.
py --no_exec -n 250
```

The config should appear in the current directory (although a path and filename can be specified in the command above with the option `-python_filename=<path>`). Many of the options will be specific to the CMSSW release used and the year of data taking that's being emulated. The `-n` option refers to the number of events to run over. If it's more than the number of events in the dataset, it doesn't matter. Then, it's as simple as running

```
cmsRun SVJ_s_LHE_GEN_SIM.py -n <no. threads>
```

which should give an output root file (although looking at it isn't really useful at this stage). For reference, I used the same configs and cmsDriver commands for both the s- and t-channel models (bar the gridpack- and output-specific names).

35.4.2 GEN-SIM to AOD (step 1)

This requires a different CMSSW version, as the original GEN-SIM files are being emulated for the 2016 run.

```
source /cvmfs/cms.cern.ch/cmsset_default.sh
export SCRAM_ARCH=slc6_amd64_gcc530
cmsrel CMSSW_8_0_21
cd CMSSW_8_0_21/src
cmsenv
cp <path>/SVJ_s_LHE_GEN_SIM.root .
scram b
cmsenv
voms-proxy-init --voms cms --valid 168:00 # for querying PU
   dataset
```

From this point, the only input needed by `cmsDriver.py` is the output root file from the previous step. In this step, the pileup mixing is performed. It is needed for accurate replication of the conditions for the year of data taking that's trying to be emulated. Normally, the pileup input would be an entire dataset. But querying it for the the dataset below is unfeasible so I just queried one file.

```
cmsDriver.py step1 --filein file:SVJ_s_LHE_GEN_SIM.root --
   fileout file:SVJ_s_AOD_step1.root --pileup_input /store/
```

```
   mc/RunIISpring15PrePremix/Neutrino_E-10_gun/GEN-SIM-DIGI-
   RAW/PUMoriond17_80X_mcRun2_asymptotic_2016_TrancheIV_v2-
   v2/100000/001EB167-3781-E611-BE3C-0CC47A4D75F4.root --mc
   --eventcontent PREMIXRAW --datatier GEN-SIM-RAW --
   conditions 80X_mcRun2_asymptotic_2016_TrancheIV_v6 --step
    DIGIPREMIX_S2,DATAMIX,L1,DIGI2RAW,HLT:@frozen2016 --
   datamix PreMix --era Run2_2016 --python_filename
   SVJ_s_AOD_step1.py --no_exec --customise Configuration/
   DataProcessing/Utils.addMonitoring -n 250

cmsRun SVJ_s_AOD_step1.py -n 8
```

The dataset I've seen many people use for the pileup input is

```
/Neutrino_E-10_gun/RunIISpring15PrePremix-
   PUMoriond17_80X_mcRun2_asymptotic_2016_TrancheIV_v2-v2/
   GEN-SIM-DIGI-RAW
```

Giorgia specified many files from this dataset in a **pileup_filelist.txt**, which I could also specify if I want to.

35.4.3 AOD (step 1) to AOD (step 2)

This step is done in the same CSSW release with the same environment as the previous step.

```
cmsDriver.py step2 --filein file:SVJ_s_AOD_step1.root --
   fileout file:SVJ_s_AOD_step2.root --mc --eventcontent
   AODSIM --runUnscheduled --datatier AODSIM --conditions 80
   X_mcRun2_asymptotic_2016_TrancheIV_v6 --step RAW2DIGI,
   RECO,EI --era Run2_2016 --python_filename SVJ_s_AOD_step2
   .py --no_exec --customise Configuration/DataProcessing/
   Utils.addMonitoring -n 250

cmsRun SVJ_s_AOD_step2.py -n 8
```

35.4.4 AOD (step 2) to miniAOD

This is step is done in the same CMSSW release with the same environment as the previous step.

```
cmsDriver.py --filein file:SVJ_s_AOD_step2.root --fileout
   file:SVJ_s_MINIAOD.root --mc --eventcontent MINIAODSIM --
   runUnscheduled --datatier MINIAODSIM --conditions 80
   X_mcRun2_asymptotic_2016_TrancheIV_v6 --step PAT --era
   Run2_2016 --python_filename SVJ_s_MINIAOD.py --no_exec --
   customise Configuration/DataProcessing/Utils.
   addMonitoring -n 250

cmsRun SVJ_s_MINIAOD.py
```

35.4.5 MiniAOD to nanoAOD

This is a relatively new step in the CMSSW chain. NanoAODs are supposed to resemble Heppy flat trees, which makes them easy to read, and only require an extra command from `cmsDriver.py` and `cmsRun`. But as backward and forward compatibility between CMSSW releases can be an issue, a newer version that supports nanoAOD creation is needed:

```
source /cvmfs/cms.cern.ch/cmsset_default.sh
export SCRAM_ARCH=slc6_amd64_gcc630
cmsrel CMSSW_9_4_4
cd CMSSW_9_4_4/src
cmsenv
cp <path>/SVJ_s_MINIAOD.root .
scram b
cmsenv
```

And then the main commands are

```
cmsDriver.py --filein file:SVJ_s_MINIAOD.root --fileout file
   :SVJ_s_NANOAOD.root --mc --eventcontent NANOAODSIM --
   datatier NANOAODSIM --conditions auto:run2_mc -s NANO --
   era Run2_2016,run2_miniAOD_80XLegacy --python_filename
   SVJ_s_NANOAOD.py --no_exec -n 250

cmsRun SVJ_s_NANOAOD.py -n 8
```

Then, a nanoAOD file should be created. Inspecting it should reveal several trees, but the only interesting one is "Events". Inside, are a list of easy-to-read branches, which should make analysis much easier than with miniAODs.

35.5 Running the FullSim chain with CRAB

When processing a sample with many events, using a grid or batch system becomes a necessity. For the GEN-SIM step in particular, the hadronisation and detector simulation can take up to a minute per event. CRAB is probably the best job manager to use as it only requires a config file that specifies the CMSSW config to use – as CRAB executes `cmsRun` by default, although other scripts/commands can be run if necessary – and how to handle the jobs and splitting. Using CRAB also allows for easy resubmission of any failed jobs. Some handy guides are located at https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideCrab and https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookCRAB3Tutorial.

For a full CMSSW chain, incorporating CRAB doesn't require much extra work: the CMSSW config is created from the gen fragment using the same `cmsDriver` command and CMSSW version as when running locally, but instead of `cmsRun`, one specifies the path to the CMSSW config in the CRAB config file, and the CRAB config is submitted to the grid, which handles everything. The only caveats I've discovered so far are that CRAB handles the path to the input gridpack weirdly, and the LHE-GEN-SIM step should be split into gridpack to LHE, and LHE to GEN-SIM steps. The first point basically means that in the first gen fragment that only specifies `externalLHEProducer`, instead of giving the absolute (or even relative) path to the gridpack, use `../<basename of gridpack>` and create the CMSSW config. Then, in the CRAB config, specify the absolute path to the gridpack with the `config.JobType.inputFiles` variable. The gridpack should be placed in a public folder on whatever remote server one is working from. On **/afs** on lxplus, the permissions of a folder can be changed with

```
fs sa <path to folder> system:anyuser rl
```

which gives any user read access. This is important as the CRAB nodes need to have access to the gridpack in order to upload it to the sandbox tied to the job. The sandbox does have a size limit of 100 MB, so one must be aware of that before running on CRAB. For central production, gridpacks can be placed on **/cvmfs** and this path rannygazoo isn't an issue. But for private production, this is not possible. The second point is really an optimisation. When running the gridpack to LHE step, it is not advisable to split it into multiple jobs for RNG reasons. Whilst one can run GEN-SIM in the same step, it will all be processed in a single job so could take a while. It is easier to do the LHE step first, and then split it into multiple jobs when running GEN-SIM.

35.5.1 Gridpack to LHE on CRAB

As before, all my files and scripts needed to run these steps should be detailed on my *SemivisibleJets* repo. The gen fragment looks very similar to when running locally, but contains the change to the

gridpack path I mentioned above:

```
import FWCore.ParameterSet.Config as cms

# Needed as I'm using an external generator
externalLHEProducer = cms.EDProducer("ExternalLHEProducer",
    args = cms.vstring('../
   DMsimp_SVJ_s_spin1_slc6_amd64_gcc481_CMSSW_7_1_30_tarball
   .tar.xz'),
    nEvents = cms.untracked.uint32(20000),
    numberOfParameters = cms.uint32(1),
    outputFile = cms.string('cmsgrid_final.lhe'),
    scriptName = cms.FileInPath('GeneratorInterface/
   LHEInterface/data/run_generic_tarball_cvmfs.sh')
)
```

using the following command to make the CMSSW config:

```
cmsDriver.py Configuration/GenProduction/python/
   SVJ_MadGraph_NNPDF30_13TeV_s_spin1_LHE_frag.py --fileout
   file:DMsimp_SVJ_s_MadGraph_NNPDF30_13TeV_LHE.root --mc --
   eventcontent LHE --datatier LHE --conditions MCRUN2_71_V1
   ::All -s LHE --python_filename
   DMsimp_SVJ_s_MadGraph_NNPDF_13TeV_LHE.py --no_exec --
   customise Configuration/DataProcessing/Utils.
   addMonitoring -n 20000
```

Note for this step that the output file can be an LHE file. Then the CRAB config, written in Python with a **.py** extension, looks like this:

```
from CRABClient.UserUtilities import config,
   getUsernameFromSiteDB
config = config()

# Not mandatory, just for simplicity in constructing
   directory/dataset names
modelName = 'DMsimp_SVJ_s_spin1_MadGraph'
datasetStr = 'mZp_1000_md_10_alphaD_0p1_NNPDF30_13TeV-LHE'
```

```
# CRAB project directory
config.General.workArea = modelName
# Directory below top level output directory
config.General.requestName = datasetStr
config.General.transferOutputs = True
config.General.transferLogs = True

config.JobType.pluginName = 'PrivateMC'
# Name of the CMSSW configuration file
config.JobType.psetName = '
   DMsimp_SVJ_s_MadGraph_NNPDF_13TeV_LHE.py'
# Path to the gridpack (directory must have public read
   permissions)
config.JobType.inputFiles = ['/afs/cern.ch/work/e/ebhal/
   public/
   DMsimp_SVJ_s_spin1_slc6_amd64_gcc481_CMSSW_7_1_30_tarball
   .tar.xz']

# This string determines the primary dataset of the newly-
   produced output
config.Data.outputPrimaryDataset = datasetStr
#config.Data.inputDBS = 'global'
config.Data.splitting = 'EventBased'
config.Data.unitsPerJob = 20000 # Number of events per job (
   LHE must have 1 job due to RNG)
config.Data.totalUnits = 20000
config.Data.outLFNDirBase = '/store/user/%s/' % (
   getUsernameFromSiteDB())
config.Data.publication = True # If true, output files are
   published on DBS. Useful for future steps
# Directory below outputPrimaryDataset in output, also
   directory below workArea in project dir
config.Data.outputDatasetTag = modelName + '_' + datasetStr

config.Site.whitelist = ['T2_UK_SGrid_Bristol', 'T2_CH_CERN'
```

```
    ] # CERN site needed so CRAB worker nodes with /afs
    mounted can be used
config.Site.storageSite = 'T2_UK_SGrid_Bristol' # Site the
    output files will be transmitted to
```

Some useful options for the CRAB config can be found at https://twiki.cern.ch/twiki/bin/view/CMSPublic/CRAB3ConfigurationFile. Once the CMSSW config has been created from the gen fragment and the CRAB config is all set to go, one must source the CRAB environment with

```
source /cvmfs/cms.cern.ch/crab3/crab.sh
```

Then, it's as simple as running

```
crab submit -c <CRAB config>
```

You will be prompted for your grid certificate pass phrase at this point. I placed the CMSSW and CRAB configs in **$CMSSW_BASE/src**, but I don't think the path matters that much. One can do a dry run with the option `-dryrun`, and check on the jobs with

```
crab status
```

If there are multiple jobs that have been submitted from the same directory, do

```
crab status -d <CRAB project directory>
```

where the name of the CRAB project directory is specified in the config file and is usually placed in the submission directory. This command gives a brief log, but a complete one can be found by following the link indicated by "Dashboard monitoring URL". This is often much more useful when debugging. In the CRAB config, if `config.Data.publication = True`, the name of the output dataset will be shown when doing `crab status`. Make a note of this as it will be needed in the next step.

One thing I've noticed from running is that CRAB blacklists Bristol fairly often. My impression is that some tests are run, and if a site fails one or more tests they get blacklisted until the issues are resolved. One can check the status of a site, *i.e.*, Bristol, at https://cms-site-readiness.web.cern.ch/cms-site-readiness/SiteReadiness/HTML/SiteReadinessReport.html#T2_UK_SGrid_Bristol.

35.5.2 LHE to GEN-SIM on CRAB

Once we have the output from the previous step, the GEN-SIM step can be completed. Again, the CMSSW config needs to be created from the gen fragment, this time only specifying the hadroniser settings. The `cmsDriver` command I used was

```
cmsDriver.py Configuration/GenProduction/python/
   SVJ_MadGraph_NNPDF30_13TeV_s_spin1_GS_frag.py --filein
   file:DMsimp_SVJ_s_MadGraph_NNPDF30_13TeV_LHE.root --
   fileout file:DMsimp_SVJ_s_MadGraph_NNPDF30_13TeV_GS.root
   --mc --eventcontent RAWSIM --customise
   SLHCUpgradeSimulations/Configuration/postLS1Customs.
   customisePostLS1,Configuration/DataProcessing/Utils.
   addMonitoring --datatier GEN-SIM --conditions
   MCRUN2_71_V1::All --beamspot Realistic50ns13TeVCollision
   -s GEN,SIM --magField 38T_PostLS1 --python_filename
   DMsimp_SVJ_s_MadGraph_NNPDF_13TeV_GS.py --no_exec -n
   20000
```

noting that the argument to `-filein` needs to match the one from `-fileout` from the previous step. In the LHE step, the file with that name is created in the output directory on the storage site specified in the CRAB config.

The CRAB config is similar to previously but contains some important changes:
The `inputFiles` and `outputPrimaryDataset` variables can be removed

Anything with "LHE" should be changed to "GEN-SIM"

The CMSSW config file must be updated

If `config.Data.publication = True` in the previous step's config, change `config.JobType.pluginName` to `'Analysis'`

The splitting can be chosen. If changed to `'EventAwareLumiBased'`, one only needs to specify `unitsPerJob`, then `config.Data.totalUnits` can be set to -1

The variable `config.Data.inputDataset` needs to be added. By doing `crab status` for the previous job, assuming `publication = True`, the output dataset name will be shown. This is the input dataset for this step

The variable `config.Data.inputDBS = 'phys03'` needs to be added
Once changed, the job(s) can be submitted.

35.5.3 GEN-SIM to AOD (step 1) on CRAB

This step is pretty simple. I needed to clone CMSSW_8_0_21 and make the CMSSW config with

```
cmsDriver.py step1 --filein file:
   DMsimp_SVJ_s_MadGraph_NNPDF30_13TeV_GS.root --fileout
```

```
file:DMsimp_SVJ_s_MadGraph_NNPDF30_13TeV_AOD_step1.root
--pileup_input filelist:~/Semi-visible_jets/
SVJ_production/global/pileup_filelist.txt --mc --
eventcontent PREMIXRAW --datatier GEN-SIM-RAW --
conditions 80X_mcRun2_asymptotic_2016_TrancheIV_v6 --step
 DIGIPREMIX_S2,DATAMIX,L1,DIGI2RAW,HLT:@frozen2016 --
datamix PreMix --era Run2_2016 --python_filename
DMsimp_SVJ_s_MadGraph_NNPDF_13TeV_AOD_step1.py --no_exec
--customise Configuration/DataProcessing/Utils.
addMonitoring -n 50000
```

where I specified Giorgia's pileup input file list from her *SVJ_production* repo. This contains a few files from the Neutrino Gun dataset I've seen many uses of.

The CRAB config is basically the same as last time, but all instances of "GEN-SIM" were replaced by "AOD_step1" and the dataset was updated to the output from the GEN-SIM step.

35.6 FullSim chain on Condor

Whilst CRAB is an easier submission tool and is better overall, the server often blacklists Bristol for days at a time which can halt progress. So for these troubled times, I developed a FullSim version that runs on Condor. The only caveat is that the LHE-GEN-SIM step doesn't work with Condor. A gridpack can be passed to the jobs, but RNG means that identical files would be created for each job. I edited a script I found from someone to split LHE files, so I just needed to untar the gridpack I had and run

```
./runcmsgrid.sh <n_events> <random seed>
```

to generate the LHE file. Then, I could run the splitting script to get individual LHE files and associate one file to a job. All the scripts and instructions are detailed on the README in my *SemivisibleJets* fork. The output nanoAODs can be combined using the **haddnano.py** script I borrowed from the *nanoAOD-tools* repository (https://github.com/cms-nanoAOD/nanoAOD-tools).

As I've been doing a lot of code development and changing a lot of existing code, it can be tough to remember what I've changed since the last-pushed version. If I do `git diff <file>`, I can see my local changes compared to the online version.

35.7 Extending the analysis and working with colleagues from Fermilab/Rochester (19/03/2018)

We have been contact by CMS experimentalists at Fermilab/Rochester. They've already made private samples with a parameter scan and have looked into discriminating variables for the analysis aspect. They've used Pythia for their s-channel generation as the theorists have stated that both Pythia and MadGraph use the same matrix element for the hard scatter. Although, they agreed that MadGraph would probably be necessary for the t-channel process (where the protons exchange the bi-fundamental mediator Φ). They are interested in collaborating with me, Ben, Annapaola and Giorgia, hoping to develop an analysis by the end of this year or early next year. They also want to look at making signal samples to emulate, and therefore compare to, 2016 and 2017 data with legacy reprocessing with CMSSW_9_4_X.

We had an initial meeting where everyone detailed their progress so far: https://indico.cern.ch/event/714981/. A twiki page as also set up: https://twiki.cern.ch/twiki/bin/viewauth/Main/SemiVisibleJet. I gave an update at the following meeting: The main feedback was that it was good I was looking into using MadGraph and the t-channel model, and that's probably where I can slot in in terms of sample production. Kevin Pedro suggested using his repo https://github.com/kpedro88/SVJProduction (which uses the batch submission repo https://github.com/kpedro88/CondorProduction) and building on that so that we have a common production framework. I also presented a condensed version of the talk at a Bristol CMS meeting: The main feedback from that was to understand the differences between s- and t-channel (t-channel has several similar mediators, s-channel has one, list the cross sections for the channels, include only the leading jet p_{T} in the jet p_{T} plots, understand the n_{jet} and $E_{\mathrm{T}}^{\mathrm{miss}}$ distributions for t-channel).

Kevin gave an initial presentation at an EXO MET+X meeting that outlines the model, sample generation and analysis proposal, then a more refined version during CMS Week: https://indico.cern.ch/event/722583/contributions/2971261/attachments/1635868/2609895/svj_overview_apr_18_2018.pdf

35.7.1 Comparisons between Fermilab's Pythia-only and my MadGraph + Pythia samples

Once I got my sample production to a good-enough state and fixed any bugs/optimised configurations, I could compare my samples (generated with MadGraph and hadronised with Pythia) to those from the Fermilab guys (generated with Pythia). Kevin linked me to his ntuples at root://cmseos.fnal.gov//store/user/lpcsusyhad/SVJ2017/ProductionV2/MINIAOD/.

But I needed to be more specific if I wanted to load the samples themselves. If I enable a proxy of my grid certificate, I can log in to the eos server with

```
eos root://cmseos.fnal.gov/
```

Then I could navigate through the directories like any other server, like

```
cd /store/user/lpcsusyhad/SVJ2017/ProductionV2/MINIAOD
ls
```

The xrootd redirector is also specified, so gives me all the information I need to access the samples. One thing to note is that their m_d is the mass of their dark *meson*, where mine is the mass of the dark *quark*. Note: the current up-to-date samples are stored in `/store/user/lpcsusyhad/SVJ2017/ProductionV3/2017/MINIAOD/`.

I also needed to iterate over the `qCut` (Pythia)/`xqcut` (MadGraph). These are cuts placed on the k_{T} of a particle during generation/hadronisation. While k_{T} usually means transverse momentum, the way they're used is a bit different. The slides at http://hep.ucsb.edu/people/cag/Matching.pdf highlight the differences, and a nice, introductory overview of the concepts can be found in Ref. [68]. Kevin sent me some instructions on how to optimise the value for the qCut: https://github.com/CMS-SUS-XPAG/GenLHEfiles/tree/master/Run2Mechanism#determining-the-qcut. I just needed to produce GEN-only events at different values of the qCut and then make distributions to see what value is optimal. For the input, I could run my normal GEN-SIM step over the LHE files, but changing the option `-step GEN,SIM` to `-step GEN`. After the initial set up with the link above (and ensuring I made a **rootlogon.C** file in the working directory containing the code they specify), I could run the following commands to make the plots:

```
ssh -Y lxplus
cd /eos/user/e/ebhal/CMSSW_7_1_30/prod/GenLHEfiles/
   Run2Mechanism/genstep
cmsenv
root -l
.L plotdjr.C
plotdjr("<path to root files>","<output basename>") #
   wildcarding allowed for root files
```

35.8 Plots for the AEPSHEP 2018 summer school

I made a poster on semi-visible jets for the AEPSHEP summer school in Vietnam: . The two plots I showcased, which illustrate the kinematics of the signal for those unfamiliar with the model, are below:

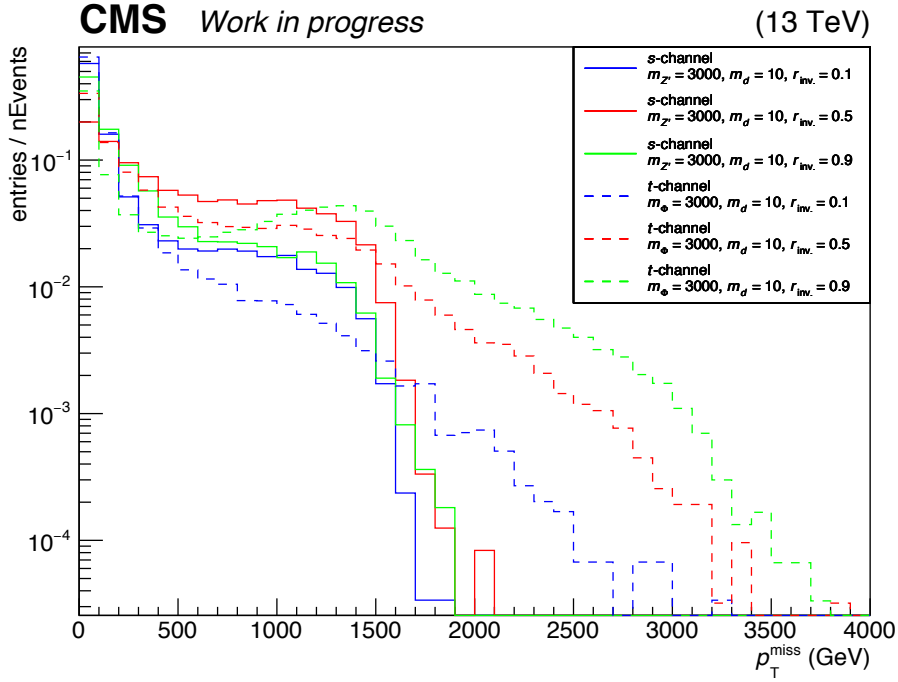

Figure 35.1: The $p_{\mathrm{T}}^{\mathrm{miss}}$ spectrum for the s-channel (solid lines) and t-channel (dashed lines) semi-visible jet models. All parameters are kept the same except r_{inv}, which is varied to demonstrate its effect on the kinematics.

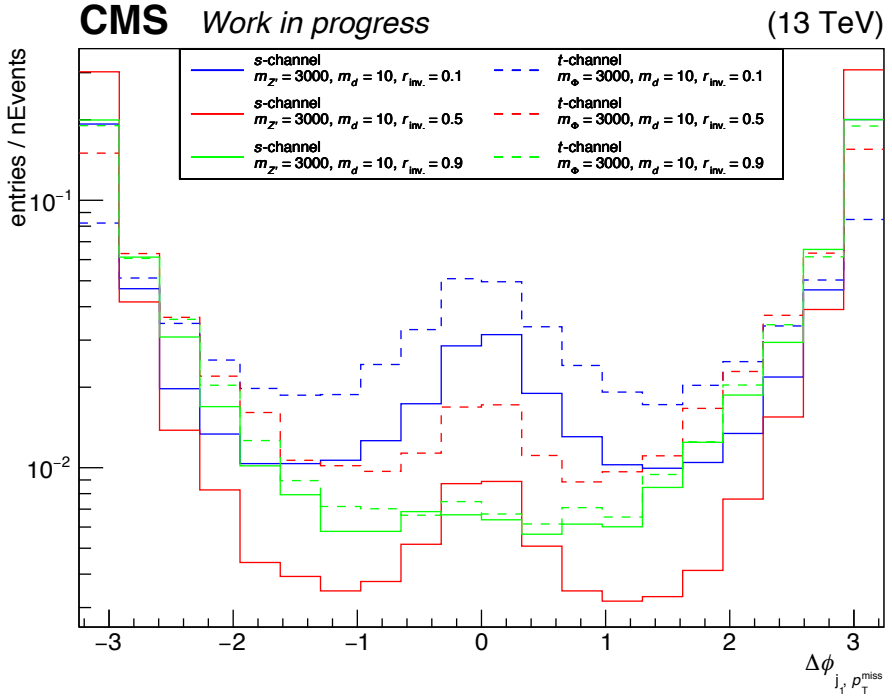

Figure 35.2: $\Delta\phi$ between the $p_{\mathrm{T}}^{\mathrm{miss}}$ and the leading jet for the s-channel (solid lines) and t-channel (dashed lines) semi-visible jet models. All parameters are kept the same except r_{inv}, which is varied to demonstrate its effect on the kinematics.

In the first plot, as expected, the s- and t-channel distributions look very different. For the t-channel, r_{inv} parameter affects the distributions as it should. A higher value yields more dark particles, and so more MET. The s-channel curves are less affected, however there is a sharp decline at roughly $m_{Z'}/2$. This is due to the nature of the portal. There's a resonant production of the Z', that decays into typically 2 jets. These should be symmetric in initial p_{T} and be produced back-to-back. So the maximum p_{T} of a SVJ (and therefore MET if all the particles remain invisible) will be $m_{Z'}/2$, hence the sharp decline afterward. If both jets have an invisible component (even if the total "dark" p_{T} is $> m_{Z'}/2$), the MET will be lower because it's formed from a *vector* sum. As the t-channel process is not a resonance and is just an exchange of the Φ mediator, its mass is less directly important. The jet p_{T} (and therefore MET) is not limited by the mass of the mediator. This is exhibited in the plot where there are events with $p_{\mathrm{T}}^{\mathrm{miss}} > m_\Phi$. While there are slight peaks in the t-channel curves for $r_{\mathrm{inv}} = 0.5, 0.9$ close to $m_\Phi/2$, we haven't figured out why, quite yet.

The second plot also shows what we expect. Having dijet events most of the time allows us to split this plot into three cases. Case 1 is where the MET is aligned with the leading jet, yielding $\Delta\phi \sim 0$. Case 2 is when the leading jet recoils from the MET (if say, the second jet retains a high invisible fraction), so $\Delta\phi \sim \pi$. Case 3 is everything else: each jet contains an invisible component (which may not be antipodal) and so the resultant MET points somewhere between the two jets.

After the conference, I also presented on SVJ for the Bristol student seminar:

We have an analysis note on GitLab: https://gitlab.cern.ch/tdr/notes/AN-19-061, with some general instructions on working with it(https://twiki.cern.ch/twiki/bin/viewauth/CMS/Internal/TdrProcessing).

N-subjettiness approach to identifying subjets: [69]. More on jet substructure: [70]. Running quark masses: [71]. Comprehensive paper on various DM MET+X searches: [72].

Chapter 36

FAST-RA1 (01/02/2018)

Over the last few months, Ben, Tai, Luke and Olivier (among others) have been working on a new framework to run data reduction and analysis. They called their team FAST (Faster Analysis Software Taskforce). Their first framework is FAST-RA1, which aims to replicate the RA1 analysis as validation to make sure the code works as it should. It is written in Python and relies mainly on pandas, NumPy, matplotlib and Tai's AlphaTwirl as dependencies. A tutorial was run on 1st Feb., and some PhD students in CMS (including myself) have been asked to help develop the code.

The slides shown can be found at . The repository is hosted on GitLab for privacy and integration with CERN services at `https://gitlab.cern.ch/fast-cms/FAST-RA1`, which I've forked. The documentation, which is automatically kept up-to-date with the code, is currently located at `https://fast-hep.web.cern.ch/fast-hep/cms/fast-ra1/`. There's also a README on the main GitLab page, which is a condensed version of the important aspects. However, it may not be up to date. All the command-line options for each of the steps are detailed in **bin/<executable>**.

Before cloning the repository, I had to generate new ssh keys for the remote servers I use; my Bristol email address is linked to GitHub, but GitLab requires my CERN account, so I couldn't just copy those SSH keys over. I could just follow the steps on `https://gitlab.cern.ch/help/ssh/README` to generate the key pairs and upload them to GitLab. As I'll be adding new SSH keys with potentially non-default filenames (e.g., **id_rsa_gitlab**), the instructions under "Working with non-default SSH key pair paths" are necessary. Then, to clone, there are multiple options I can choose from. If on lxplus, I could clone the URL under "KRB5", as lxplus has Kerberos authentication built in. But for Imperial and Soolin, I needed to clone the URL under "SSH". And so the submodules,

etc., are picked up, I needed to include the `-recursive` option when cloning.

As is good practice, I will develop code and make my own branches, etc. in my own fork of the repo. But as the framework is being developed by lots of people and at a fast pace, I need to keep up with commits and new features. I shouldn't develop code on the master branch of my fork. That should be reserved for the latest revision of the code. So, to get the latest changes from the master branch of the head fork (which is where the latest, stable version of the code should live), I can follow the instructions at this link: https://help.github.com/articles/syncing-a-fork/. Should be as simple as doing `git fetch <remote> <branch>; git pull <remote> <branch>`, assuming there are no conflicts (which I can check with a `git diff`). Then, I can merge any of my code into my newly-updated master branch to see how well it plays with the latest code.

36.1 Generating weighted cut flow tables

If I want to make cut flow tables, I really only need to run the `trees2dataframes` step, making sure the event selection is correct. I could also use AlphaTools to calculate weights by cloning it:

```
git clone --recursive git@github.com:CMSRA1/AlphaTools.git
    external/AlphaTools
```

Then I could specify the option `-alphatools` when running to include it and run the sequence. However, I had still had to make sure everything in AlphaTools was correct, like editing the base directory, location of the samples, adding the configuration file listing the samples, etc. But because I couldn't specify AlphaTools-like command-line arguments, I couldn't choose a pset (so didn't know which samples were being loaded). After some debugging, I realised that the samples loaded are from the `mcSRsamples` list. So if I added my collection of samples to that listed, they could be loaded. For accurate cut flow tables, I had to remove `skimSequence` from `sequence2016` and add the cut `ev : ev.failed_alphatools[0] == "False"` at the start of the cut flow.

As I did in *cutflowirl*, to run over event weights from AlphaTools for, e.g. weighted cut flow tables, I had to make some changes to the *alphatwirl* and *alphatwirl-interface* submodules. I added the relevant classes and made sure they were picked up by, and imported in, the right scripts in both submodules and FAST-RA1. I added the option in **trees2dataframes** to run over weights and get a weighted cut flow table out. The main annoyance was having to make commits in three repositories. In the submodules, I checked out new branches and committed there. Then, I could add the commit hash to my branch in FAST-RA1 (so that if I were to clone the repo at another site, everything would sync properly) with

```
git commit external -m "<message>"
```

```
git push <remote> <branch in FAST-RA1>
```

so that the commits in the submodules would be picked up by my branch in FAST-RA1. For running over event weights, I had to edit some stuff in AlphaTools so the right trees would be picked up and run over. I also had to add the `uncertainties` module as a requirement in FAST-RA1 for un-pickling the cross section pickle files. Then, I could run **trees2dataframes** with `-alphatools -weighted` to generate weighted cut flow tables.

36.2 Validating datacards produced by FAST-RA1

One of my first tasks in contributing to FAST-RA1 was to validate the datacards produced in FAST-RA1. I needed to run those, with the shapes root files (containing the TH2Ds, equivalent to when applying AlphaTools), through AlphaStats to check they were correct. We used the T2tt-4bd (250,170) mass point as a test. Olivier took the split tree and made datacards/shapes, and I used the files I already had when running the analysis for that model. At each step, I compared the files in the directories to make sure everything was in order.

I could run `makeCardsAndWs` over the FAST datacards, which ended up being the most painful step. AlphaStats is quite rigid in the way it handles the datacards, so Olivier and I had to iterate to make sure the formatting, etc. was exactly correct. I also had to copy the pickle files (that are usually made during the `optimiseBinning` step) from the T2tt-4bd datacards I already had to the new folders. After that step succeeded, I could run `runCombineTask`. Making the limit plots for a visual comparison didn't work as the interpolation/plotting failed due to the fact we only had one mass point. However, we could compare the limit values in the datacards for a numerical comparison. We could also make limit-per-bin plots as they only require a single mass point. I made some slides giving an update this task:

We soon realised that we would need more mass points in order to generate the limit plane and then compare plots, rather than numbers in the datacards. So Olivier decided to make the input for the entire T2tt-4bd mass plane. He got batch submission sorted at Bristol, meaning I had to copy the files over to Imperial with

```
rsync --exclude=\*~ -vuz -rltoD  --compress-level=9 /hdfs/
   user/od17981/FAST-RA1/<dir> ebhal@lx01.hep.ph.ic.ac.uk:"/
   vols/cms/RA1/80X/MC/$(date '+%Y%m%d')_FAST-
   RA1_datacards_T2tt-4bd/"
```

And then I had to copy the pickle files from *all* my T2tt-4bd datacard directories to each of the directories in these new FAST datacards. I could do this with

```
cd /vols/cms/RA1/80X/MC/20171026_T2tt_4bd/AlphaStats_Nominal
   /
for model in SMS-T2tt_mStop-*; do cp $model/mht*.pkl /vols/
   cms/RA1/80X/MC/<new_dir>/${model}/; done
```

In the main directory, I also needed the files **signalModels.txt** and **configuration.txt**. I made a GitHub gist for the complete instructions in case someone like Olivier wanted to reproduce the results: https://gist.github.com/eshwen/d6955bf1febfe28007376e836eb20668.

Chapter 37

Combined $H \rightarrow$ inv. analysis (25/05/2018)

37.1 Motivation

Whilst working on semi-visible jets, I am also involved in a combined Higgs to invisible analysis. This is "combined" in the sense that we will perform a Higgs $\rightarrow$ inv. search over all possible SM production modes in a single analysis. This is in contrast to the regular approach in which a separate analysis would be conducted for each channel and results would be combined at the end. Our strategy should give much better sensitivity than has previously been possible.

Despite this analysis seeming purely Standard Model-based, there is the possibility to explore new physics. The branching fraction for $H \rightarrow \nu\nu$ is predicted to be approximately 0.1% [73]. However, the current experimental limit on this value is 24% [74] (**update: now 19%** [75]). If the Higgs couples to new, exotic particles, this branching ratio should increase as the coupling is related to the mass of the particle. If we are able to close the gap between the theoretical and experimental values, it may prove to be a powerful constraint on new physics models. Once the analysis has been finalised, new models can be considered such as those involving dark matter.

In the SM, the Higgs can only decay to neutrinos via $H \rightarrow ZZ \rightarrow 4\nu$, with a predicted branching ratio of 1.06×10^{-3} [73]. It may also decay directly to neutrinos ($H \rightarrow \nu\nu$) if neutrinos acquire their mass from the Higgs. Although, as neutrino masses (if they do indeed have mass) are very small, this decay would be suppressed.

37.2 Models and sample production

A subset of the Higgs production modes we are considering are gluon-gluon fusion (ggF), top pair production with a Higgs (ttH), Higgs production in association with a vector boson (VH), and vector boson fusion (VBF). Our first step is to analyse samples from each of these processes and attempt to construct orthogonal signal regions to cover the largest phase space possible. An analysis will then be performed to extract the limit on the invisible (neutrino) branching fraction. As these processes are Standard Model and not BSM, this is a simpler undertaking in the sense it does not involve parameter scans or any other nuisances that arises from those models.

My current task is to investigate ttH sample production, binning it, and exploring an event selection and discriminating variables. In the future, it may develop into analysing dark matter models in this context, if we decide on pursuing BSM physics. For all of the modes above, we plan to use the centrally-produced nanoAOD MC However, for ttH, they haven't been produced yet as of 30/05/2018 (but the request has gone in, so it should be available soon). The dataset entry is on McM though, so I could see the CMSSW version and cmsDriver command required to produce my own samples from the miniAODs for the time being. So, I made some nanoAOD files which I could run through FAST-RA1. (A few people on this analysis are actively involved in developing FAST, and this framework will very likely be what we use in the analysis.)

37.2.1 Building the dataframe of files to run over

I needed to build a list of files to run over, which is straightforward if they're local but more involved if they're not. I could use `t2df_find_files.py`, with the option of an output text file containing the list to use in the next steps. The argument I needed to specify was the actual files, which allows wildcarding. If I looked up the dataset on DAS and clicked on "files", I could use the prefix `root://cms-xrd-global.cern.ch//` in the argument, then append the file paths "/store/.../*.root". This would use AAA with the global redirector to look for the file names. So the full command would look like this:

```
bin/t2df_find_files.py -d /TTJets_HT-2500
   toInf_TuneCUETP8M1_13TeV-madgraphMLM-pythia8/
   RunIISummer16NanoAOD-
   PUMoriond17_05Feb2018_94X_mcRun2_asymptotic_v2_ext1-v1/
   NANOAODSIM -o file_list_ttJets_2500toInf.txt --mc root://
   cms-xrd-global.cern.ch///store/mc/RunIISummer16NanoAOD/
   TTJets_HT-2500toInf_TuneCUETP8M1_13TeV-madgraphMLM-
   pythia8/NANOAODSIM/
```

```
PUMoriond17_05Feb2018_94X_mcRun2_asymptotic_v2_ext1-v1
/00000/*.root
```

If that doesn't work, as sometimes things the file query can hang, I can instead use Shane's `cms-das-query` package in **external/**. I just need to supply the dataset name, like the following:

```
bin/das_query --do-xsdb-query -f pandas -o ${FAST_RA1_\ROOT
}/file_list_ttJets_2500toInf.txt /TTJets_HT-2500
toInf_TuneCUETP8M1_13TeV-madgraphMLM-pythia8/
RunIISummer16NanoAOD-
PUMoriond17_05Feb2018_94X_mcRun2_asymptotic_v2_ext1-v1/
NANOAODSIM
```

which also gives me an output dataframe with the list of files. If I want to run over multiple datasets, I can specify the same output file. The information from the new datasets will be appended it. If I want to run over multiple datasets for a specific process, I can use a loop. If I add each dataset name to a file, say **qcd_bkg_datasets.dat**, I can run

```
for dataset in $(cat qcd_bkg_datasets.dat); do bin/das_query
  --do-xsdb-query -f pandas -o ./qcd_bkg_file_list.txt
$dataset; done
```

Note that if using `das_query`, I need to source the setup script in that directory, make sure my grid certificate proxy is active, make sure the **xsecdb** submodule is cloned, and install the pip packages in the requirements file.

37.2.2 Building the dataframes from nanoAODs

Then, I could specify binning and an event selection in a YAML config file, then I could run `nanoaod2dataframes_cfg` to get a dataframe. I can look at the other configs to get an idea of the layout and arguments, and then the actual executable for its arguments and options. So an example command would be

```
bin/nanoaod2dataframes_cfg --components
file_list_ttJets_2500toInf.txt --dont-use-nanoaodtools -o
 single_top_test fast_ra1/trees_to_dataframe/configs/
nanoaod_esh_ttH.yaml
```

If the file names from the previous step start with "/store", I need to enable my grid certificate proxy and add the option `-xrd-redirector root://cms-xrd-global.cern.ch//` so the executable knows to prepend the redirector on to them. If running over many files, it is preferable to

run on Condor rather than locally. I can specify the option `-mode htcondor`, but if running over files with xrootd, I need to initialise my grid certificate proxy somewhere that the Condor jobs can access (i.e., on **/afs** rather than the default **/tmp**). If I run the commands

```
export X509_USER_PROXY=/afs/cern.ch/user/e/ebhal/x509up_u${
   UID}
voms-proxy-init --voms cms --valid 168:00
```

A proxy will be saved in my home directory and, assuming `getenv = True` in the Condor job files, everything should work. When doing this often, it can get tedious having to enter my password all the time. I wrote a little script that only refreshes my proxy if required:

```
export X509_USER_PROXY=/afs/cern.ch/user/e/ebhal/x509up_u${
   UID}
inception=$(( $(date "+%s") - $(date "+%s" -r ${
   X509_USER_PROXY}) ))
proxy_length_hr=168 # 1 week
proxy_length_s=$(( $proxy_length_hr * 3600 ))
if (( "$inception" < "$proxy_length_s" )); then
    echo "No need to re-initialise proxy as it's still valid
   "
else
    voms-proxy-init --voms cms --valid ${proxy_length_hr}:00
fi
```

On Condor, the monitoring from AlphaTwirl needs to stay alive until all jobs finish. This makes it preferable to run everything in a `screen` session. One problem is that if I detach the session for too long, the kerberos authentication token expires and the whole thing will error out. By following this link: https://lhcb.github.io/analysis-essentials/shell-extras/persistent-screen.html (making sure the "CERN.CH" is capitalised when I try), I can set up screen sessions that refresh the token every so often. If my CERN account password changes, I'll have to repeat the steps with my new password. So my basic workflow for this step would look like this:

```
<make sure Kerberos keytab has been generated with link
   above. Only needs to be done once>
k5reauth -f -i 1200 -p ebhal -k ~/.krb5_tokens/ebhal.keytab
   -- screen
kinit
```

```
export X509_USER_PROXY=/afs/cern.ch/user/e/ebhal/x509up_u${
    UID}
voms-proxy-init --voms cms --valid 168:00
cd <path>/FAST-RA1
source setup.sh
bin/nanoaod2dataframes_cfg --mode htcondor <other arguments/
    options>
```

which will get things running.

If Condor jobs fail whilst the process is running, they will be automatically resubmitted, but *not* if they're held. In these cases, I can run

```
while true; do date; jobs="$(condor_q -hold $USER -format "%
    d." ClusterId -format "%d " ProcId)"; [ -n "$jobs" ] &&
    condor_release $jobs; sleep 120; done
```

in a separate window, also in a screen session for good measure.

The output from `nanoaod2dataframes_cfg` will be a dataframe with the events that pass the selections specified in the config file, and binned according to it. These can be loaded and plotted using `pandas` and `matplotlib`. FAST doesn't have any plotting functions built in, but those two packages are fairly straightforward to use.

37.2.3 Cross section reweighting

When skimming over datasets (e.g., if you want to apply some pre-selection and reduce its overall footprint), reweighting for the cross section needs to be taken into account. To first order, the weight applied is

$$w = \frac{\sigma \mathcal{L}}{N_{\text{tot.}} \varepsilon} \tag{37.1}$$

where σ is the cross section of the dataset (on DAS/XSDB for public datasets), $\mathcal{L}$ is the integrated luminosity, $N_{\text{tot.}}$ is the number of events in the dataset before any skimming, and ε is the filter efficiency.

For datasets that are split, e.g., by HT (like QCD or Drell-Yan), it's more complex as the weight needs to take all the datasets over the range into account. For Drell-Yan, the new calculation would be

$$w_i = \frac{\sigma_i' \mathcal{L}}{N_{\text{tot.}} \varepsilon}, \text{ where } \sigma_i' = \left(\frac{\sigma_i}{\Sigma_i \sigma_i} \right) \sigma_{\text{tot.@NLO}} \tag{37.2}$$

where the weight for a split dataset i needs to be corrected by normalising the cross section by the sum of the cross sections in all the datasets (assuming the split datasets are contiguous, i.e., HT100-200, HT200-400, etc.), then multiplied by the total NLO cross section.

37.3 Skimming over datasets

The new FAST framework, written from the ground up to be fully vectorised (reading in and running over a "chunk" of several thousand events at a time as opposed to a single-event loop), is much quicker than FAST-RA1. The bottleneck, in terms of processing time, is now in I/O. When performing studies and tweaking various aspects of the analysis, we want to iterate as fast as possible. The easiest way to do this is by reducing the number of events that must be processed, i.e., by skimming over the files we eventually want to run.

NanoAOD-tools (`https://github.com/cms-nanoAOD/nanoAOD-tools`) has built-in support to run over nanoAOD files and run centrally-maintained modules (to add branches for various weights and object cleaning/ID) as well as custom ones. Our goal is to first run all the datasets we plan to use in FAST through nanoAOD-tools first, where we can add these extra weight and scale factor branches, perform some object cleaning, and apply a loose preselection (i.e., HLT paths and some filters). CRAB can be used to funnel the actual processing and allow us to store the output on /hdfs at Bristol. As we will be using FAST at Bristol, the I/O bottleneck is mitigated and the number of events to be processed is also reduced.

Vukasin and Ann-Marie have a setup going for their VBF-side processing, which I've forked: `https://github.com/eshwen/VBFHToInv-nanoAOD-tools`. I've written custom modules for various things and consolidated all the centralised modules we need. Then it's straightforward to run all the public datasets on CRAB and send the output to Bristol. The main problem was running over private datasets as CRAB only supports those accessible on DAS. For this, I had to write a Condor framework to process those and then manually transfer them to Bristol. To access the output files, one can use the local path if they're working on Soolin. But for external users, they need the Bristol xrootd redirector `root://lcgse01.phy.bris.ac.uk:11001//`. If the files were produced with CRAB, they can append onto that `/store/user/<user>/<path>/`. If they are instead produced another way (like on Condor and manually transferred across), they should instead append `<path minus "/hdfs">`.

37.4 Analysis

Once we have the skimmed samples, we can run them through the analysis software. This is essentially a new FAST-RA1, but rewritten from the ground up. All the code is stored in https://gitlab.cern.ch/cms-chip/chip. This stores the analysis-specific code (**C**ombined **H**iggs to **I**nvisible **P**roject) such as scribblers and sample lists. The machinery that runs everything is the package `fast_carpenter`: https://pypi.org/project/fast-carpenter/. It takes in a YAML-style sequence cfg specifying each stage of the analysis like in FAST-RA1, and a YAML-style dataset cfg that takes in a list of files and any other metadata (cross section, etc.) that can be built with `fast_curator` (https://pypi.org/project/fast-curator/). Behind the scenes, it uses `uproot` to convert flat trees into dataframes, `AlphaTwirl` for handling batch submission, and some other packages.

I've helped by developing modules and scribblers in the cms-chip repository, as well as making some plotting tools. All the code should be able to run a mostly-complete analysis from the sequence cfg, and includes support for weights, systematics as well as complex scribblers and other modules.

The analysis note is here: http://cms.cern.ch/iCMS/jsp/db_notes/noteInfo.jsp?cmsnoteid=CMS%20AN-2018/299. We also have a GitLab repo, migrated from the far-outdated SVN: https://gitlab.cern.ch/tdr/notes/AN-18-299. We can also edit using Overleaf, an online LaTeX editor. By running the commands

```
git clone ssh://git@gitlab.cern.ch:7999/tdr/notes/AN-18-299
cd AN-18-299
git remote add overleaf https://git.overleaf.com/5
   cd2a70d3d57d07d68fa386a
git checkout -b overleaf_to_gitlab_v1
git pull overleaf master --allow-unrelated-histories
```

Whenever edits/updates are made to the project on Overleaf (https://www.overleaf.com/project/5cd2a70d3d57d07d68fa386a), it automatically updates the branch `overleaf/master`. By pulling the latest changes at a certain point in time, a merge request can then be made to the main GitLab repo above.

37.5 IOP 2019 (25/01/2019)

As I'm in my third year, I've been asked to present at the Joint APP and HEPP Annual Conference, organised by the IOP. I'll be giving a 15 minute talk on Higgs to invisible (as I've contributed to it more than semi-visible jets, recently). The title is **Combined Search for an Invisibly Decaying Higgs Boson in Hadronic Channels at $\sqrt{s} = 13$ TeV with CMS**, and the abstract is below:

The leading upper limit on the Higgs boson to invisible state branching ratio (BR) is 24%, while the Standard Model prediction sits far below at 0.1%. The observed value was measured using pp data collected by the CMS experiment between 2011 and 2015. Our analysis targets a better limit by using 13 TeV data from 2016-2018 – an integrated luminosity of over 130 fb^{-1} – in addition to performing the combination over all Higgs production modes from the outset rather than in a posthoc fashion. The hadronic channels we include are gluon-gluon fusion, ttH, vector boson fusion (VBF) and Higgs production in association with a vector boson (VH). Analysing each production mode in an orthogonal search region gives a high degree of sensitivity compared to previous attempts. In this talk, the finalised event selection, signal categorisation, data-driven background estimation and systematic uncertainties for the non-VBF production modes will be presented. A sufficiently accurate limit on the BR that is still above the Standard Model prediction may be interpreted in a beyond-Standard Model context. Constraints can be placed on theories that posit exotic particles or dark matter that couple to the Higgs, enhancing the invisible state BR.

My finished slides are here: . They include a very recent result for $ttH(H \rightarrow \text{inv.})$: [76], and also a combination from ATLAS (plots taken from conference note, paper uploaded soon after talk): [77].

An interesting new publication from CMS about ttH, which may prove useful, is in Ref. [78]. When looking into 2016 and 2017 data in nanoAOD, I found a good overview into how data is recorded and managed in CMS:

Chapter 38

Annual Progress Monitoring (second year) (01/06/2018)

Just as last year, I had to go through the Annual Progress Monitoring (APM) for my second year. I had to write a report: and go through an interview. The report outlined the work I'd done over my second year, including the SUSY α_{T} analysis, the EXO semi-visible jets analysis, the HIG Higgs to invisible analysis, and service work. I also had to discuss future plans.

Notes from the interview (13/08/2018) to follow up on:

- Research the motivations for studying/trying to discover dark matter at the LHC. These include the WIMP miracle and cosmological constraints on the $\sigma \times \mathrm{BR}$ for dark matter.
- Talk to Henning about a proper thesis outline that include chapter headings and overall details.
- Figure out timelines for semi-visible jets and Higgs to invisible analyses so that I can start making deadlines about when to get things finished by and when to start writing parts of my thesis.

38.1 Thesis outline

The following contains my thesis outline written in August 2018:

- Dedication
- Acknowledgements
- Abstract
- Contents

Introduction to High Energy Physics:

– Give an overview of particle physics in general. Discuss some of the history of the field, CERN and the LHC.
– Give an overview of the fundamental forces and particles.
– Give an overview of the state of dark matter, evidence for its existence, motivations to study it in general and at the LHC.

Theory (the Standard Model and Dark Matter):

– Discuss the Standard Model in detail, emphasising certain aspects as they relate to dark matter and the Higgs field (and boson).
– Discuss the theory behind the semi-visible jets analysis: strongly interacting dark sector in Hidden Valley scenario with a portal to the visible sector. Mentioning dark quarks, dark confinement scale, dark hadronisation and decay, running coupling, etc.
– Discuss the theory behind combined Higgs to inv.: branching fraction of H -> nunu set at 1% (include context) whilst current experimental limit is 24%

The Large Hadron Collider and CMS experiment:

– Explain CERN and the LHC in more detail.
– Give an overview of the CMS experiment and detector (including all subsystems and object identification).
– Either as a subsection in this chapter or in a separate chapter, discuss the Level-1 Trigger in depth. Emphasise the jet and energy sum triggers as I've worked on them, and Calorimeter Layer-2 for the same reason.

A search for dark matter from a semi-visible jet final state:

– Discuss how the theoretical aspects from the Theory chapter translate into an experimental search.
– Include object definitons, overall analysis strategy, triggers(?), signal production, event selection, background estimation and results/limit (including comparisons to similar searches).
– Current material: no public plots as of yet. Hope to have enough material for a PAS (if required) for Moriond 2019.

A search for dark matter from constraining the Higgs boson to invisible state decay mode:

– Discuss how the theoretical aspects from the Theory chapter translate into an experimental search.
– Include object definitons, overall analysis strategy, triggers(?), signal production, event selection, background estimation and results/limit (including comparisons to previous results).
– Current material: no public plots as of yet. Hope to have enough material for a PAS (if required) for Moriond 2019.

Summary, Conclusions and Prospects:

– Include a summary of thesis and work done over the course of my PhD with emphasis on the most important results/contributions.
– Mention the direction the semi-visible jet and Higgs to invisible analyses can take (sharing ideas/strategies I have, potential improvements with more LHC data and future prospects from potential future experiments).

– Appendices if required
– Index/glossary of HEP terms and acronyms
– Bibliography/references (either at the end of the thesis itself or at the end of each chapter)

Timeline:

– Combined Higgs to Invisible plans: "preliminary analysis" for Moriond 2019. Should consist of a first-pass analysis of 2016-2018 data with all four Higgs to invisible production modes and approved plots.
– Semi-visible jets plans: "preliminary analysis" for Moriond 2019. Should consist of a first-pass analysis of 2016-2017 data with s-channel signal. Will hopefully include t-channel either simultaneously or at a later date.
– Thesis plans: start writing by September 2019 for hand-in in March 2020. By September 2019, both above analyses should be well under way. Ideally, the papers will have been approved/published by this time. If not, they should be close and have produced all necessary plots (that I can use as placeholders until the final versions are ready).

Omission of 2016 paper on natural and split SUSY:

– Right now, the goal is just to include analysis chapters on semi-visible jets and combined Higgs to invisible. Henning suggested omitting natural/long-lived SUSY analysis for now as it makes thesis less coherent.
– Depending on how much progress is made with the two analyses that are currently planned, we may include a short chapter on SUSY.

Bibliography

[1] J. Abdallah *et al.*, 'Simplified Models for Dark Matter Searches at the LHC,' *Phys. Dark Univ.*, vol. 9-10, pp. 8–23, 2015. DOI: 10.1016/j.dark.2015.08.001. arXiv: 1506.03116 [hep-ph].

[2] D. Abercrombie *et al.*, 'Dark Matter Benchmark Models for Early LHC Run-2 Searches: Report of the ATLAS/CMS Dark Matter Forum,' A. Boveia, C. Doglioni, S. Lowette, S. Malik and S. Mrenna, Eds., 2015. arXiv: 1507.00966 [hep-ex].

[3] CMS Collaboration, 'Search for New Physics in the V/jet + MET final state,' CERN, Geneva, Tech. Rep. CMS-PAS-EXO-12-055, 2015.

[4] S. P. Martin, 'A Supersymmetry primer,' *Adv. Ser. Direct. High Energy Phys*, vol. 18, pp. 1–98, 1998. DOI: 10.1142/9789812839657_0001,10.1142/9789814307505_0001. arXiv: hep-ph/9709356 [hep-ph].

[5] CMS Collaboration, 'Search for new physics in final states with jets and missing transverse momentum in $\sqrt{s} = 13$ TeV pp collisions with the α_{T} variable,' CERN, Geneva, Tech. Rep. CMS-PAS-SUS-15-005, 2015.

[6] I. J. R. Aitchison, 'Supersymmetry and the MSSM: An Elementary introduction,' 2005. arXiv: hep-ph/0505105 [hep-ph].

[7] J. R. Ellis, 'Supersymmetry for Alp hikers,' in *2001 European school of high-energy physics, Beatenberg, Switzerland, 26 Aug-8 Sep 2001: Proceedings*, 2002, pp. 157–203. arXiv: hep-ph/0203114 [hep-ph].

[8] H. Murayama, 'Physics Beyond the Standard Model and Dark Matter,' in *Les Houches Summer School - Session 86: Particle Physics and Cosmology: The Fabric of Spacetime Les Houches, France, July 31-August 25, 2006*, 2007. arXiv: 0704.2276 [hep-ph].

[9] M. E. Peskin, 'Dark matter and particle physics,' *J. Phys. Soc. Jap.*, vol. 76, p. 111 017, 2007. DOI: 10.1143/JPSJ.76.111017. arXiv: 0707.1536 [hep-ph].

[10] J. Goodman, M. Ibe, A. Rajaraman, W. Shepherd, T. M. P. Tait and H.-B. Yu, 'Constraints on Dark Matter from Colliders,' *Phys. Rev.*, vol. D82, p. 116 010, 2010. DOI: 10.1103/PhysRevD.82.116010. arXiv: 1008.1783 [hep-ph].

[11] O. Buchmueller, S. A. Malik, C. McCabe and B. Penning, 'Constraining dark matter interactions with pseudoscalar and scalar mediators using collider searches for multijets plus missing transverse energy,' *Phys. Rev. Lett.*, vol. 115, p. 181 802, 18 Oct. 2015. DOI: 10.1103/PhysRevLett.115.181802.

[12] CMS Collaboration, 'Search for dark matter in final states with an energetic jet, or a hadronically decaying W or Z boson using $12.9\ \mathrm{fb}^{-1}$ of data at $\sqrt{s} = 13$ TeV,' 2016.

[13] G. Bertone, D. Hooper and J. Silk, 'Particle dark matter: Evidence, candidates and constraints,' *Phys. Rept.*, vol. 405, pp. 279–390, 2005. DOI: 10.1016/j.physrep.2004.08.031. arXiv: hep-ph/0404175 [hep-ph].

[14] K. C. Freeman, 'On the Disks of Spiral and So Galaxies,' *Astrophys. J.*, vol. 160, p. 811, Jun. 1970. DOI: 10.1086/150474.

[15] A. H. Broeils, 'The mass distribution of the dwarf spiral NGC 1560,' *Astron. Astrophys.*, vol. 256, pp. 19–32, Mar. 1992.

[16] M. Persic, P. Salucci and F. Stel, 'The universal rotation curve of spiral galaxies - I. The dark matter connection,' *Mon. Not. R. Astron. Soc.*, vol. 281, pp. 27–47, Jul. 1996. DOI: 10.1093/mnras/281.1.27. eprint: astro-ph/9506004.

[17] J. Einasto, 'Dark Matter,' *ArXiv e-prints*, Jan. 2009. arXiv: 0901.0632 [astro-ph.CO].

[18] J. E. Gunn and J. R. Gott III, 'On the Infall of Matter Into Clusters of Galaxies and Some Effects on Their Evolution,' *Astrophys. J.*, vol. 176, p. 1, Aug. 1972. DOI: 10.1086/151605.

[19] D. Merritt, A. W. Graham, B. Moore, J. Diemand and B. Terzić, 'Empirical Models for Dark Matter Halos. I. Nonparametric Construction of Density Profiles and Comparison with Parametric Models,' *Astron. J.*, vol. 132, pp. 2685–2700, Dec. 2006. DOI: 10.1086/508988. eprint: astro-ph/0509417.

[20] M. Lisanti, 'Lectures on Dark Matter Physics,' in *Proceedings, Theoretical Advanced Study Institute in Elementary Particle Physics: New Frontiers in Fields and Strings (TASI 2015): Boulder, CO, USA, June 1-26, 2015*, 2017, pp. 399–446. DOI: 10.1142/9789813149441_0007. arXiv: 1603.03797 [hep-ph].

[21] M. Kamionkowski, 'WIMP and axion dark matter,' in *High-energy physics and cosmology. Proceedings, Summer School, Trieste, Italy, June 2-July 4, 1997*, 1997, pp. 394–411. arXiv: hep-ph/9710467 [hep-ph].

[22] C. Yozin and K. Bekki, 'The quenching and survival of ultra-diffuse galaxies in the Coma cluster,' *Mon. Not. R. Astron. Soc.*, vol. 452, no. 1, pp. 937–943, 2015. DOI: 10.1093/mnras/stv1073. arXiv: 1507.05161 [astro-ph.GA].

[23] M. Drewes, 'The phenomenology of right handed neutrinos,' *Int. J. Mod. Phys. E*, vol. 22, no. 08, p. 1 330 019, 2013. DOI: 10.1142/S0218301313300191.

[24] T. Han, J. D. Lykken and R.-J. Zhang, 'On Kaluza-Klein states from large extra dimensions,' *Phys. Rev.*, vol. D59, p. 105 006, 1999. DOI: 10.1103/PhysRevD.59.105006. arXiv: hep-ph/9811350 [hep-ph].

[25] S. D. M. White and M. J. Rees, 'Core condensation in heavy halos: A two-stage theory for galaxy formation and clustering,' *Mon. Not. R. Astron. Soc.*, vol. 183, no. 3, p. 341, 1978. DOI: 10.1093/mnras/183.3.341.

[26] 'Dark Matter Summary Plots from CMS for ICHEP 2016,' Aug. 2016.

[27] Planck Collaboration, P. A. R. Ade, N. Aghanim, M. Arnaud, M. Ashdown, J. Aumont, C. Baccigalupi, A. J. Banday, R. B. Barreiro, J. G. Bartlett *et al.*, 'Planck 2015 results. XIII. Cosmological parameters,' *Astron. Astrophys.*, vol. 594, A13, A13, Sep. 2016. DOI: 10.1051/0004-6361/201525830. arXiv: 1502.01589.

[28] S. Weinberg, *Cosmology*, ser. Cosmology. OUP Oxford, 2008, ISBN: 9780191523601.

[29] S. Mertens, 'Direct Neutrino Mass Experiments,' *J. Phys. Conf. Ser.*, vol. 718, no. 2, p. 022 013, 2016. DOI: 10.1088/1742-6596/718/2/022013. arXiv: 1605.01579 [nucl-ex].

[30] C. Quigg, 'Cosmic Neutrinos,' in *Proceedings, 35th SLAC Summer Institute on Particle Physics: Dark matter: From the cosmos to the Laboratory (SSI 2007): Menlo Park, California, July 30- August 10, 2007*, 2008. arXiv: 0802.0013 [hep-ph].

[31] D. Clowe, A. Gonzalez and M. Markevitch, 'Weak-lensing mass reconstruction of the interacting cluster 1e 0657–558: Direct evidence for the existence of dark matter,' *Astrophys. J.*, vol. 604, no. 2, p. 596, 2004.

[32] B. Cox and J. Forshaw, *Universal: A Journey Through the Cosmos*. Penguin Books Limited, 2016, ISBN: 9780141968346.

[33] K. Nakamura and Particle Data Group, 'Review of particle physics,' *J. Phys. G*, vol. 37, no. 7A, p. 075 021, Jul. 2010. DOI: 10.1088/0954-3899/37/7a/075021.

[34] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer and T. Stelzer, 'MadGraph 5 : Going Beyond,' *J. High Energy Phys.*, vol. 06, p. 128, 2011. DOI: 10.1007/JHEP06(2011)128. arXiv: 1106.0522 [hep-ph].

[35] T. Sjostrand, S. Mrenna and P. Z. Skands, 'A Brief Introduction to PYTHIA 8.1,' *Comput. Phys. Commun.*, vol. 178, pp. 852–867, 2008. DOI: 10.1016/j.cpc.2008.01.036. arXiv: 0710.3820 [hep-ph].

[36] S. Ovyn, X. Rouby and V. Lemaitre, 'DELPHES, a framework for fast simulation of a generic collider experiment,' 2009. arXiv: 0903.2225 [hep-ph].

[37] E. Conte, B. Fuks and G. Serret, 'MadAnalysis 5, A User-Friendly Framework for Collider Phenomenology,' *Comput. Phys. Commun.*, vol. 184, pp. 222–256, 2013. DOI: 10.1016/j.cpc.2012.09.009. arXiv: 1206.1599 [hep-ph].

[38] M. Cacciari, G. P. Salam and G. Soyez, 'FastJet User Manual,' *Eur. Phys. J. C*, vol. 72, p. 1896, 2012. DOI: 10.1140/epjc/s10052-012-1896-2. arXiv: 1111.6097 [hep-ph].

[39] ATLAS Collaboration, 'Search for Dark Matter in association with a Higgs boson decaying to b-quarks in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector,' CERN, Geneva, Tech. Rep. ATLAS-CONF-2016-019, Mar. 2016.

[40] L. Randall and D. Tucker-Smith, 'Dijet Searches for Supersymmetry at the LHC,' *Phys. Rev. Lett.*, vol. 101, p. 221 803, 2008. DOI: 10.1103/PhysRevLett.101.221803. arXiv: 0806.1049 [hep-ph].

[41] S. Chatrchyan *et al.*, 'Search for supersymmetry at the lhc in events with jets and missing transverse energy,' *Phys. Rev. Lett.*, vol. 107, p. 221 804, 22 Nov. 2011. DOI: 10.1103/PhysRevLett.107.221804.

[42] T. Sakuma, 'Squark/gluino searches in hadronic channels with CMS,' 2016. arXiv: 1609.07445 [hep-ex].

[43] R. Placakyte, 'Parton Distribution Functions,' in *Proceedings, 31st International Conference on Physics in collisions (PIC 2011): Vancouver, Canada, August 28-September 1, 2011*, 2011. arXiv: 1111.5452 [hep-ph].

[44] G. Petrucciani, A. Rizzi and C. Vuosalo, 'Mini-aod: A new analysis data format for cms,' *J. Phys. Conf. Ser*, vol. 664, no. 7, p. 072 052, 2015.

[45] CMS Collaboration, 'A search for new phenomena in pp collisions at $\sqrt{s} = 13$ TeV in final states with missing transverse momentum and at least one jet using the α_{T} variable,' *Submitted to: Eur. Phys. J. C*, 2016, CMS-SUS-15-005, CERN-EP-2016-246. arXiv: 1611.00338 [hep-ex].

[46] A. M. Sirunyan *et al.*, 'Search for dark matter produced in association with heavy-flavor quarks in proton-proton collisions at $\sqrt{s} = 13$ TeV,' 2017. arXiv: 1706.02581 [hep-ex].

[47] A. M. Sirunyan *et al.*, 'Particle-flow reconstruction and global event description with the cms detector,' *JINST*, vol. 12, P10003, 2017. DOI: 10.1088/1748-0221/12/10/P10003. arXiv: 1706.04965 [physics.ins-det].

[48] M. Cacciari, G. P. Salam and G. Soyez, 'The Anti-k(t) jet clustering algorithm,' *J. High Energy Phys.*, vol. 04, p. 063, 2008. DOI: 10.1088/1126-6708/2008/04/063. arXiv: 0802.1189 [hep-ph].

[49] A. M. Steane, 'An introduction to spinors,' 2013. arXiv: 1312.3824 [math-ph].

[50] L. OKUN, Ed., *Leptons and Quarks*, ser. North-Holland Personal Library. Amsterdam: Elsevier, 1984. DOI: 10.1016/B978-0-444-86924-1.50001-5.

[51] T. Kaluza, 'Zum unitätsproblem in der physik,' *Sitzungsber. Preuss. Akad. Wiss. Berlin (Math. Phys.)*, p. 966, 1921.

[52] O. Klein, 'Quantentheorie und fünfdimensionale Relativitätstheorie,' *Zeitschrift fur Physik*, vol. 37, pp. 895–906, Dec. 1926. DOI: 10.1007/BF01397481.

[53] L. Williams, 'Electromagnetic Control of Spacetime and Gravity: The Hard Problem of Interstellar Travel,' *The Astronomical Review*, vol. 7, no. 2, pp. 5–28, Apr. 2012. DOI: 10.1080/21672857.2012.11519699.

[54] T. Flacke, D. W. Kang, K. Kong, G. Mohlabeng and S. C. Park, 'Electroweak Kaluza-Klein Dark Matter,' *J. High Energy Phys.*, vol. 04, p. 041, 2017. DOI: 10.1007/JHEP04(2017)041. arXiv: 1702.02949 [hep-ph].

[55] G. M. Shore, 'Superluminality and UV completion,' *Nucl. Phys.*, vol. B778, pp. 219–258, 2007. DOI: 10.1016/j.nuclphysb.2007.03.034. arXiv: hep-th/0701185 [hep-th].

[56] M. Dobbs and J. B. Hansen, 'The hepmc c++ monte carlo event record for high energy physics,' *Computer Physics Communications*, vol. 134, no. 1, pp. 41–46, 2001. DOI: 10.1016/S0010-4655(00)00189-2.

[57] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen and P. Z. Skands, 'An Introduction to PYTHIA 8.2,' *Comput. Phys. Commun.*, vol. 191, pp. 159–177, 2015. DOI: 10.1016/j.cpc.2015.01.024. arXiv: 1410.3012 [hep-ph].

[58] A. Buckley, J. Ferrando, S. Lloyd, K. Nordström, B. Page, M. Rüfenacht, M. Schönherr and G. Watt, 'LHAPDF6: parton density access in the LHC precision era,' *Eur. Phys. J. C*, vol. 75, p. 132, 2015. DOI: 10.1140/epjc/s10052-015-3318-8. arXiv: 1412.7420 [hep-ph].

[59] T. Cohen, M. Lisanti and H. K. Lou, 'Semi-visible Jets: Dark Matter Undercover at the LHC,' *Phys. Rev. Lett.*, vol. 115, no. 17, p. 171 804, 2015. DOI: 10.1103/PhysRevLett.115.171804. arXiv: 1503.00009 [hep-ph].

[60] CMS Collaboration, 'Search for top squark pair production in compressed-mass-spectrum scenarios in proton-proton collisions at $\sqrt{s} = 8$ TeV using the α_{T} variable,' CERN, Geneva, Tech. Rep. arXiv:1605.08993. CMS-SUS-14-006. CERN-EP-2016-103, May 2016, Comments: Submitted to Phys. Lett. B.

[61] V. Khachatryan *et al.*, 'A search for new phenomena in pp collisions at $\sqrt{s} = 13$ TeV in final states with missing transverse momentum and at least one jet using the α_{T} variable,' *Eur. Phys. J. C*, vol. 77, no. 5, p. 294, 2017. DOI: 10.1140/epjc/s10052-017-4787-8. arXiv: 1611.00338 [hep-ex].

[62] M. R. Pennington, 'Evolving images of the proton: Hadron physics over the past 40 years,' *J. Phys.*, vol. G43, p. 054 001, 2016. DOI: 10.1088/0954-3899/43/5/054001. arXiv: 1604.01441 [hep-ph].

[63] J. J. Brooke, B. L. S. Mathias, A. Tapper and N. Wardle, 'Calibration and Performance of the Jets and Energy Sums in the Level-1 Trigger,' Sep. 2013.

[64] A. M. Sirunyan *et al.*, 'Search for natural and split supersymmetry in proton-proton collisions at $\sqrt{s} = 13$ TeV in final states with jets and missing transverse momentum,' *J. High Energy Phys.*, vol. 2018, no. 5, p. 25, May 2018. DOI: 10.1007/JHEP05(2018)025.

[65] M. Duerr, A. Grohsjean, F. Kahlhoefer, B. Penning, K. Schmidt-Hoberg and C. Schwanenberger, 'Hunting the dark Higgs,' *J. High Energy Phys.*, vol. 04, p. 143, 2017. DOI: 10.1007/JHEP04(2017)143. arXiv: 1701.08780 [hep-ph].

[66] T. Cohen, M. Lisanti, H. K. Lou and S. Mishra-Sharma, 'LHC Searches for Dark Sector Showers,' *J. High Energy Phys.*, vol. 11, p. 196, 2017. DOI: 10.1007/JHEP11(2017)196. arXiv: 1707.05326 [hep-ph].

[67] J. Allison *et al.*, 'Recent developments in geant4,' *Nucl. Instrum. Methods Phys. Res. A*, vol. 835, pp. 186–225, 2016. DOI: 10.1016/j.nima.2016.06.125.

[68] G. P Salam and M. Cacciari, 'Jet clustering in particle physics, via a dynamic nearest neighbour graph implemented with cgal,' Apr. 2006.

[69] J. Thaler and K. Van Tilburg, 'Identifying Boosted Objects with N-subjettiness,' *J. High Energy Phys.*, vol. 03, p. 015, 2011. DOI: 10.1007/JHEP03(2011)015. arXiv: 1011.2268 [hep-ph].

[70] M. Freytsis, T. Volansky and J. R. Walsh, 'Tagging Partially Reconstructed Objects with Jet Substructure,' *Phys. Lett. B*, vol. 769, pp. 333–338, 2017. DOI: 10.1016/j.physletb.2016.08.044. arXiv: 1412.7540 [hep-ph].

[71] A. V. Bednyakov, B. A. Kniehl, A. F. Pikelner and O. L. Veretin, 'On the b-quark running mass in QCD and the SM,' *Nucl. Phys.*, vol. B916, pp. 463–483, 2017. DOI: 10.1016/j.nuclphysb.2017.01.004. arXiv: 1612.00660 [hep-ph].

[72] M. Aaboud *et al.*, 'Constraints on mediator-based dark matter and scalar dark energy models using $\sqrt{s} = 13$ TeV pp collision data collected by the ATLAS detector,' 2019. arXiv: 1903.01400 [hep-ex].

[73] S. Heinemeyer *et al.*, *Handbook of LHC Higgs Cross Sections: 3. Higgs Properties: Report of the LHC Higgs Cross Section Working Group*, S. Heinemeyer, Ed., ser. CERN Yellow Reports: Monographs. Jul. 2013.

[74] V. Khachatryan *et al.*, 'Searches for invisible decays of the Higgs boson in pp collisions at $\sqrt{s} = 7$, 8, and 13 TeV,' *J. High Energy Phys.*, vol. 02, p. 135, 2017. DOI: 10.1007/JHEP02(2017)135. arXiv: 1610.09218 [hep-ex].

[75] A. M. Sirunyan *et al.*, 'Search for invisible decays of a Higgs boson produced through vector boson fusion in proton-proton collisions at $\sqrt{s} = 13$ TeV,' *Phys. Lett. B*, vol. 793, pp. 520–551, 2019. DOI: 10.1016/j.physletb.2019.04.025. arXiv: 1809.05937 [hep-ex].

[76] 'First constraints on invisible Higgs boson decays using $\mathrm{t\bar{t}H}$ production at $\sqrt{s} = 13$ TeV,' CERN, Geneva, Tech. Rep. CMS-PAS-HIG-18-008, 2019.

[77] M. Aaboud *et al.*, 'Combination of searches for invisible Higgs boson decays with the ATLAS experiment,' *Phys. Rev. Lett.*, vol. 122, no. 23, p. 231 801, 2019. DOI: 10.1103/PhysRevLett.122.231801. arXiv: 1904.05105 [hep-ex].

[78] A. M. Sirunyan *et al.*, 'Observation of $t\bar{t}H$ Production,' *Phys. Rev. Lett.*, vol. 120, p. 231 801, 23 Jun. 2018. DOI: 10.1103/PhysRevLett.120.231801.