Using simplified likelihood in reinterpretation of physics beyond the Standard Model

A thesis submitted to Indian Institute of Science Education and Research Pune in partial fulfillment of the requirements for the BS-MS Dual Degree Programme

by

Avinash Verma

Supervisor: Dr. Nishita Desai

TIFR Mumbai



Indian Institute of Science Education and Research
Pune Dr. Homi Bhabha Road, Pashan, Pune 411008
INDIA

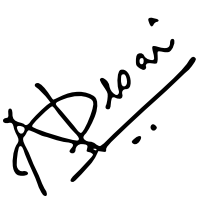
April, 2022

© Avinash Verma 2022

All rights reserved

Certificate

This is to certify that this dissertation entitled **Using simplified likelihood in reinterpretation of physics beyond the Standard Model** towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Avinash Verma at Tata Institute of Fundamental Research under the supervision of Dr. Nishita Desai, Department of physics, Mumbai during the academic year 2021-2022



Dr. Nishita Desai

Committee:

Dr. Nishita Desai

Dr. Seema Sharma

This thesis is dedicated to my parents Mr. Srinath Verma and Mrs. Savitri Devi, my grandfather Mr. Ram kumar Verma, who have always been supportive throughout my life.

Declaration

I hereby declare that the matter embodied in the report entitled **Using simplified likelihood in reinterpretation of physics beyond the Standard Model** are the results of the work carried out by me at the Department of physics, Tata Institute of Fundamental Research, Mumbai, under the supervision of Dr. Nishita Desai and the same has not been submitted elsewhere for any other degree.



Avinash Verma

Acknowledgement

I want to express my sincere gratitude and appreciation to my supervisor Dr. Nishita Desai from TIFR Mumbai, for guiding me on the path of exploring the field of CMS experiments And providing me with her continuous guidance and suggestions. It is also my pleasure and honour to acknowledge Dr. Seema Sharma, IISER Pune for all her support and advice on the computational aspect of this master's project and for introducing me to Nishita Ma'am, who led to the genesis of this project.

From the lab, I would like to extend my sincere thanks to Chirag for helping me out in the necessary software installation. Furthermore, a special mention must go to Mr. Mangesh Sonawane for making the project work smooth and fluid.

I would also like to express my gratitude for the support and blessings that my parents and family members have provided me throughout the year. My special appreciation goes to Richa, who always kept a smile on my face throughout the entire process of my project due to her warmth and well-being. It won't have been possible to reach this point without all the people that make it possible in managing things subtly and efficiently. I thank the incredible people in this little space, including Kaustubh Atey, Gitesh Masram, Ankur Prakash, and Aditya Singh Bagri. Without them, it won't have been possible to keep up this work manageable.

Abstract

Reinterpretation of the results published by experimental collaborations is vital for theorists to check their own BSM model against the published data. Often, background model used in an analysis has large number of parameters to model the uncertainties. To test the applicability of an analysis to a new BSM model requires dedicated reinterpretation (CheckMATE is one such tool). This means only a few signal hypothesis can be tested, given the limitation on computational resources. To overcome this, the idea of simplified likelihood was proposed.

Simplified likelihood uses a reduced set of parameters to model the background contributions. It makes reinterpretation of the CMS experiments relatively easy. In this thesis, a procedure for constructing the simplified likelihood has been presented. Validation of this procedure has been done using toy data provided by the CMS. We also implemented a CMS search for dark matter and shows how this simplified likelihood framework can be used to set exclusions on the model parameters.

Contents

Abstract **1**

1 Introduction **3**
1.1 Standard Model of particle physics 3
1.2 Large Hadron Collidor (LHC) 4
1.3 CMS detector 4
1.4 BSM searches at LHC 5
1.5 Reinterpretation of LHC analysis 6

2 Introduction to Simplified Likelihood **7**
2.1 Simplified Likelihood Framework 7
2.1.1 Likelihood function 7
2.1.2 Bayes' theorem 7
2.1.3 Maximum Likelihood Estimation 8
2.1.4 Poisson distribution 8
2.1.5 Simplified Likelihood 8
2.1.6 Covariance matrix creation 10
2.2 Newton-Raphson method 13
2.3 Results for toy data 13
2.4 Discussion 16
2.4.1 Simplified case: uncorrelated background contributions . . 16
2.4.2 Simplified case: correlation across background contributions 17

3 CheckMATE **19**
3.1 Introduction 19
3.2 Process flow 19
3.3 Statistical evaluation 20

3.4 Usage . . . 21
3.5 Results of the run . . . 22
3.6 AnalysisManager . . . 23
3.6.1 Final state objects . . . 24
3.6.2 Final state functions . . . 25

4 CMS Monojet search **27**
4.1 Introduction . . . 27
4.2 Final state objects required for the analysis . . . 28
4.3 Jet clustering using anti-kt algorithm . . . 29
4.4 Jet pruning . . . 30
4.5 N-subjettiness . . . 31
4.6 Event selection . . . 31
4.7 Result . . . 32
4.8 Discussion . . . 33

5 Applying SL framework to monojet search **35**
5.1 Dark matter with axial vector mediator . . . 35
5.2 Generating signal events . . . 36
5.3 Result . . . 38
5.4 Discussion . . . 38

A CheckMATE **42**
A.1 Implemented dark matter search . . . 42

Chapter 1

Introduction

1.1 Standard Model of particle physics

Standard Model (SM) is the most successful theory of particle physics. It describes three of the four fundamental forces of the nature (electromagnetic force, weak and strong nuclear force) with great accuracy, but hasn't been able to incorporate the gravitational interaction. Currently this model has 12 elementary particles of spin $\frac{1}{2}$ (fermions), 4 gauge bosons (spin 1) and higgs boson (spin 0) discovered recently. Although Standard Model has been very consistent in providing the experimental predictions, it stills falls behind in becoming a complete theory for fundamental interactions. Some inadequacies in the model are explained below.

- **Gravitation** The model has not been able to explain the gravitational interaction, experimental confirmation of graviton (hypothetical particle assumed to be a mediator for the gravitational interaction) has still not been achieved.
- **Massive neutrinos** Observation of neutrino oscillations has established the presence of massive neutrinos, which are currently not explained by the model.
- **The higgs mass** Quantum contribution to the higgs mass makes it huge and comparable to Planck scale, which does not fit with the observed mass.

- **Dark Matter** Several astronomical observations have provided evidence for the existence of dark matter in the universe. The SM doesn't has any explanation for this.

1.2 Large Hadron Collidor (LHC)

In order to validate or reject a theory, we need observations. In order to study Standard Model and other BSM models, we need to study the interaction and behaviour of fundamental particles at high energy which often gives rise to new phenomenon. Such high energy interaction of fundamental particles is often seen in astronomical studies. But these self-occuring phenomenons are often very hard to study, hence we need a control environment to study these interactions. For this purpose, we use particle accelerators which provide a control environment for high energy collisions studies.
LHC is the world's largest particle accelerator ever built near Geneva. It has been operational since 2008. It consists of a 27-km long ring of superconducting magnet and other structures to boost the particle energy along the way. Two high energy particle (primarly proton) beams are made to travel in opposite direction along the ring at a speed closer to light. These beams are accelerated and directed by the magnetic fields produced using superconducting magnets. LHC has four locations (detectors) where these beams are made to collide- CMS, ATLAS, ALICE and LHCb. These detectors then study these collisons and provide further analysis.

1.3 CMS detector

Compact Muon Solenoid (CMS) is one of the two general purpose detectors at LHC to study the proton-proton and heavy ion collisions. The detector has been build to study the signatures of new physics in the proton-proton collision. It identifies different final state objects produced during the collision and measures their four-momenta and directions with great accuracy. CMS is cylindrical in shape and its axis passes through the beam pipe.
It has a layered structure and each layer is designed to detect different final state objects as seen in figure 1.1. It has a superconducting solenoid with internal radius of 3m, which produces a magnetic field of 3.8T along the beam direction. The heavier SM particles like W,Z and H boson and also top quarks and

tau leptons are unstable and decays into muons, electrons and neutrinos. The quarks and gluons undergo fragmentation and hadronization forming a spray of hadrons collectively called jets. All of these particles are identified in the different layer of detector.

Particle-Flow (PF) algorithm is used to identify different particles, it utilises the information from all the detector components to optimally combine them. Photons energy is identified from ECAL. Electrons energy is determined from the combination of momentum at the primary interaction vertex in the tracker, and corresponding ECAL value. Muons energy is determined from the curvature of corresponding track. Charged hadron energy is determined from the combination of momentum measured in tracker and corresponding HCAL and ECAL information. At last, neutron hadron energy is obtained from the corresponding HCAL and ECAL value.

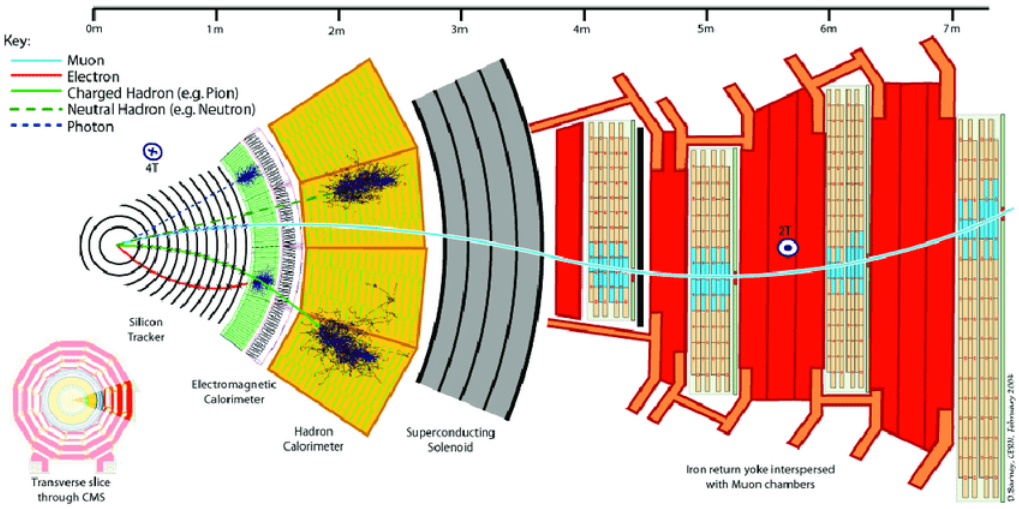


Figure 1.1: Transverse slice of the CMS detector

1.4 BSM searches at LHC

LHC performs several BSM searches like SUSY, Dark matter, heavy resonance and Long Lived particles by applying variety of search techniques. Often, results of these analyses are presented in the form of model-independent searches. Generally one look for a specific signal (corresponding to some BSM model) in

a data supposed to be composed of potential signal and plathera of background data. After that, this becomes a hypothesis testing where one looks whether the obtained statistics is more consistence with signal+background or background only hypothesis. The experimental collaborators at LHC tries to look for different experimental signatures in the final state that could potentially be the sign of a new physics. For example, high missing transverse energy, one lepton channel, two lepton channel, Jet channels etc.

1.5 Reinterpretation of LHC analysis

Usually the results for BSM searches presented by experimental collaborations are often interpreted using very few new physics models, which serve as a benchmark for the sensitivity of the search. In case of dark matter search using different techniques and models, we have not been able to get any dark matter signatures in the data analyzed so far. Hence it become more imperative to test other models that has been proposed. One way of reinterpretation is to generate signal contribution using event generators like PYTHIA and getting detector responses using detector simulator like DelPhes. Background predictions and systematic uncertainity (as provided by the experimental collaborations) are further used to perform the reinterpretation.
A key ingredient in reinterpretation is to be able to predict the expected number of events in each search region for the given BSM model. As expected events has two components, signal (generated using the event generators) and background (full background model is needed to know this). As covariance matrix is being provided by the CMS collaborations, this can be used to model the full background contribution for the given BSM model. Hence Simplified likelihood is used to model the background contribution of the BSM model under consideration.
In this thesis, we have started with summing up the required theory for developing the simplified likelihood model and then tested it on a toy data. After that we have added a CMS monojet analysis in the CheckMATE which can be used by other people to check their own model or put new constraints on the existing BSM model. Then we used this analysis and simplified likelihood to get the test statistic for a simplified dark matter model with axial-vector mediator.

Chapter 2

Introduction to Simplified Likelihood

2.1 Simplified Likelihood Framework

As the subatomic physics is governed by quantum mechanics, the outcome of a collision experiment cannot be known ahead of the time. The best a theory can do is to predict the probability of an outcome. Probability of this outcome needs to be maximised for the model under consideration. One way to achieve this is by using the simplified likelihood (SL) [11] function.

2.1.1 Likelihood function

Likelihood describes the joint probability of observed data given model parameters of the statistical model. For each specific parameter θ in the parameter space, likelihood function $L(\theta|X)$ gives the probability to observe the data X.

2.1.2 Bayes' theorem

Bayes' theorem gives the probability of an event based on the prior knowledge of conditions that might be related to the event. Bayesian inference is a method to update the probability of a hypothesis as more and more data is available. Bayes' theorem

$$P(H|E) = \frac{P(E|H).P(H)}{P(E)}$$

H: Hypothesis
$P(H)$: Prior probability of the hypothesis i.e. before the current data E is observed.
$P(H|E)$: Posterior probability i.e. probability of H after E is observed.
$P(E|H)$: Probability of observing E given H. This is called likelihood.

2.1.3 Maximum Likelihood Estimation

Maximum likelihood estimation (MLE) is a method to find the parameters of a probability distribution (model) given some observed data. This is achieved by maximizing a likelihood function such that given data is maximally probable under the assumed distribution.
Let $L(\boldsymbol{\theta} : \mathbf{y})$ be the likelihood function, where $\boldsymbol{\theta}$ are the parameters of pdf and $\mathbf{y}$ is the observed data. Then this likelihood can be maximised (w.r.t. θ_i) by requiring

$$\frac{\partial L(\boldsymbol{\theta} : \mathbf{y})}{\partial \theta_i} = 0 \tag{2.1}$$

Parameters of the model are obtained by solving the above equation.

2.1.4 Poisson distribution

It is a discrete probability distribution that expresses the probability of observing a given number of events in a given (fixed) interval.

$$P(x = k) = \frac{\lambda^k . e^{-\lambda}}{k!}$$

k: Number of events
λ: Expected value of x

2.1.5 Simplified Likelihood

The full likelihood used to set confidence limits is given by

$$L(\mu, \theta) = P(data|\mu.s(\theta) + b(\theta)).p(\tilde{\theta}|\theta)$$

where:

$s(\theta)$: Expected signal
$b(\theta)$: Expected background

μ: Signal strength modifier
$p(\tilde{\theta}|\theta)$: Probability density function for θ.

Here $P(data|\mu.s(\theta)+b(\theta))$ is the probability to observe the data given expected signal and background events (i.e. model). Here θ is the nuisance parameter that is used to model the variations in both signal and background due to systematic uncertainities.
In order to make theory predictions as close to observed data as possible this likelihood needs to be maximised. We know that the probability to observe certain number of events is given by Poisson distribution. But maximising the full likelihood is very computationally expensive. Hence we will use binned likelihood for this maximization, by categorizing the observed events in different bins based on some discriminating variable.
Assuming that number of events in a particular bin also follow Poisson distribution, we have

$$P(data|s+b) = \frac{(s_i+b_i)^{n_i}e^{-(s_i+b_i)}}{n_i!}$$

s_i: Expected signal in i^{th} bin
b_i: Expected background in i^{th} bin
n_i: Observed event in i^{th} bin

The number of nuisance parameters can be very large, making background modeling very difficult. However assuming that these nuisance parameters are independent in nature gives rise to SL framework, which is defined as:

$$L_s(\mu,\theta) = \prod_{i=1}^{N} \frac{(\mu\cdot s_i+b_i+\theta_i)^{n_i}\ e^{-(\mu\cdot s_i+b_i+\theta_i)}}{n_i!}\ \cdot \exp(-\frac{1}{2}\theta^T V^{-1}\theta) \qquad (2.2)$$

where

V: covariance matrix of the expected background in each region. We also add a signal strength modifier μ which can be extracted by maximising the likelihood.
Considering the above Simplified Likelihood form in equation 2.2, log likelihood

will become

$$\ln(L_s(\mu,\theta)) = [\sum_{i=1}^{N} n_i \ln(\mu \cdot s_i + b_i + \theta_i) - (\mu \cdot s_i + b_i + \theta_i) - \ln n_i!] - \frac{1}{2}\theta^T V^{-1}\theta \tag{2.3}$$

Some assumptions have to be made in order to get the above simplified equation 2.2

1. Background follows gaussian distribution, and variance does not depend on θ.

2. Number of events n in each region is independent.

3. Expected signal is small compared to the background. Hence uncertainties in the signal can be neglected, and it can be assumed that all the systematic uncertainties are from the background.

All these assumptions lead to modelling $p(\tilde{\theta}|\theta)$ as a multivariate gaussian distribution given by:

$$\exp(-\frac{1}{2}\theta^T V^{-1}\theta)$$

The next step is to define a test statistic that encapsulates all the information about observed data, expected signal, background, and uncertainties involved. We define:

$$q_\mu = -2 \ln \frac{L_s(\mu, \hat{\theta}_\mu)}{L_s(\hat{\mu}, \hat{\hat{\theta}})} \tag{2.4}$$

where

μ is signal strength, $\hat{\theta}_\mu$ is MLE of nuisance parameter with given μ

$\hat{\mu}$ is MLE for signal strength and $\hat{\hat{\theta}}$ is MLE for nuisance parameter

This q_μ can be used to check for the presence or absence of signal and set exclusion limits on the BSM model under consideration. This is achieved by comparing q_μ with its probability density function. Details on setting confidence limits using this method can be found here [17].

2.1.6 Covariance matrix creation

We expect CMS will provide the covariance matrix for the backgrounds in the different search regions, however assuming all uncertainity come from MC only, we can also get this matrix through simulation using event generators as follows.

More details can be found here [8].
In the simplified likelihood model, the systematic uncertainties are considered to be incapsulated in background modelling, and the signal only depends on parameter of interest. Simplified likelihood is given by

$$L_s(\alpha,\theta) = \prod_{I=1}^{P} P(n_I^{obs} \mid n_{s,I}(\alpha) + a_I + b_I\theta_I + c_I\theta_I^2) \cdot \frac{e^{-\frac{1}{2}\theta^T\rho^{-1}\theta}}{\sqrt{(2\pi)^P}}$$

α: Parameter of interest
θ: Nuisance parameter
ρ: Correlation matrix of the expected background contributions
P: Number of nuisance parameters
Here background of I^{th} region is

$$n_I = a_I + b_I\theta_I + c_I\theta_I^2$$

Often, uncertainties and event yields are coded in the Monte Carlo event generator. We can use this to get the parameters for our SL, which are (a,b,c,ρ). In order to obtain the parameters, we will work with the first moment, third moment, and covariance matrix for the event yield (n_I).

Systematic uncertainty in background is denoted by θ where θ has been assumed to follow multivariate gaussian distribution, with $E[\theta_I] = 0$ and $E[\theta_I^2] = 1$ also $E[\theta_I\theta_J] = \rho_{IJ}$

Calculation of parameters of SL:

1. Calculating first moment:

$$\begin{aligned} m_{1,I} &= E[n_I] \\ &= E[a_I + b_I\theta_I + c_I\theta_I^2] \\ &= E[a_I] + E[b_I\theta_I] + E[c_I\theta_I^2] \\ &= a_I + b_I E[\theta] + c_I E[\theta_I^2] \\ &= a_I + c_I \end{aligned} \tag{2.5}$$

2. Calculating third centered moment:

$$
\begin{aligned}
m_{3,I} &= E[(n_I - E[n_I])^3] \\
&= E[(a_I + b_I\theta_I + c_I\theta_I^2 - a_I - c_I)^3] \\
&= E[(b_I\theta_I + c_I\theta_I^2 - c_I)^3] \\
&= E[b^3\theta_I^3 + c^3\theta_I^6 - c^3 - 3c^3\theta_I^4 + 3c^3\theta_I^2 + 3b^2c\theta_I^4 \\
&+ 3bc^2\theta_I^5 - 3bc^2\theta_I^3 - 3b^2c\theta_I^2 - 3bc^2\theta_I^3 + 3bc^2\theta_I] \\
&= b^3E[\theta_I^3] + c^3E[\theta_I^6] - c^3 - 3c^3E[\theta_I^4] + 3c^3E[\theta_I^2] + 3b^2cE[\theta_I^4] \\
&+ 3bc^2E[\theta_I^5] - 3bc^2E[\theta_I^3] - 3b^2cE[\theta_I^2] - 3bc^2E[\theta_I^3] + 3bc^2E[\theta_I]
\end{aligned}
$$

3. As θ_I follows a guassian distribution, so we calculated the $E[\theta_I^6] = 15$ and $E[\theta_I^5] = 0$ and $E[\theta_I^4] = 3$ and $E[\theta_I^3] = 0$ using moment generating function (MGF), putting these values in above equation, we get:

$$m_{3,I} = 6b_I^2c_I + 8c_I^3 \tag{2.6}$$

4. In the same way, covariance matrix for event yield can be calculated and will get:

$$m_{2,IJ} = E[(n_I - E[n_I])(n_J - E[n_J])] = b_Ib_J\rho_{IJ} + 2c_Ic_J\rho_{IJ}^2 \tag{2.7}$$

5. We need one more equation to solve all the four parameters completely, so our fourth equation will be replacing J in $m_{2,IJ}$ with I:
   So :

$$m_{2,II} = b_I^2 + 2c_I^2 \; as \; \rho_{II} = 1 \tag{2.8}$$

   So we have four equation in $m_{1,I}$ $m_{2,II}$ $m_{2,IJ}$ $m_{3,I}$. We can get all the four parameters for SL using these.

$$\rho_{IJ} = \frac{1}{4c_Ic_J}(\sqrt{(b_Ib_J)^2 + 8c_Ic_Jm_{2,IJ}} - b_Ib_J) \tag{2.9}$$

   Here ρ_{IJ} is the correlation matrix, we can get covariance matrix from this.

2.2 Newton-Raphson method

Newton-Raphson method [7] is used to find the roots of a real valued function, it works in a iterative manner and gives a better approximation at every step. For a basic single variable function $f(x)$ and a initial guess of root x_i,

$$x_n = x_i - \frac{f(x_i)}{f'(x_i)} \tag{2.10}$$

Here x_n is the better approximation of the root than the initial guess x_i. This process can be iterated by replacing x_i with x_n until we get the sufficient precision.

This method can also be generalised for a system of nonlinear equations as follows.

Let's take a system of equations as

$$f_1(x_1, x_2, \ldots, x_n) = 0 \tag{2.11}$$

$$\vdots \tag{2.12}$$

$$f_n(x_1, x_2, \ldots, x_n) = 0 \tag{2.13}$$

In order to solve, number of variables should be equal to number of equations. Writing the above system of equations in vector form

$$\mathbf{f(x)=0}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{f(x)} = (f_1(x), \ldots, f_n(x))$

Newton-Raphson iteration for this system will be

$$\boldsymbol{x_n = x_i - (Df(x_i))^{-1} f(x_i)} \tag{2.14}$$

Here $\boldsymbol{Df(x)}$ is the jacobian matrix whose (i,j)th value is

$$\frac{\partial f_i(\mathbf{x})}{\partial x_j}$$

2.3 Results for toy data

To check our implementation, we reproduce the calculations given in [11] using their toy data and covariance matrix.

Table 2.1: Toy data used for validation from [11]

Search region(ı)	Data ((n_i))	Total background(i)	Signal($_i$)
1	1964	2006.4	47.0
2	877	836.4	29.4
3	354	350.0	21.1
4	182	147.1	14.3
5	82	62.0	9.4
6	36	26.2	7.1
7	15	11.1	4.7
8	11	4.7	4.3

Table 2.2: Covariance matrix across background regions

	1	2	3	4	5	6	7	8
8	-381.1	38.3	111.2	92.5	61.0	36.4	20.6	11.2
7	-719.8	78.1	212.1	174.3	114.1	67.6	38.0	20.6
6	-1316.5	154.9	392.3	318.1	206.2	121.3	67.6	36.4
5	-2286.4	294.0	691.1	551.8	353.9	206.2	114.1	61.0
4	-3599.9	513.3	1110.9	871.2	551.8	318.1	174.3	92.5
3	-4520.3	754.6	1454.0	1110.9	691.1	392.3	212.1	111.2
2	-1691.3	603.1	754.6	513.3	294.0	154.9	78.1	38.3
1	16787.2	-1691.3	-4520.3	-3599.9	-2286.4	-1316.5	-719.8	-381.1

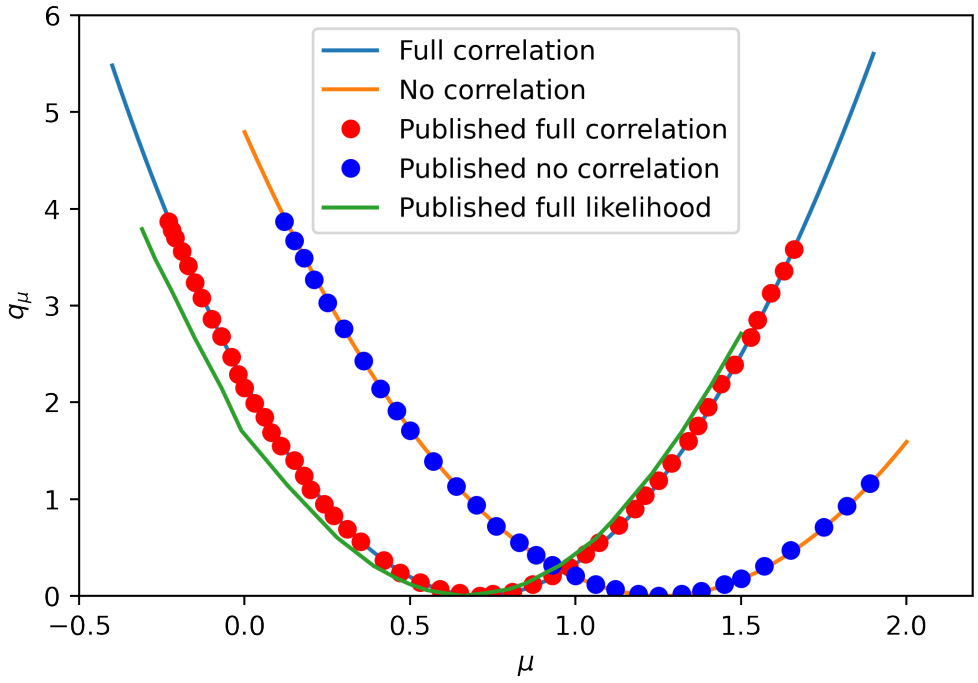


Figure 2.1: Comparison of our Simplified likelihood (with correlation and without correlation) with published [11] curve for the same and with the full likelihood provided by CMS.

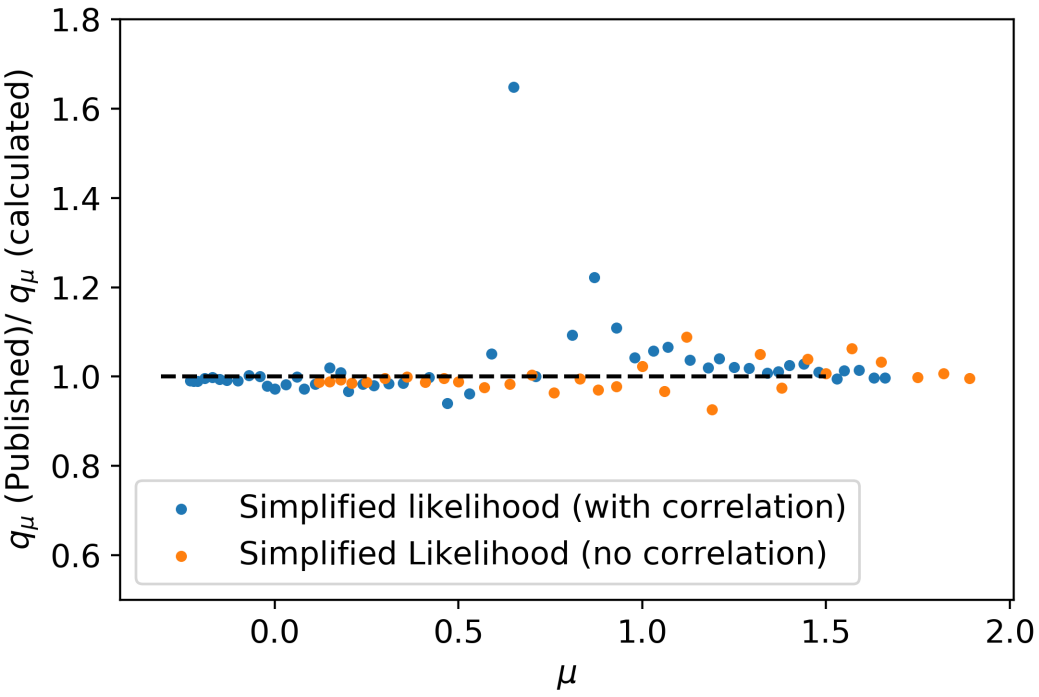


Figure 2.2: Ratio of published vs calculated test statistic (q_μ)

2.4 Discussion

2.4.1 Simplified case: uncorrelated background contributions

In this case, all the covariance matrix elements except diagonal values become zero. Hence the log likelihood equation 2.3 becomes

$$[\sum_{i=1}^{N} n_i \ln(\mu \cdot s_i + b_i + \theta_i) - (\mu \cdot s_i + b_i + \theta_i) - \ln n_i!] \; - \frac{1}{2} \sum_{1}^{N} V_{ii}^{-1} \theta_i^2 \qquad (2.15)$$

Maximising above Simplified Likelihood as per equation 2.1 gives

$$\sum_{i=1}^{N} \frac{n_i \cdot s_i}{\hat{\mu} \cdot s_i + b_i + \theta_i} = \sum_{i=1}^{N} s_i \tag{2.16}$$

and

$$\frac{n_i}{\mu \cdot s_i + b_i + \hat{\theta}_i} = 1 + V_{ii}^{-1} \hat{\theta}_i \tag{2.17}$$

where equation 2.16 and 2.17 are maximised with respect to μ and θ_i respectively.
For each value of μ, $\hat{\theta}_\mu$ is calculated analytically as

$$\hat{\theta}_i = \frac{-(\mu \cdot s_i \cdot V_{ii}^{-1} + b_i \cdot V_{ii}^{-1} + 1) + \sqrt{(\mu \cdot s_i \cdot V_{ii}^{-1} + b_i \cdot V_{ii}^{-1} + 1)^2 \ - 4 \cdot V_{ii}^{-1}(\mu \cdot s_i + b_i - n_i)}}{2V_{ii}^{-1}} \tag{2.18}$$

These μ and their corresponding $\hat{\boldsymbol{\theta}}_{\boldsymbol{\mu}}$ is used to profile out $\hat{\mu}$ (looking for the value of μ for which log likelihood (Equation 2.3) is maximum)
As we now have the parameters of the log likelihood, we can now construct the test statistic q_μ (Equation 2.4) with different μ values.
From figure 2.1 it is evident that calculated q_μ and published q_μ are in great agreement (in case of no correlation). In figure 2.2, ratio of both the values is close to 1. Values of both q_μ published and q_μ calculated reaches 0 around μ=1.26, hence we see a slight deviation from line $y = 1$. Mean squared error (MSE) of the q_μ is 0.0004.

2.4.2 Simplified case: correlation across background contributions

In this case, covariance matrix as provided in Table 2.2 is used.
Maximising the log likelihood (equation 2.3) as per equation 2.1 gives

$$\sum_{i=1}^{N} \frac{n_i \cdot s_i}{\hat{\mu} \cdot s_i + b_i + \theta_i} = \sum_{i=1}^{N} s_i \tag{2.19}$$

and

$$\frac{n_i}{\mu \cdot s_i + b_i + \hat{\theta}_i} = 1 + \sum_{j} V_{ij}^{-1} \theta_j \tag{2.20}$$

where equation 2.19 and 2.20 are maximised with respect to μ and θ_i respectively.
We uses Newton-Raphson method (section 2.2) to solve for these system of equations numerically.
For a particular value of μ, we have 8 equation in $\boldsymbol{\theta}$, $\hat{\boldsymbol{\theta}}_{\boldsymbol{\mu}}$ is calculated by finding the root of these system of equations (given below in vector form)

$$F(\theta) = diag(\mu \cdot s + b + \theta)([1] + [V^{-1}][\theta]) - [n]$$

then this $\hat{\boldsymbol{\theta}}_{\boldsymbol{\mu}}$ is used to profile out $\hat{\mu}$
From figure 2.1 it is evident that calculated q_μ and published q_μ for SL framework are in great agreement. MSE in this case is 0.0007. Figure 2.1 also shows the comparison of full likelihood and simplified likelihood. MSE in this case is 0.12. From figure 2.2, we see that ratio plot for q_μ follow the line $y = 1$ closely. Only exception is in the middle around μ=0.71, here both calculated and published values reaches 0, hence we see deviations from the expected line. Also from the figure 2.1, it can be seen that SL framework does a good job in approximating the full likelihood.

Chapter 3

CheckMATE

3.1 Introduction

It has been more than 13 years since the LHC (Large Hadron Collider) has been operational. Since then, an enormous amount of data has been collected. Due to limiting computational and man power resources to determine the efficiency against analysis cuts for each new model using full detector simulation, experimental collaborations cannot interpret data for every BSM model. One way to investigate other BSM models is to follow the same procedures that have been used by experimental collaborations to perform the corresponding search.
CheckMATE [13] is a tool that facilitates the user to reinterpret limits set by experimental collaborations or test their own BSM model. It is written in C++ and python. The idea is to replicate the effects of all selection cuts applied to real data by using detector simulation applied to Monte Carlo data followed by same selection cuts. Once the analysis is validated against the published data; then user can pass MC events corresponding to the BSM model under consideration, generated using any modern MC event generators.

3.2 Process flow

1. Monte-Carlo event file and the corresponding cross-section for the given BSM physics process under consideration is provided by the user. The choice of event generator is up to the user as long as LHE [3] or HepMC

[14] output is available.

2. Events are passed to a detector simulator (Delphes [12] in this case), which is tuned to resemble the ATLAS and CMS detector. Delphes smears particle momenta runs jet algorithm to match jets, performs τ/b-tagging of the jets, and reconstructs missing transverse energy vector (definition of these reconstructed objects are defined in Section 2.6.1). The results of this process are stored in a ROOT [5] tree, which can further be modified if required.

3. Process all event to check for overlap removal of objects, final state object selections, and apply the isolation criteria required by the signal regions. Then events are catorised in different signal regions. Events in the signal regions are stored both as actual number of events and also normalized as per the given cross-section.

4. Finally the statistical significance of the result is evaluated by calculating likelihood or compairing to 95% upper limits.

Users can select from several analyses in the appendix for testing their model. New analysis can be added to CheckMATE using the AnalysisManager [16].

3.3 Statistical evaluation

The CheckMATE calculate the expected events (S) in a signal region using the process mentioned above. The uncertainties on S are calculated using the number of MC events (statistical uncertainty) and error on the cross-section as provided by the user (systematic uncertainty). Let us call total uncertainty to be ΔS. These numbers are compared with the given experimental upper limit. This comparison is performed using two methods.

- Several experimental searches give 95% upper limit value for their signal numbers. Then a quick way of determining whether the model is excluded or not can be found as-

$$r = \frac{(S - 1.96\Delta S)}{S^{95\%}_{Exp}}$$

"$1.96\Delta S$" signifies the 95% coverage of the values under 1.96 standard deviation of the mean.
So if value of r ≥ 1, model is said to be excluded at 95% confidence limit.

- CheckMATE can also explicitly provide the confidence level for the given signal region if a user wishes. CheckMATE uses a log-likelihood ratio test with CL_s method to calculate the confidence limit for the expected number of events in the signal region.

3.4 Usage

CheckMATE can be downloaded from the official site and can be setup using:

```
$ wget http://www.hepforge.org/archive/checkmate/
    ↪ CheckMATE-Current.tar.gz
$ tar -xvf CheckMATE-Current.tar.gz
$ cd CheckMATE-X.Y.Z
$ $./configure
$ make
```

[1] After successful installation, executables can be found inside the `bin` folder.CheckMATE can be directly run in the terminal by providing the events file, cross-section, and other information. However, it is more convenient to enter all this info in a text file, and passing it to the CheckMATE directly.

```
$ ./CheckMATE file.dat
```

Here is a list of mandatory inputs in the file to run an analysis.

- **Name of the run**: This creates a unique folder with the same name in the results directory, where all the results for the corresponding run will be kept.
- **Name of the analyses**: User has to provide one or more analyses against which the input data will be tested.
- **Events file**: User has to provide at least one input file(either in .hep or .hepmc or .lhe format).

[1] a more detailed installation instruction can be found on the official CheckMATE site.

- **Processes**: If more than one events file is used, they should be sorted into processes. i.e., CheckMATE should know whether input files are generated for the same process or a different one.
- **Total cross-section and error**: Cross-section of the generated process has to be provided in order to normalize the number of events in the signal regions as per the luminosity of the experimental analyses.

3.5 Results of the run

A successful run of the analyses will create a folder by the name of run in the results directory. Upon opening the folder, we see several other folders inside.

- **analysis**: This folder contains the result files of all the analyses that have acted on the delphes ROOT file. There are mainly three types of files that are generated for each analysis.
  - `Cutflow` file: This file contains the number of events that pass each applied cut. This file is useful for the user to check the efficiency of each cut in the analysis.
  - `Signal` file: This file contains the number of signal events in each signal region. It gives the sum of weights (`Sum_W`) and the sum of squared weights (`Sum_W2`) of all events in a signal region. It also gives a normalized number (`N_Norm`) of events in each signal region, calculated using cross-section and luminosity of the process.
  - `Log` file: This file contains any standard outputs or error/warning messages generated during the run. It has a naming convention as `analysisstdout_analysisname.log`
- **evaluation**: This folder contains results in more elaborate form and also contains statistical evaluation of the results. The type of files in the evaluation folder is as follows.
  - **Event process file**: This file has a general name of `[process]_eventResults.txt` and contains the result of all the signal regions from all the analyses for the given process. Data in each signal region is stored in normalized form. It also contains a column for the error in each signal region. This signal error is calculated as a quadratic sum of both statistical and systematic errors.

– **Process results file**: This file has a general name of `[process]_processResults.txt`, this file is the same as the event process file except for different events files for the same process are also combined. They are combined using proper weights from different events file results. If any signal region has 0 events, its statistical error is replaced by the statistical error of 1 event.

– **Total result file**: Results from different processes are further combined to get signal events in each analysis. signal_errors are added in quadrature as they are independent. This expected signal event for each signal region is then compared with experimental data using two methods discussed above. This comparison result(r-value) is stored along with expected and observed $S^{95\%}$ limits in total_results.txt file.

– **Best signal regions file**: This file contains the best signal region of each analysis, i.e., the signal region which has the highest expected sensitivity. This corresponds to the signal region, which has the largest rexpcons value.

3.6 AnalysisManager

CheckMATE facilitates the re-interpretation of experimental searches, the way this is achieved had already been discussed in greater detail in Chapter 3. The main ingredient for this re-interpretation in CheckMATE is the analyses against which the new models are tested. These analyses resemble the prescription used by experimental collaborations for the search. These analyses and their dependencies are stored in several specific locations within CheckMATE.
AnalysisManager offers a quick and easy way to add a new analysis, and it can also be used to modify an already existing analysis. It provides great convenience in managing and storing all the information necessary to run an analysis at their appropriate locations.
In order to add a new analysis, the user has to run the AnalysisManager and provide some information regarding the analysis.

- Some general information has to be provided, like the name of the analysis, name of the author, and email address to keep track of the analysis.

- After that, some questions regarding the analysis have to be answered, like which detector it belongs to, i.e., ATLAS [1] or CMS [2], the integrated

luminosity, and the energy of pp collision to normalize the events.

- Properties of the final state particles have to be provided for Delphes to work as closely as with experimental analysis. Options like leptons isolations, photons isolation criteria, jet flavor tagging have to be entered.

- Finally name of all the signal regions and the number of observed events, expected background events, and errors have to be provided. These pieces of information are usually available in the corresponding experimental publication.

Once all this information is provided to AnalysisManager, it creates the backbone of the analysis, which can be compiled. Now what remains is to define the analysis prescription as provided by the publication in the form of C++ code. CheckMATE has several inbuilt features and functions that make it easy to recast the new LHC analysis.

3.6.1 Final state objects

CheckMATE reconstructs all the final state objects using as per the LHC standards. All these objects have been highly tuned to match the output of both CMS and ATLAS detector.

- **Electrons** Electrons are reconstructed on all the three ATLAS definitions. Loose, medium, tight. All of these are stored in a C++ vector. These three definitions are based on p_t and η. By default, soft electron isolation is always applied such that in a cone of ΔR=0.2, the sum of p_T of other calorimeter objects should be less than 0.2 of electron p_T

- **Muons** CheckMATE uses two methods for the reconstruction of muons. Namely `muonCombined` and `muonsCombinedPlus`, both of which are functions of η and ϕ

- **Jets** In the case of ATLAS, jets are reconstructed with smeared calorimeter deposits, and in the case of CMS, they are reconstructed with particle-flow information. Jets are constructed using the anti-kt algorithm with a cone sizeΔR=0.4, known as the AK4 jet. Other definitions for jets can also be used. FastJet [10] functions are also accessible and can be used to check for constituents, n-subjettiness ratio to further study the jets.

- **Photons** Two definitions for photons are available they are photonsLoose and photonsMedium.
- **Taus** If the momentum vector of a tau decay matches that of a reconstructed jet, that jet is considered a tau lepton. Based on the number of reconstructed tracks, they are further divided into either 1-prong or multi-prong. Here also, both of these categories are reconstructed as loose, medium, and tight taus, which are functions of p_T and η
- **B-jets** Each jet is checked for their overlap with b-quark and c-quark; if none of them is true, it is assumed to originate from light quark. The probability of these three jet types to be tagged as b-jet is different depending on p_t and η. The user has to provide the probability for tagging b-quark. The remaining two tags are automatically calculated and implemented.
- **Missing energy** ATLAS reconstructs missing energy as the sum of smeared calorimeter objects, and CMS uses the sum particle flow objects. In order to overcome the pileup effect, an extra smearing factor is used. This smearing is done by adding a vector with uniformly random and direction magnitude, which follows a Gaussian distribution with width 20 GeV centered around 0.
- **Track and Calorimeter cells** Information about all reconstructed tracks and calorimeter cells are available within Checkmate, if required to develop more complex analysis.

3.6.2 Final state functions

CheckMATE has several inbuilt functions to make analysis implementation easier.

- **Isolation** Isolation criteria can isolate objects based on conditions from their nearby objects.
- **filterPhaseSpace** This function is used to select final state objects within a specific range of p_t and η as required by the publication.
- **Object Removal** This function is used to remove objects within the vicinity of another object. For example, electrons can also be double-counted in jets because of both deposit energy in the calorimeter. Hence

this overlap should be removed. The user has to provide the cone radius and object to be removed.

- **countSignalEvent** As the name suggests, this function counts the number of events in a signal region. In the analysis, wherever this function is used, if an event passes beyond this point, then it will be counted.
- **countCutFlowEvent** This inbuilt function counts the number of events after applying a particular cut. This is mainly used for debugging purposes and has nothing to do with the actual analysis.

Chapter 4

CMS Monojet search

4.1 Introduction

Several Astronomical phenomena have been observed regarding the presence of dark matter (DM). LHC has been very active in the search for dark matter particles, but till now no compelling evidence has been gathered regarding the non-gravitational interaction of Dark matter particles with SM particles. Many theories have been proposed in which DM particles interact with SM particles with sufficient strength. Most DM models assume the DM particles to consist of weakly interacting massive particles (WIMP). In this scenario, DM particles could may directly be produced in LHC collisions, if this is the case, these particles would not interact directly with the detector. However, due to momentum conservation in the transverse plane of collision, imbalance of p_T (missing transverse momentum) against the observable particles (jets) can serve as a good signature for DM candidates. These jets are formed as a result of fragmentation and hadronization of gluons and quarks. This is called the monojet search. One other possibility is that they can produce in association with electroweak boson. This results in the mono-V signal. Both of these final states have been extensively used by CMS to put new constraints at 95% CL on allowed models.

We implemented a CMS monojet analysis [18] that was performed with an integrated luminosity of 12.9 fb^{-1} at the center of mass energy of 13TeV. It is a Dark Matter search with jets and missing energy. Following the CheckMATE

notation, we will refer to this analysis as `cms1703_01651`. This implementation will help other users to check different models that involves monojet signatures.

Implementation of this analysis was done as per the steps discussed in section 3. For validation, a simulated process of proton-proton collision going to jets and Z-boson (where Z-boson further decays to muons) was used. This simulated sample was generated using MadGraph [4].

4.2 Final state objects required for the analysis

- **Electrons** are reconstructed as per the criteria discussed in 3.6.1. Their isolation criteria involve that transverse momenta of particles within a cone of ΔR=0.3 should be less than 14% of the electron's transverse momentum. This criterion is inserted when tuning the delphes settings at the start.
- **Muons** are reconstructed as per the criteria discussed in 3.6.1. Their isolation criteria involve that transverse momenta of all the particles within a cone of ΔR=0.4 should be less than 25% of the muon's transverse momentum. Another isolation criteria for muons(to be used in control regions) is also required where transverse momenta of particles within a cone of ΔR=0.4 should be less than 15% of the muon's transverse momentum.
- **Photons** are reconstructed based on ECAL shower shape
- **Jets** Two types of jets are used for signal identification.
  - **AK4 jets** These jets are constructed using anti-kt algorithm with cone radius ΔR=0.4 in fastjet.
  - **AK8 jets** These jets are constructed using anti-kt algorithm with cone radius ΔR=0.8 in fastjet.
- **b-jets** b-jet tagging probability is taken to be 60%. b-jets are used to remove top quark background.
- **Taus** Jets are tagged as taus if their mass matches that of a hadronically decaying tau lepton.

- **Missing transverse momentum vector($\overrightarrow{p}_t^{miss}$)** This is constructed as the negative vector sum of the transverse momenta (p_T) of all the particle-flow candidates in an event. Its magnitude is denoted by E_T^{miss}

Variable	Mono-V requirement	Monojet requirement
E_T^{miss}	> 250 GeV	> 200 GeV
Leading AK4 jet p_T	100	
Leading AK4 jet $\|\eta\|$	< 2.5	
Energy fraction of charged hadron in leading AK4 jet	> 0.1	
Energy fraction of neutral hadron in leading AK4 jet	< 0.8	
Number of muons ($p_T > 10$ GeV, $\|\eta\| < 2.4$)	0	
Number of electrons ($p_T > 10$ GeV, $\|\eta\| < 2.5$)	0	
Number of photons ($p_T > 15$ GeV, $\|\eta\| < 2.5$)	0	
Number of b jets ($P_T > 15$ GeV, $\|\eta\| < 2.4$)	0	
$\Delta\phi$ between four highest p_T jets and E_T^{miss}	>0.5 radians	
Leading AK8 jet p_T	> 250 GeV	
Leading AK8 jet η	< 2.4	Fails any of the
Leading AK8 jet τ_2/τ_1	< 0.6	mono-V criteria
Leading AK8 jet mass (m_J)	$65 < m_J < 105$ GeV	

Table 4.1: Cut-flow table for `cms1703_01651`

4.3 Jet clustering using anti-kt algorithm

Jets are a collimated spray of particles formed by the hadronization of a parton in a collision. A Jet clustering algorithm plays a vital role in the study of hadron hadron collisions in a collision experiment. It uses calorimetry and tracking information to reconstruct jets. For this analysis, we are using anti-kt algorithm for jets reconstruction.
Steps involved in clustering a jet using anti-kt algorithm [9] is as follows.

1. For each pair of stable final state particles, distance d_{ij} is defined as

$$d_{ij} = \Delta R_{ij}^2 / R^2$$

$$d_{iB} = 1$$

Here d_{iB} is the beam distance for particle i. $\Delta R_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2$, where y_i and ϕ_i are rapidity and azimuthal angle of the particle i. R is the jet radius parameter, 0.4 in the case of AK4 and 0.8 in the case of AK8 jets.

2. If the minimum of both these distances is d_{ij} then merge particle i and j by summing there four momenta, if the minimum is d_{iB} then particle i is declared to be final jet and is removed from the list.

3. Repeat step 1 and 2 until the stable particle list goes empty.

4.4 Jet pruning

Pruning [15] of jets removes the soft constituents and helps with better jet mass resolution. It is used for fat jets to ensure that mass reconstruction of the V boson is accurate. Jet's constituents are reclusterd using some sequential recombination algorithms. Soft constituents are removed in every recombination step, thus improving the jet mass resolution.
Steps involved in jet pruning are as follows.

1. jet's constituents are taken into a list and parameters `R_cut_factor` and `Z_cut` are defined.

2. Cambridge–Aachen algorithm is used on the list to recluster the jet, keeping in mind these two conditions in each recombination step.

   - One of the p_T^i and p_T^j is $< Z_{cut} * p_T^{i+j}$
   - The geometric distance between i and j is larger than the parameter R_{cut}, with R_{cut} = `R_cut_factor`$\times 2\mathrm{m}/p_T$, where m and pt are the mass and transverse momentum of the original jet being pruned.

3. When both above conditions are satisfied in the process, clustering is done by discarding the softer of the two, while continuing to include the harder one.

4. The resulting jet is the pruned jet.

4.5 N-subjettiness

Decay of highly boosted electroweak bosons or top quarks produces collimated spray of hadrons, which jet reclustering algorithim reconstruct as single jets. Thus these jet reconstruction become ineffective due to large background of ordinary QCD jets. N-subjettiness [19] variable tells us that to what degree, a jet can be regarded to be composed of N subjets.
Steps involved in calculating n-subjettiness variable are as follows.

1. N candidate subjets are choosen using the process described in [19].

2. On these N subjets, subjettiness variable τ_N is calculated as

$$\tau_N = \frac{1}{d_0} \sum_k p_{T,k} min\{\Delta R_{1,k}, \Delta R_{2,k}, ...\Delta R_{N,k}\}$$

   Here, k runs over the constituent particles in a given jet, $p_{T,k}$ are their transverse momenta, and $\Delta R_{J,k} = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}$

   is the distance in the rapidity-azimuth plane between a candidate subjet J and a constituent particle k. The normalization factor d_0 is taken as

$$d_0 = \sum_k p_{T,k} R_0$$

   where R_0 is the characteristic jet radius used in the original jet clustering algorithm.

4.6 Event selection

At first a trigger of 100 GeV is applied on both E_T^{miss} and H_T^{miss} where H_T^{miss} is the magnitude of the vector p_T sum of jets reconstructed at the trigger level. Jets used here are required to have $p_T > 20$ GeV and $|\eta| < 5.0$. After this, all the cuts given in Table 4.1 are applied.
In case of dimuon control region, muon veto is not applied. A dimuon event must contain exactly two oppositely charged muons, each with $p_T > 10$ GeV. Events are vetoed if there is an additional muon or electron with $p_T > 10$ GeV. At least one of the two muons is required to have $p_T > 20$ GeV. In the dimuon control region, hadronic recoil is used as a proxy for E_T^{miss}, where hadronic recoil is defined as excluding leptons and photons from the calculation of E_T^{miss}.

Since the analysis does not provide a dedicated cutflow, we will use dimuon control region to validate our analysis.

4.7 Result

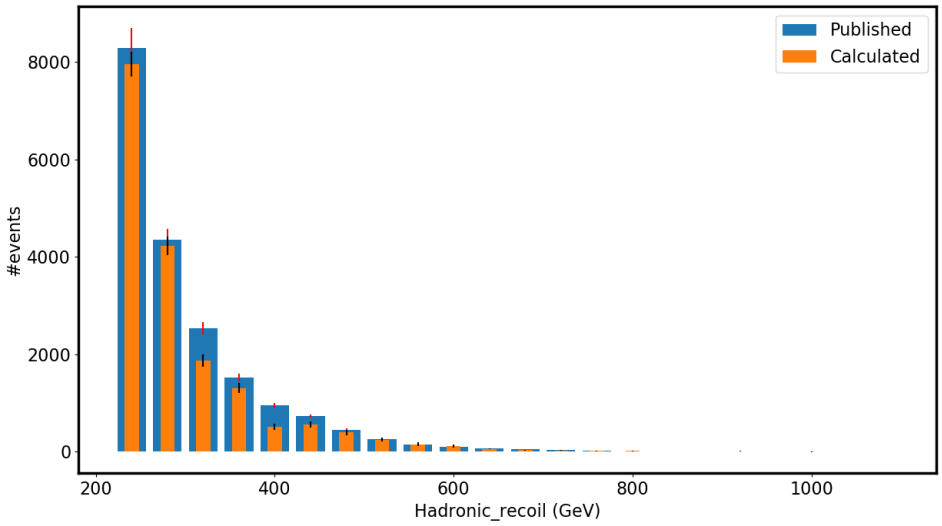


Figure 4.1: Comparison of number of events in monojet bins in the dimuon control region

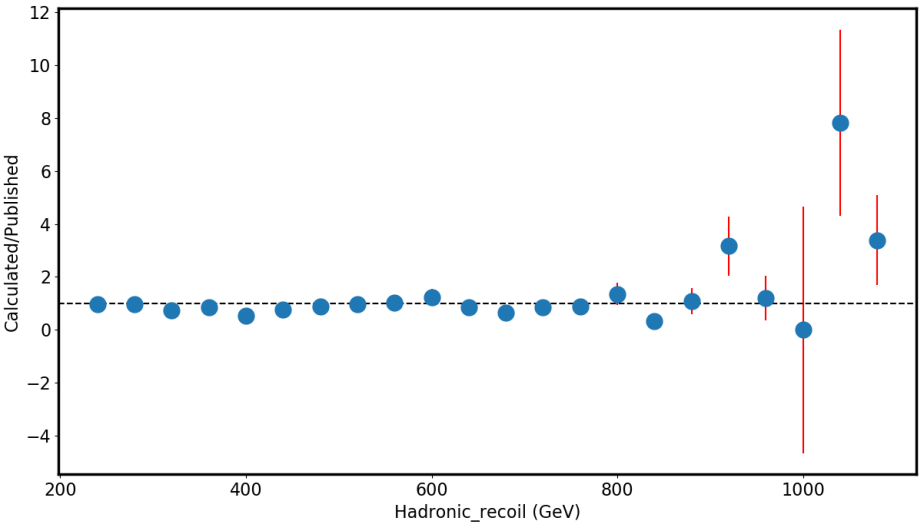


Figure 4.2: Ratio of number of events in each bin as given by the CheckMATE and publication

4.8 Discussion

Dimuon control sample was generated using MadGraph, a total of 100k events were generated, cross section of the process is 62.864 pb. Comparison between the events number as returned by CheckMATE and published one can be seen in figure 4.1 and 4.2. We see that our analysis implemented in CheckMATE is consistent with published number.
Slight mismatch in the later bins are because we need better MC sample at high jet p_T, which we could not generate with enough accuracy.

Hadronic recoil (GeV)	CMS	CheckMATE
200-230	8286	7956.66±254.29
230-260	4353	4226.21±185.33
260-290	2532	1869.29±123.25
290-320	1524	1308.55±103.12
320-350	948	512.02±64.50
350-390	736	560.78±67.51
390-430	456	398.24±56.90
430-470	256	251.94±45.25
470-510	148	154.42±35.42
510-550	100	121.91±31.47
550-590	68	56.90±9.01
590-640	55	35.55±7.12
640-690	33	28.44±6.36
690-740	19.5	17.07±4.92
740-790	11.5	15.64±4.72
790-840	8.5	2.84±2.01
840-900	6.6	7.11±3.18
900-960	3.6	11.37±4.02
960-1020	2.4	2.84±2.01
1020-1090	1.4	0±6.51
1090-1160	0.91	7.11±3.18
>1160	1.68	5.69±2.84

Table 4.2: Number of events in each bin

Chapter 5

Applying SL framework to monojet search

5.1 Dark matter with axial vector mediator

Many simplified DM models assume the DM particles to be Dirac fermions which interact with SM particles via spin-0 or spin-1 mediators. Based on the mediator, these interactions are classified to be of four types, Vector mediator, Axial-vector mediator, scalar and pseudoscalar mediator. We will use the simplified DM model in which interaction between DM and SM particles is mediated by spin-1 mediator such as Z' boson to generate the signal for mediator mass 1.6 GeV. We choose this specific model because CMS has published the test statistic value for this model using simplified likelihood. Constraint on the mediator mass can be put using the simplified likelihood method.

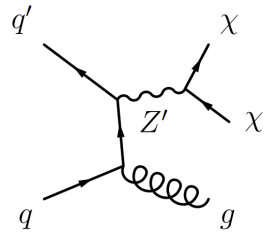


Figure 5.1: Dark photon + mono-jet

5.2 Generating signal events

Monojet signal (spin-1 mediator) were generated using MadGraph with `DMsimp_s_spin1` [6] model. Coupling constant of dark matter mediator with SM quarks (g_q) is taken to be 0.25 and with dark matter (g_D) is taken to be 1. Mediator and dark matter masses are set to be 1.6 TeV and 1 GeV respectively. These events were then passed to CheckMATE in order to apply the cuts as per analysis `cms1703_01651`. Events returned by CheckMATE are given in the table below.

Table 5.1: Number of different events in each bin

E_T^{miss}	Signal events	Background	Observed events
200-230	152.89	132100 ±4000	140642
230–260	110.27	69900 ±2200	73114
260–290	82.42	36800 ±1100	38321
290–320	83.55	20780 ±630	21417
320–350	61.38	12340 ±400	12525
350–390	64.23	9160 ±320	9515
390–430	48.31	5100 ±200	5174
430–470	31.26	3000 ±120	2947
470–510	25.57	1763 ±79	1777
510–550	16.48	1032 ±48	1021
550–590	21.03	683 ±37	694
590–640	14.77	554 ±28	554
640–690	11.36	322 ±19	339
690–740	10.23	178 ±13	196
740–790	2.84	122 ±8	123
790–840	2.27	74.5 ±6.6	80
840–900	1.70	77.8 ±7.2	68
900–960	3.98	38.4 ±4.8	37
960–1020	0.00	25.1 ±3.4	23
1020–1090	2.84	21.1 ±3.0	12
1090–1160	1.70	9.4 ±1.9	7
>1160	4.54	22.6 ±3.0	26

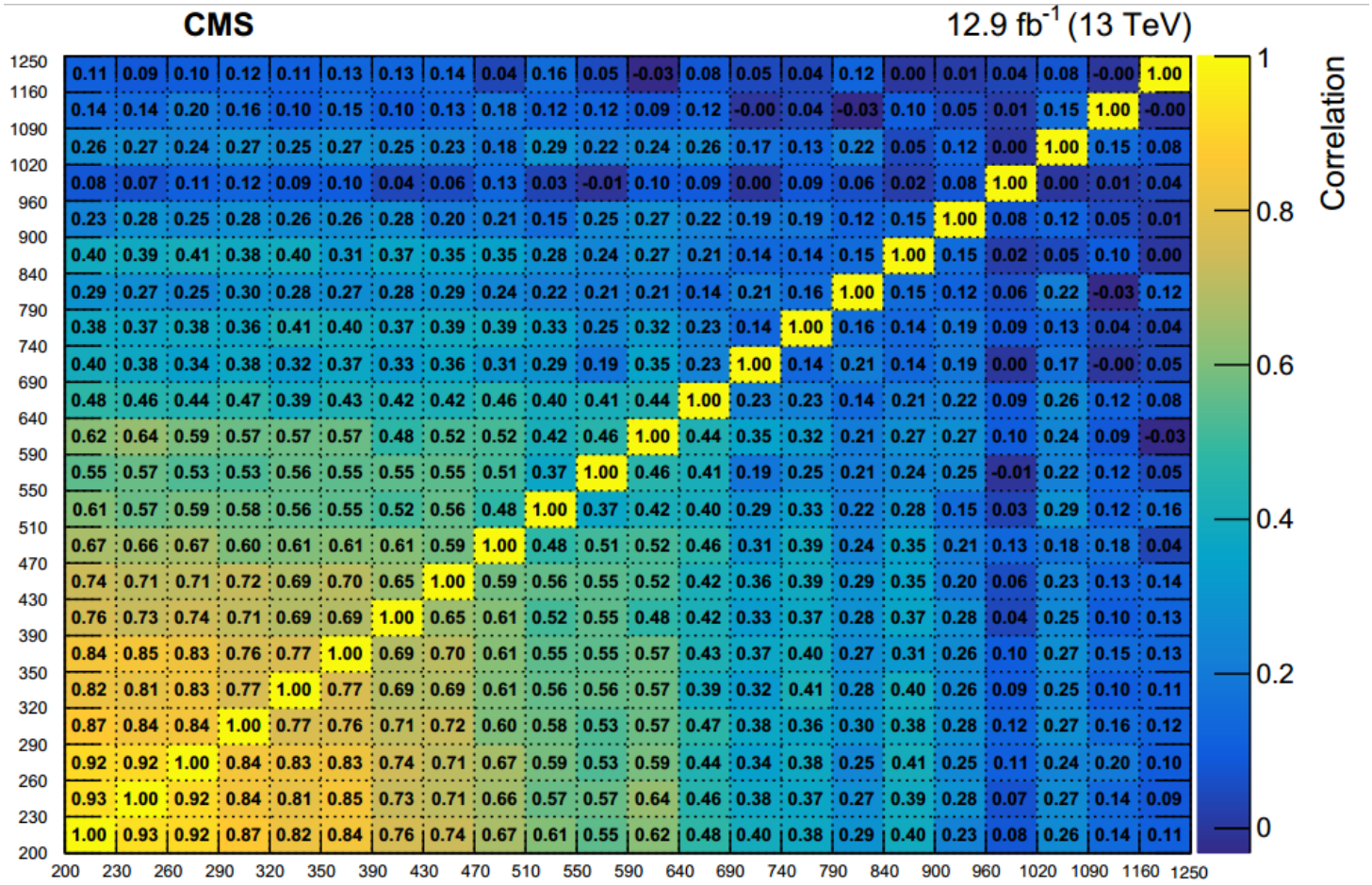


Figure 5.2: Correlation matrix across different bins

5.3 Result

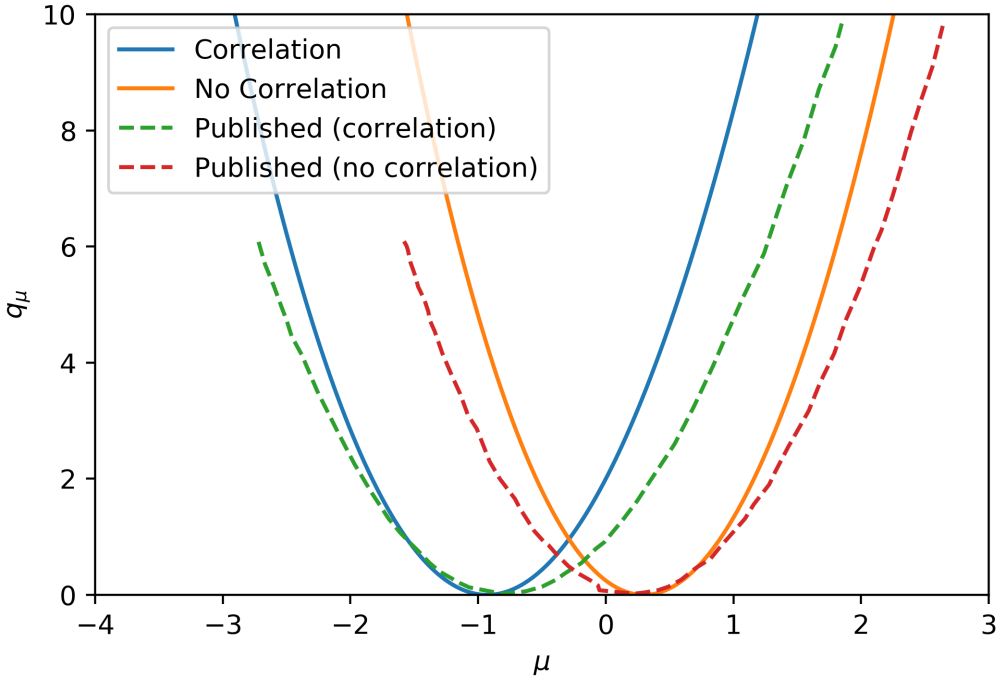


Figure 5.3: Test statistic (q_μ) constructed using the SL framework, both cases of correlation and no correlation across background has been considered

5.4 Discussion

Data provided in table 5.1 and correlation matrix (from which covariance matrix was generated using the standard deviation of background contribution in each bin) was used to maximize the log likelihood equation 2.3 as discussed in section 2.4.

Figure 5.3 gives the maximum likelihood estimate of signal strength modifier (μ) and nuisance parameter (θ). For a particular value of μ, test statistic (q_μ) can be used to check whether the μ is excluded or not with some confidence α. We can also set 95% confidence level upper limit on μ using this test statistic.

Bibliography

[1] Georges Aad et al. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 1–29.

[2] Roman Adolphi et al. "The CMS experiment at the CERN LHC". In: *Jinst* 803 (2008), S08004.

[3] J. Alwall et al. "A standard format for Les Houches Event Files". In: *Computer Physics Communications* 176.4 (Feb. 2007), pp. 300–304. DOI: 10.1016/j.cpc.2006.11.010. URL: https://doi.org/10.1016%2Fj.cpc.2006.11.010.

[4] Johan Alwall et al. "MadGraph 5: going beyond". In: *Journal of High Energy Physics* 2011.6 (June 2011). DOI: 10.1007/jhep06(2011)128. URL: https://doi.org/10.1007%2Fjhep06%28201129128.

[5] Ilka Antcheva et al. "ROOT—A C++ framework for petabyte data storage, statistical analysis and visualization". In: *Computer Physics Communications* 182.6 (2011), pp. 1384–1385.

[6] Mihailo Backović et al. *Higher-order QCD predictions for dark matter production at the LHC in simplified models with s-channel mediators*. 2015. DOI: 10.48550/ARXIV.1508.05327. URL: https://arxiv.org/abs/1508.05327.

[7] Adi Ben-Israel et al. "A Newton-Raphson method for the solution of systems of equations". In: *J. Math. Anal. Appl* 15.2 (1966), pp. 243–252.

[8] Andy Buckley et al. "The simplified likelihood framework". In: *Journal of High Energy Physics* 2019.4 (Apr. 2019). DOI: 10.1007/jhep04(2019)064. URL: https://doi.org/10.1007%2Fjhep04%282019%29064.

[9] Matteo Cacciari, Gavin P Salam, and Gregory Soyez. "The anti-ktjet clustering algorithm". In: *Journal of High Energy Physics* 2008.04 (Apr. 2008), pp. 063–063. DOI: 10.1088/1126-6708/2008/04/063. URL: https://doi.org/10.1088%2F1126-6708%2F2008%2F04%2F063.

[10] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. "FastJet user manual". In: *The European Physical Journal C* 72.3 (Mar. 2012). DOI: 10.1140/epjc/s10052-012-1896-2. URL: https://doi.org/10.1140%2Fepjc%2Fs10052-012-1896-2.

[11] The CMS Collaboration. *Simplified likelihood for the re-interpretation of public CMS results*. Tech. rep. Geneva: CERN, Jan. 2017. URL: https://cds.cern.ch/record/2242860.

[12] J De Favereau et al. "DELPHES 3: a modular framework for fast simulation of a generic collider experiment". In: *Journal of High Energy Physics* 2014.2 (2014), pp. 1–26.

[13] Daniel Dercks et al. "CheckMATE 2: From the model to the limit". In: *Computer Physics Communications* 221 (Dec. 2017), pp. 383–418. DOI: 10.1016/j.cpc.2017.08.021. URL: https://doi.org/10.1016%2Fj.cpc.2017.08.021.

[14] M Dobbs and J B Hansen. *The HepMC C++ Monte Carlo Event Record for High Energy Physics*. Tech. rep. revised version number 1 submitted on 2001-02-27 09:54:32. Geneva: CERN, June 2000. URL: http://cds.cern.ch/record/684090.

[15] Stephen D. Ellis, Christopher K. Vermilion, and Jonathan R. Walsh. "Recombination algorithms and jet substructure: Pruning as a tool for heavy particle searches". In: *Physical Review D* 81.9 (May 2010). DOI: 10.1103/physrevd.81.094023. URL: https://doi.org/10.1103%2Fphysrevd.81.094023.

[16] Jong Soo Kim et al. *A framework to create customised LHC analyses within CheckMATE*. 2015. DOI: 10.48550/ARXIV.1503.01123. URL: https://arxiv.org/abs/1503.01123.

[17] "Procedure for the LHC Higgs boson search combination in Summer 2011". In: (Aug. 2011).

[18] Albert M Sirunyan et al. "Search for dark matter produced with an energetic jet or a hadronically decaying W or Z boson at $\sqrt{s} = 13$ TeV". In: *JHEP* 07 (2017), p. 014. DOI: 10.1007/JHEP07(2017)014. arXiv: 1703.01651 [hep-ex].

[19] Jesse Thaler and Ken Van Tilburg. "Identifying boosted objects with N-subjettiness". In: *Journal of High Energy Physics* 2011.3 (Mar. 2011). DOI: 10.1007/jhep03(2011)015. URL: https://doi.org/10.1007%2Fjhep03%282011%29015.

Appendix A

CheckMATE

A.1 Implemented dark matter search

```
#include "cms1703_01651.h"
#include "cms1703_test.h"
#include "cmath"
#include "fastjet/tools/Filter.hh"
#include "fastjet/contribs/Nsubjettiness/Nsubjettiness
   ↪ .hh"
#include "fastjet/tools/Pruner.hh"

// AUTHOR: Avinash Verma
//  EMAIL: avinash.verma@students.iiserpune.ac.in
void Cms1703_01651::initialize() {
  setAnalysisName("cms1703_01651");
  setInformation(""
    "# Search for dark matter produced with an
       ↪ energetic\n"
    "# jet or ahadronically decaying W or Z boson\n"
  "");
  setLuminosity(12.9*units::INVFB);
  bookSignalRegions("MONOJ230_260;MONOJ260_290;
     ↪ MONOJ290_320;MONOJ320_350;MONOJ350_390;
     ↪ MONOJ390_430;MONOJ430_470;MONOJ470_510;
```

```
      ↪ MONOJ510_550;MONOJ550_590;MONOJ590_640;
      ↪ MONOJ640_690;MONOJ690_740;MONOJ740_790;
      ↪ MONOJ790_840;MONOJ840_900;MONOJ900_960;
      ↪ MONOJ960_1020;MONOJ1020_1090;MONOJ1090_1160;
      ↪ MONOJ1160;MONOV250_300;MONOV300_350;
      ↪ MONOV350_400;MONOV400_500;MONOV500_600;
      ↪ MONOV600_750;MONOV750");
18    // You can also book cutflow regions with
      ↪ bookCutflowRegions("CR1;CR2;...").  Note that
      ↪ the regions are
19    //  always ordered alphabetically in the cutflow
      ↪ output files.
20
21    // You should initialize any declared variables here
22  }
23
24  void Cms1703_01651::analyze() {
25    // Your eventwise analysis code goes here
26    // The following objects are always defined unless
      ↪ they are 'ignored' above. They form std::
      ↪ vector objects of the respective Delphes class
      ↪  type (except for Etmiss which is a single
      ↪ object)
27    // All std::vector members and etmiss have the
      ↪ common properties PT, Eta, Phi and P4() with
      ↪ the latter giving access to the full ROOT
      ↪ TLorentzVector.
28    // Within a std::vector, all members are ordered
      ↪ with highest pt coming first.
29
30    // electronsLoose, electronsMedium, electronsTight
      ↪   are list of electrons that passed respective
      ↪  efficiency and reconstruction cuts
31    // muonsCombinedPlus, muonsCombined
      ↪                  as above for muons
32    // photonsMedium
```

```
    ↪                                          as above
    ↪ for photons
33  // jets are all reconstructed jets
    ↪                    as above for jets. Note that
    ↪  electrons are most certainly also
    ↪ reconstructed as a jet -> overlap removal do
    ↪ avoid double counting necessary!
34  // tracks, towers
    ↪                                        calorimeter
    ↪  and tracker information. Usually not needed.
35  // missingET
    ↪                                             rec
    ↪ missing ET EXCLUDING muons.
36
37
38  // Here is a couple of useful functions and lines:
39  //------------Phase Space Cuts (defined for jets,
    ↪ electronsXYZ, muonsXYZ, photonsXYZ)
40  // jets = filterPhaseSpace(jets, 20., -2.8, 2.8)  //
    ↪  The vector 'jets' only contains jets with pt
    ↪ >= 20 GeV and -2.8 < eta < 2.8. This function
    ↪ is applicable to other particles too (
    ↪ electronsMedium, ... ).
41  // jets = overlapRemoval(jets, electronsLoose, 0.2)
    ↪ Removes all jets for which there exists any
    ↪ electron in 'electronsLoose' with deltaR <
    ↪ 0.2.
42  // jets = overlapRemovel(jets, 0.2) If two jets
    ↪ overlap within deltaR < 0.2, only the harder
    ↪ jet is stored.
43
44  //------------Isolation Checks (defined for
    ↪ electronsXYZ, muonsXYZ, photonsXYZ
45  //------------        For each object, if the user
    ↪ entered N isolation conditions, they can be
46  //------------        checked individually be the
```

```
    ↪ second argument (running from 0 to N-1).
47  // electronsMedium = filterIsolation(electronsMedium
    ↪ , 0)            Removes electrons that do not
    ↪ pass the first isolation condition entered
    ↪ into the AnalysisManager by the user
48  // std::vector<int> flags; flags.push_back(0); flags
    ↪ .push_back(2);
49  // electronsMedium = filterIsolation(electronsMedium
    ↪ , flags)        Same as above, but both the
    ↪ first and the third condition have to be
    ↪ fulfilled
50  // electronsMedium = filterIsolation(electronsMedium
    ↪ )               Same as above, but all
    ↪ conditions have to be fulfilled.
51
52  //-----------Flavour Tag Checks (defined for jets
    ↪ only)
53  //----------          Tau tags "loose", "medium" or
    ↪ "tight" can be individually checked if the
    ↪ user enabled tau tagging in the AM.
54  //----------          For b-tags, if N working
    ↪ points have been defined, the ith condition
    ↪ can be tested by putting i-1 into the second
    ↪ argument (if there is only one, the argument
    ↪ can be omitted)
55  // if checkTauTag(jets[0], "tight")
    ↪ leadingJetIsTagged = True;
56  // if checkBTag(jets[0], 0) leadingJetIsBTagged =
    ↪ True;
57
58
59  //-----------Auxiliary Information
60  // - Always ensure that you don't access vectors out
    ↪  of bounds. E.g. 'if(jets[1]->PT > 150)'
    ↪ should rather be if (jets.size() > 1 && jets
    ↪ [1]->PT > 150).
```

```
61    // - Use rand()/(RAND_MAX+1.) for random numbers
         ↪ between 0 and 1. The random seed is determined
         ↪  from system time or by the RandomSeed
         ↪ parameter in CheckMATE.
62    // - The 'return' statement will end this function
         ↪ for the current event and hence should be
         ↪ called whenever the current event is to be
         ↪ vetoed.
63    // - Many advanced kinematical functions like mT2
         ↪ are implemented. Check the manual for more
         ↪ information.
64    // - If you need output to be stored in other files
         ↪ than the cutflow/signal files we provide,
         ↪ check the manual for how to do this
         ↪ conveniently.
65
66    missingET->addMuons(muonsCombined);  // Adds muons
         ↪ to missing ET. This should almost always be
         ↪ done which is why this line is not commented
         ↪ out.
67    std::vector<Electron *> electrons_veto;
68    std::vector<Muon *> muons_veto;
69    std::vector<Photon *> photons_veto;
70    // construct tau
71    std::vector<Jet *> taus;
72    std::vector<Jet *> taus_veto;
73    std::vector<Jet *> jets20;
74    jets20 = filterPhaseSpace(jets, 20., -5.0, 5.0);
75    float jets20_ptx = 0;
76    float jets20_pty = 0;
77    for (int i = 0; i < jets20.size(); i++) {
78      jets20_ptx += jets20[i]->P4().Px();
79      jets20_pty += jets20[i]->P4().Py();
80    }
81    float jets20_pt = sqrt(jets20_ptx * jets20_ptx +
         ↪ jets20_pty * jets20_pty);
```

```
82
83    if (jets.size() > 0)
84    {
85      for (int i = 0; i < jets.size(); i++)
86      {
87        // if (checkTauTag(jets[i], "tight")) {
88        // if (checkTauTag(jets[i], "medium")) {
89        if (checkTauTag(jets[i], "loose") and fabs(jets[
              ↪ i]->Charge) == 1)
90        {
91          taus.push_back(jets[i]);
92        }
93      }
94    }
95    taus_veto = filterPhaseSpace(taus, 18., -2.3, 2.3);
96    // b-jet construction
97    std::vector<Jet *> bjets;
98    std::vector<Jet *> bjets_veto;
99    if (jets.size() > 0)
100   {
101     for (int i = 0; i < jets.size(); i++)
102     {
103       if (checkBTag(jets[i]))
104       {
105         bjets.push_back(jets[i]);
106       }
107     }
108   }
109   bjets_veto = filterPhaseSpace(bjets, 15., -2.4, 2.4)
        ↪ ;
110
111
112
113
114
115   std::vector<fastjet::PseudoJet> particles;
```

```
116   for (int i = 0; i < true_particles.size(); i++)
117   {
118     if (true_particles[i]->Status == 1)
119     {
120       int pid = true_particles[i]->PID;
121
122       particles.push_back(fastjet::PseudoJet(
          ↪ true_particles[i]->P4().Px(),
          ↪ true_particles[i]->P4().Py(),
          ↪ true_particles[i]->P4().Pz(),
          ↪ true_particles[i]->P4().E()));
123       particles.back().set_user_index(pid);
124     }
125   }
126   fastjet::JetDefinition jetAK4(fastjet::
       ↪ antikt_algorithm, 0.4);
127   fastjet::ClusterSequence AK4jet(particles, jetAK4);
128   std::vector<fastjet::PseudoJet> AK4 = sorted_by_pt(
       ↪ AK4jet.inclusive_jets());
129   //
       ↪ ////////////////////////////////////////////////////
       ↪
130
131   // another ak4 jet for charge index
132
133   std::vector<fastjet::PseudoJet> ak4_charge;
134   for (int i = 0; i < true_particles.size(); i++)
135   {
136     if (true_particles[i]->Status == 1)
137     {
138       int charge = true_particles[i]->Charge;
139
140       ak4_charge.push_back(fastjet::PseudoJet(
          ↪ true_particles[i]->P4().Px(),
          ↪ true_particles[i]->P4().Py(),
          ↪ true_particles[i]->P4().Pz(),
```

```
            ↪ true_particles[i]->P4().E()));
141         ak4_charge.back().set_user_index(charge);
142       }
143     }
144     fastjet::JetDefinition jetAK4_charge(fastjet::
            ↪ antikt_algorithm, 0.4);
145     fastjet::ClusterSequence AK4jet_charge(ak4_charge,
            ↪ jetAK4_charge);
146     std::vector<fastjet::PseudoJet> AK4_charge =
            ↪ sorted_by_pt(AK4jet_charge.inclusive_jets());
147
148     // constructing AK8 jets
149
150     std::vector<fastjet::PseudoJet> particles_AK8;
151     for (int i = 0; i < true_particles.size(); i++)
152     {
153       if (true_particles[i]->Status == 1)
154       {
155         int AK8_pid = true_particles[i]->PID;
156         particles_AK8.push_back(fastjet::PseudoJet(
                ↪ true_particles[i]->P4().Px(),
                ↪ true_particles[i]->P4().Py(),
                ↪ true_particles[i]->P4().Pz(),
                ↪ true_particles[i]->P4().E()));
157         particles_AK8.back().set_user_index(AK8_pid);
158       }
159     }
160     fastjet::JetDefinition jetAK8(fastjet::
            ↪ antikt_algorithm, 0.8);
161     fastjet::ClusterSequence AK8jet(particles_AK8,
            ↪ jetAK8);
162     std::vector<fastjet::PseudoJet> AK8 = sorted_by_pt(
            ↪ AK8jet.inclusive_jets());
163
164     // AK8_charge
165
```

```
166   std::vector<fastjet::PseudoJet> ak8_charge;
167   for (int i = 0; i < true_particles.size(); i++)
168   {
169     if (true_particles[i]->Status == 1)
170     {
171       int AK8_charge = true_particles[i]->Charge;
172       ak8_charge.push_back(fastjet::PseudoJet(
          ↪ true_particles[i]->P4().Px(),
          ↪ true_particles[i]->P4().Py(),
          ↪ true_particles[i]->P4().Pz(),
          ↪ true_particles[i]->P4().E()));
173       ak8_charge.back().set_user_index(AK8_charge);
174     }
175   }
176   fastjet::JetDefinition jetAK8_charge(fastjet::
       ↪ antikt_algorithm, 0.8);
177   fastjet::ClusterSequence AK8jet_charge(ak8_charge,
       ↪ jetAK8_charge);
178   std::vector<fastjet::PseudoJet> AK8_charge =
       ↪ sorted_by_pt(AK8jet_charge.inclusive_jets());
179
180   //
       ↪ ////////////////////////////////////////////////////////////
       ↪
181
182   electrons_veto = filterPhaseSpace(electronsLoose,
       ↪ 10., -2.5, 2.5);
183   muons_veto = filterPhaseSpace(muons, 10.);
184
185   photons_veto = filterPhaseSpace(photonsLoose, 15.,
       ↪ -2.5, 2.5);
186
187   // filtering AK4 with pt > 30
188   std::vector<fastjet::PseudoJet> AK4_filtered_pt30;
189   if (AK4.size() > 0)
190   {
```

```
191       for (int i = 0; i < AK4.size(); i++)
192       {
193         if (AK4[i].pt() > 30.)
194         {
195           AK4_filtered_pt30.push_back(AK4[i]);
196         }
197       }
198     }
199
200     // filtering AK4 jets with pt > 20 and eta<5
201     std::vector<fastjet::PseudoJet>
          ↪ AK4_filtered_pt20_eta5;
202
203     for (int i = 0; i < AK4.size(); i++)
204     {
205       if (AK4[i].pt() > 20. and fabs(AK4[i].eta()) < 5.)
206       {
207         AK4_filtered_pt20_eta5.push_back(AK4[i]);
208       }
209     }
210
211
212     //
          ↪ ////////////////////////////////////////////////////////////////
          ↪
213
214     // defining hadronic recoil
215
216     float sum_leptons_ptx = 0;
217     float sum_leptons_pty = 0;
218     float sum_leptons_ptz = 0;
219     float sum_leptons_E = 0;
220     float missET_x = 0;
221     float missET_y = 0;
222     float missET_z = 0;
223     float missET_E = 0;
```

```
224
225   if (electrons.size() > 0)
226   {
227     for (int i = 0; i < electrons.size(); i++)
228     {
229       sum_leptons_ptx += electrons[i]->P4().Px();
230       sum_leptons_pty += electrons[i]->P4().Py();
231       sum_leptons_ptz += electrons[i]->P4().Pz();
232       sum_leptons_E += electrons[i]->P4().E();
233     }
234   }
235   if (photons.size() > 0)
236   {
237     for (int i = 0; i < photons.size(); i++)
238     {
239       sum_leptons_ptx += photons[i]->P4().Px();
240       sum_leptons_pty += photons[i]->P4().Py();
241       sum_leptons_ptz += photons[i]->P4().Pz();
242       sum_leptons_E += photons[i]->P4().E();
243     }
244   }
245   if (taus.size() > 0)
246   {
247     for (int i = 0; i < taus.size(); i++)
248     {
249       sum_leptons_ptx += taus[i]->P4().Px();
250       sum_leptons_pty += taus[i]->P4().Py();
251       sum_leptons_ptz += taus[i]->P4().Pz();
252       sum_leptons_E += taus[i]->P4().E();
253     }
254   }
255   if (muons.size() > 0)
256   {
257     for (int i = 0; i < muons.size(); i++)
258     {
259       sum_leptons_ptx += muons[i]->P4().Px();
```

```
260         sum_leptons_pty += muons[i]->P4().Py();
261         sum_leptons_ptz += muons[i]->P4().Pz();
262         sum_leptons_E += muons[i]->P4().E();
263       }
264     }
265
266     // sum the missing components of true particles if
        ↪ size is greater than 0 and satus is 1
267     if (true_particles.size() > 0)
268     {
269       for (int i = 0; i < true_particles.size(); i++)
270       {
271         if (true_particles[i]->Status == 1)
272         {
273           missET_x += true_particles[i]->P4().Px();
274           missET_y += true_particles[i]->P4().Py();
275           missET_z += true_particles[i]->P4().Pz();
276           missET_E += true_particles[i]->P4().E();
277         }
278       }
279     }
280     float missET_pt = sqrtf(missET_x * missET_x +
        ↪ missET_y * missET_y);
281
282
283     float had_recoil_x = missET_x - sum_leptons_ptx;
284     float had_recoil_y = missET_y - sum_leptons_pty;
285     float had_recoil_z = missET_z - sum_leptons_ptz;
286     float had_recoil_E = missET_E - sum_leptons_E;
287     float had_recoil = sqrtf(had_recoil_x * had_recoil_x
        ↪  + had_recoil_y * had_recoil_y);
288
289     float hadx = -missingET->P4().Px() - sum_leptons_ptx
        ↪ ;
290     float hady = -missingET->P4().Py() - sum_leptons_pty
        ↪ ;
```

```
291   float hadz = -missingET->P4().Pz() - sum_leptons_ptz
         ↪ ;
292   float hadE = missingET->P4().E() - sum_leptons_E;
293   float hadronic_recoil = sqrtf(hadx * hadx + hady *
         ↪ hady);
294
295   // defining sum of muons pt components
296   float sum_muons_ptx = 0;
297   float sum_muons_pty = 0;
298
299   if (muons.size() > 0)
300   {
301     for (int i = 0; i < muons.size(); i++)
302     {
303       sum_muons_ptx += muons[i]->P4().Px();
304       sum_muons_pty += muons[i]->P4().Py();
305     }
306   }
307
308   float muons_pt = sqrtf(sum_muons_ptx * sum_muons_ptx
         ↪  + sum_muons_pty * sum_muons_pty);
309
310   /////veto//////
311
312   // trigger
313   if(jets20_pt < 120.)
314     return;
315
316   if (had_recoil < 200.0)
317   {
318     return;
319   }
320
321   if (photons_veto.size() != 0 || electrons_veto.size
         ↪ () != 0 || taus_veto.size() != 0 || bjets_veto
         ↪ .size() != 0)
```

```
322       return;
323
324     if (muons_veto.size() != 2)
325       return;
326     if (muons_veto[0]->Charge * muons_veto[1]->Charge >
          ↪ 0)
327       return;
328
329     if (muons_veto[0]->PT < 20.0 && muons_veto[1]->PT <
          ↪ 20.0)
330       return;
331
332     if ((muons_veto[0]->P4() + muons_veto[1]->P4()).M()
          ↪ < 60.0 || (muons_veto[0]->P4() + muons_veto
          ↪ [1]->P4()).M() > 120.0)
333       return;
334
335     if (AK4.size() > 0)
336     {
337       if (jets[0]->PT < 100.0 || fabs(jets[0]->Eta) >
            ↪ 2.5)
338         return;
339     }
340
341     std::vector<fastjet::PseudoJet> constituents = AK4
          ↪ [0].constituents();
342     std::vector<fastjet::PseudoJet> constituents_charge
          ↪ = AK4_charge[0].constituents();
343     // print user index of constituents
344
345     float photons_E = 0;
346     for (int i = 0; i < constituents.size(); i++)
347     {
348       if (constituents[i].user_index() == 22)
349
350       {
```

```
351         photons_E += constituents[i].E();
352       }
353     }
354 
355     float neutral_hadron_E = 0;
356     float charged_hadron_E = 0;
357 
358     for (int i = 0; i < constituents.size(); i++)
359     {
360       if (constituents_charge[i].user_index() == 0)
361       {
362         neutral_hadron_E += constituents_charge[i].E();
363       }
364       else
365       {
366         charged_hadron_E += constituents_charge[i].E();
367       }
368     }
369 
370     neutral_hadron_E -= photons_E;
371     float leading_jet_E = AK4[0].E();
372 
373     float fraction_neutral = neutral_hadron_E /
        ↪ leading_jet_E;
374     float fraction_charged = charged_hadron_E /
        ↪ leading_jet_E;
375 
376     if (fraction_neutral > 0.8)
377       return;
378 
379     if (fraction_charged < 0.1)
380       return;
381 
382 
383 
384 
```

```
385   bool AK8_mass = false;
386   bool AK8_charged_constituents = false;
387
388   fastjet::contrib::NsubjettinessRatio nSub21_beta1(2,
         ↪  1, fastjet::contrib::OnePass_KT_Axes(),
         ↪ fastjet::contrib::NormalizedMeasure(1.0, 0.8))
         ↪ ;
389
390   // leading AK8 mass
391   float rfact = 0.5;
392   float Zcut = 0.10;
393   fastjet::Pruner pruner(fastjet::cambridge_algorithm,
         ↪  Zcut, rfact);
394   fastjet::PseudoJet pruned_jet = pruner(AK8[0]);
395   float mass_pruned = pruned_jet.m();
396   // cout << mass_pruned << endl;
397   bool pruned_leading = false;
398   bool pruned_subjettiness = false;
399   if (mass_pruned > 65.0 && mass_pruned < 105.0)
400   {
401     AK8_mass = true;
402   }
403   if (pruned_jet.pt() > 250.0 && fabs(pruned_jet.eta()
         ↪ ) < 2.4)
404   {
405     pruned_leading = true;
406   }
407   float tau = nSub21_beta1(pruned_jet);
408   if (tau < 0.6)
409   {
410     pruned_subjettiness = true;
411   }
412
413   // cout << "pruned jet energery: " << pruned_jet.E()
         ↪  << endl;
414   // cout << "pruned jet transverse energy: " <<
```

```
    ↪ pruned_jet.Et() << endl;
415 // std::vector<fastjet::PseudoJet>
    ↪ constituents_pruned = pruned_jet.constituents
    ↪ ();
416 // int pruned_size = constituents_pruned.size();
417 // float tra_mass=0;
418 // for (int i = 0; i < constituents_pruned.size(); i
    ↪ ++)
419 // {
420 //   for (int j = 0; j < pruned_size; j++)
421 //   {
422 //     if (i != j)
423 //     {
424 //       float e1=constituents_pruned[i].Et();
425 //       float e2=constituents_pruned[j].Et();
426 //       float pt1=constituents_pruned[i].pt();
427 //       float pt2=constituents_pruned[j].pt();
428 //       float b1=pt1/e1;
429 //       float b2=pt2/e2;
430 //       float m1=constituents_pruned[i].m();
431 //       float m2=constituents_pruned[j].m();
432 //       float rap1=constituents_pruned[i].rap();
433 //       float rap2=constituents_pruned[j].rap();
434 //       float phi1=constituents_pruned[i].phi();
435 //       float phi2=constituents_pruned[j].phi();
436 //       tra_mass += m1*m1+m2*m2 + 2*e1*e2*(cosh(
    ↪ rap1-rap2) - b1*b2*cos(phi1-phi2));
437 //     }
438 //   }
439 // }
440 // float mass=sqrtf(tra_mass/2);
441 // cout<<"mass: "<<mass<<endl;
442 // cout<<"mass_pruned: "<<mass_pruned<<endl;
443
444 // check leading AK8 charged constituents ratio
445
```

```
446   std::vector<fastjet::PseudoJet> constituents_AK8 =
         ↪ AK8[0].constituents();
447   std::vector<fastjet::PseudoJet>
         ↪ constituents_charge_AK8 = AK8_charge[0].
         ↪ constituents();
448   float photons_E_AK8 = 0;
449   for (int i = 0; i < constituents_AK8.size(); i++)
450   {
451     if (constituents_AK8[i].user_index() == 22)
452
453     {
454       photons_E_AK8 += constituents_AK8[i].E();
455     }
456   }
457
458   float neutral_hadron_E_AK8 = 0;
459   float charged_hadron_E_AK8 = 0;
460
461   for (int i = 0; i < constituents_AK8.size(); i++)
462   {
463     if (constituents_charge_AK8[i].user_index() == 0)
464     {
465       neutral_hadron_E_AK8 += constituents_charge_AK8[
             ↪ i].E();
466     }
467     else
468     {
469       charged_hadron_E_AK8 += constituents_charge_AK8[
             ↪ i].E();
470     }
471   }
472
473   neutral_hadron_E_AK8 -= photons_E_AK8;
474   float leading_jet_E_AK8 = AK8[0].E();
475
476   float fraction_neutral_AK8 = neutral_hadron_E_AK8 /
```

```
      ↪ leading_jet_E_AK8;
477   float fraction_charged_AK8 = charged_hadron_E_AK8 /
      ↪ leading_jet_E_AK8;
478
479   if (fraction_neutral_AK8 < 0.8 &&
      ↪ fraction_charged_AK8 > 0.1)
480   {
481     AK8_charged_constituents = true;
482   }
483
484   if (pruned_leading && pruned_subjettiness &&
      ↪ AK8_mass && AK8_charged_constituents &&
      ↪ had_recoil > 250.0)
485   {
486     if (had_recoil > 250.0 && had_recoil < 300.0)
487       countSignalEvent("MONOV250_300");
488
489     if (had_recoil > 300.0 && had_recoil < 350.0)
490       countSignalEvent("MONOV300_350");
491
492     if (had_recoil > 350.0 && had_recoil < 400.0)
493       countSignalEvent("MONOV350_400");
494
495     if (had_recoil > 400.0 && had_recoil < 500.0)
496       countSignalEvent("MONOV400_500");
497
498     if (had_recoil > 500.0 && had_recoil < 600.0)
499       countSignalEvent("MONOV500_600");
500
501     if (had_recoil > 600.0 && had_recoil < 750.0)
502       countSignalEvent("MONOV600_750");
503
504     if (had_recoil > 750.0)
505       countSignalEvent("MONOV750");
506   }
507
```

```
508   else
509   {
510     //////signal regions ///////////
511     ////////////////////////////////
512     ////////////////////////////////
513
514     if (had_recoil > 230.0 && had_recoil < 260.0)
515       countSignalEvent("MONOJ230_260");
516     if (had_recoil > 260.0 && had_recoil < 290.0)
517       countSignalEvent("MONOJ260_290");
518     if (had_recoil > 290.0 && had_recoil < 320.0)
519       countSignalEvent("MONOJ290_320");
520     if (had_recoil > 320.0 && had_recoil < 350.0)
521       countSignalEvent("MONOJ320_350");
522     if (had_recoil > 350.0 && had_recoil < 390.0)
523       countSignalEvent("MONOJ350_390");
524
525     if (had_recoil > 390.0 && had_recoil < 430.0)
526       countSignalEvent("MONOJ390_430");
527
528     if (had_recoil > 430.0 && had_recoil < 470.0)
529       countSignalEvent("MONOJ430_470");
530
531     if (had_recoil > 470.0 && had_recoil < 510.0)
532       countSignalEvent("MONOJ470_510");
533
534     if (had_recoil > 510.0 && had_recoil < 550.0)
535       countSignalEvent("MONOJ510_550");
536
537     if (had_recoil < 590.0 && had_recoil > 550.0)
538       countSignalEvent("MONOJ550_590");
539
540     if (had_recoil < 640.0 && had_recoil > 590.0)
541       countSignalEvent("MONOJ590_640");
542
543     if (had_recoil < 690.0 && had_recoil > 640.0)
```

```
544         countSignalEvent("MONOJ640_690");
545
546       if (had_recoil < 740.0 && had_recoil > 690.0)
547         countSignalEvent("MONOJ690_740");
548
549       if (had_recoil < 790.0 && had_recoil > 740.0)
550         countSignalEvent("MONOJ740_790");
551
552       if (had_recoil < 840.0 && had_recoil > 790.0)
553         countSignalEvent("MONOJ790_840");
554
555       if (had_recoil < 900.0 && had_recoil > 840.0)
556         countSignalEvent("MONOJ840_900");
557
558       if (had_recoil < 960.0 && had_recoil > 900.0)
559         countSignalEvent("MONOJ900_960");
560
561       if (had_recoil > 960.0 && had_recoil < 1020.0)
562         countSignalEvent("MONOJ960_1020");
563
564       if (had_recoil < 1090.0 && had_recoil > 1020.0)
565         countSignalEvent("MONOJ1020_1090");
566
567       if (had_recoil > 1090.0 && had_recoil < 1160.0)
568         countSignalEvent("MONOJ1090_1160");
569
570       if (had_recoil > 1160.0)
571         countSignalEvent("MONOJ1160");
572     }
573
574 }
575
576 void Cms1703_01651::finalize() {
577   // Whatever should be done after the run goes here
578 }
```