CS 4632: Modeling and Simulation
Milestone 7: Validation and Verification

Casey Hampson (ID: 001079192)

16 April, 2025

Contents

1 Introduction

The previous milestone, Sensitivity and Scenerio Analysis, was concerned with testing the model's performance and behavior by varying its inputs. This milestone builds on top of the previous milestone by more rigorously comparing model outputs with that from other real-world models and results. Additionally, it seeks to ensure that the various components of the model are performing as expected and well enough within the entire program.

1.1 Disclaimer

As a disclaimer, I want to mention a few specific features of my particular model, something that I emailed you about recently. In particular, when it comes to input variation defining expectations for the model's behavior, my situation is a bit unique. I'll use the same analogy I used in the email: if I were programming something like a traffic simulation, for instance, there are a number of parameters that one can realistically vary and get perfectly viable results. These could be number of cars, structure of the roads, number of lanes, and so on. All of these would in principle lead to perfectly valid behavior that would be observed in the real world (I-75, at any point of the day, being a fantastic example of the maximum amount of traffic possible).

My model, however, is a bit different. Its entire formulation, in the physics sense, it based on a number of different theorems, assumptions, etc. in quantum physics that essentially restrict the number of inputs I can vary before the model becomes either not representative anymore of the process I want to simulate, or worse, not even mathematically valid in the first place (i.e. it invalidates certain assumptions present in the theory). Because of this, there is only so much I am able to do in terms of, like in the previous model, definitions of scenarios and whatnot, since there are really only two meaningful scenarios: either I provide a certain set of inputs that specify the process I want and are mathematically/physically valid, or I don't and the output should be considered bogus no matter if it breaks the model.

With regards to this model, there are of course purely statistical tests I can run, most notably on the Monte Carlo part of the simulation, along with the event generation side of things since those types of tests are related purely to the fact that there are a limited number of samples I am considering. On the other hand, trying to quantify *systematic* errors, which are what we physicists call errors associated with varying inputs and considering phenomena that could impact the particular process under study, is extremely challenging. Specifically in the case of my model, the best that I can do is quote research on errors associated with the underlying theorems, since I am unable to numerically run these tests as I described earlier. Hopefully I have plead my case well enough; I know that this is my own fault for choosing such an incompatible model to simulate for this class, but it is of course too late to change now, so I just have to settle with what I can do.

As a practical note about this particular milestone, I have included every single section present in the provided template just in case, but I have described what I hope to be valid rationale for why I did not go through with the guidelines for that particular section.

2 Validation

2.1 Values of Hard-Coded Constants in the Literature

The simplest variables to consider that I have set within my program are the choices of constants. The Particle Data Group (PDG) [1] contains listings for all of the values and their current known accuracy that I have used for my program. These can be found online at this link. In particular, the file `Constants.hpp` contains a number of these constants, the precision of which I have matched within my program:

- N_C, T_R, C_A, and C_F: These are group-theoretical constants, and are exact, i.e. not calculable or anything. They are not contained within PDG tables, but a Quantum Field Theory textbook usually contains the values for these variables; see, for instance, Chapter 16/17 of [2].
- α: The QED coupling constant, otherwise known as the fine-structure constant. This is calculable and known to a very high precision.

- m_Z, Γ_z: The mass and width (lifetime, essentially) of the Z-boson. This is often used as a reference scale for many calculations.
- m_C, m_B: The masses of the charm and bottom quarks, which are often higher than the lowest cutoff scale but lower than final evolution scales, so these are used as intermediate references in some cases.
- "Magic Factor": The magic factor is a simple unit conversion factor. In our calculations, we often set $\hbar = c = 1$ to simplify things, but of course, we must restore their values at the very end. This is done via this factor. As a unit conversion, there is no error associated with this value.
- G_F, θ_W: The Fermi coupling constant and Weinberg angle (often called the *weak-mixing angle*), are values used in calculations of the cross section.
- $\alpha_s(m_Z)$: The value of the strong coupling constant evaluated as the scale associated with the mass of the Z-boson. As mentioned, the Z-boson mass is very often used as a reference scale, and therefore so is the value of the strong coupling constant at that scale.

All of these values are *constants* in the very sense of the word: they must be hard-coded, otherwise, as I attempted to describe in the Section 1.1, if they were to be configurable, we would be simulating non-physical processes. Importantly, all of these quantities have been calculated to a very high precision, as one can see by examining the PDG tables. Therefore, it is reasonable to assume that the error incurred on the model output by the choice of these parameters is negligible.

2.2 Values of Configurable Input Parameters in the Literature

The configurable parameters require a bit more consideration. The easiest one, though is by far the center-of-mass energy. The default value is 14 TeV. There is nothing specifically in the literature for this parameter – it is simply just the current ECM that the LHC in CERN is colliding protons at. In fact, I believe that the value up until recently was actually 13.6 TeV. Either way, there is no particular limit on this parameter, it can be as high or low as one desires. Despite this, keeping values > 10 TeV is ideal since this is the range of current experimental data.

Some of the other parameters, namely the cutoff parameters, are a lot more complex. They can in principle be varied depending on the process, but the lower limit is the limit in which the perturbative nature of Quantum Chromodynamics (QCD) breaks down, i.e. the methods we used to solve the problem in the first place no longer become valid. This is universally understood to be $\sim \mathcal{O}(1\,\text{GeV})$ [3], and values around this parameter are picked as defaults in Pythia8 and Herwig7, for instance [4] [5]. Further, this is also chosen for computational efficiency, as quantities involving this scale often go like $1/E_{\text{min}}$, $\pm \ln E_{\text{min}}$ which diverge for $E_{\text{min}} \to 0$.

Therefore, the choice of the cutoff parameter, particularly in the case of the parton evolution, is a good choice and reflects choices often used in the literature. We also saw in the previous milestone that the variation of the output results was minimal for naturally small variations in this cutoff scale, reflecting not only that the choice of parameter value is good, but so is the performance of the model with choices in the neighborhood of the parameter.

The choice of the cutoff parameter in the case of the hard scattering process is a bit more flexible. In principle, it could be lowered to the same scale characteristic of the breakdown of perturbative QCD, but the parton showering algorithm saw directly the evolution of the parton down to these low scales, but in the hard scattering case, we are considering significantly higher energies, so we can afford to further increase the cutoff for additional computational efficiency (further avoiding divergences allow us to more easily use the hit-or-miss method, by allowing a simpler overestimate function), as we are decreasing the available phase space by a relatively smaller amount than in the parton showering case.

Shown in Table 1 are some definitive minimum and maximum ranges for the values of these cutoff parameters. To reiterate: the low end is more or less fixed, with the -0.5 low end on t_{min} being the absolute minimum before things break (Pythia8 actually will refuse to let you lower the cutoff below this value unless you explicity force it [4]), describing the energy at which the theorems on which this simulation are based break down. Upper ends are more flexible, and are dependent on what factors one wants to consider. The ranges in the table fit well for this use case, and represent what is found in the literature.

Variable	Name	Default	Low	High
$t_{\min}$	Parton evolution cutoff scale	1 GeV	-0.5	$+5.0$
$E_{\min}$	Minimum hard scattering phase space scale	60 GeV	-50.0	$+25.0$

Table 1: Acceptable ranges for the cutoff scales.

The other parameters within the configuration file, like those tested in the previous milestone, are intermediate calculational parameters that, as we saw, have no effect on the output of the simulation, so those will not be tested any further.

2.3 Monte Carlo Convergence

It is of interest to ensure that the Monte Carlo algorithm performs well, particularly in the residual statistical error incurred via the inherent limited number of iterations we can compute. In general, we can assess the error via standard statistical methods, such as computing the variance and standard deviation. The basic definition for the variance, denoted by σ^2, of a quantity X is

$$\sigma^2 = \langle X^2 \rangle - \langle X \rangle^2 \,, \tag{1}$$

or, in discrete terms:

$$\sigma^2 = \frac{1}{N}\sum_i X_i^2 + \left(\frac{1}{N}\sum_i X_i\right)^2 . \tag{2}$$

The standard deviation is simply the square root of the variance, hence it is denoted σ. By running the Monte Carlo simulation for enough iterations, we can reduce this standard deviation to within acceptable bounds. In particular, Table 2 contains, for a few values of the center-of-mass energy E_{ECM}, the calculated cross section and corresponding standard deviation measurements, using one million Monte Carlo evaluations.

E_{ECM}	Cross Section (picobarns)	Error
12.0 TeV	1541.2249 pb	±5.5898
13.0 TeV	1685.1843 pb	±6.0952
14.0 TeV	1829.2264 pb	±6.6145
15.0 TeV	1972.5091 pb	±7.1462

Table 2: Values of the cross section calculation along with standard deviations for different values of E_{ECM}, using one million Monte Carlo evaluations.

From the results, we can tell that the error is already very small for one million iterations. The code takes around 1 second to do these computations, so for comparisons sake, Table 3 contains the same data but instead taking ten million Monte Carlo iterations, which increase the runtime by a factor of 10 (as one would expect).

E_{ECM}	Cross Section (picobarns)	Error
12.0 TeV	1550.8109 pb	±1.7763
13.0 TeV	1687.4082 pb	±1.9312
14.0 TeV	1823.7320 pb	±2.0855
15.0 TeV	1963.3334 pb	±2.2437

Table 3: Values of the cross section calculation along with standard deviations for different values of E_{ECM}, using ten million Monte Carlo evaluations.

As one would expect, increasing the number of iterations reduces the standard deviation associated with the calculation. However, one million iterations still gives errors of around $\sim 0.3\,\%$, which is more than good enough. There is nothing to compare to the literature in this regard, as this is a simple statistical error test.

2.4 Comparison of Cross Section Calculation with MadGraph5

MadGraph5 is a tool that is used by and large for cross section calculations and hard scattering event generation. It provides a bit of a simpler interface for the direct calculation of the cross section, so for this comparison, we will use it. In Table 4 are listed values from $E_{\text{ECM}} = 12\,\text{TeV}$ to $15\,\text{TeV}$, comparing the results obtained from MadGraph5 and ColSim.

E_{ECM}	MadGraph5 σ_{xs}	MadGraph5 σ_{sd}	ColSim σ_{xs}	ColSim σ_{sd}	Relative Error
12 TeV	1550.8109 pb	± 1.7763	1553	± 1.244	0.14 %
13 TeV	1687.4082 pb	± 1.9312	1689	± 1.561	0.12 %
14 TeV	1823.7320 pb	± 2.0855	1832	± 1.435	0.49 %
15 TeV	1963.3334 pb	± 2.2437	1972	± 1.623	0.44 %

Table 4: Comparison of cross section calculations between MadGraph5 and ColSim for selected center of mass energies.

From the results, we can tell that this portion of the simulation holds up extremely well, with the relative error between my results and results from MadGraph5 being less than half a percent. Higher energies start yielding larger errors, but the error is still very manageable and reflects the performance of the model.

2.5 Comparison of Phase Space Kinematic Distributions with Pythia8

The other part of the program consists of the parton showering. Pythia8 is one of the more popular parton showering algorithms out there. However, there is one issue, and it is that the parton showering model implemented in ColSim is extremely simplistic, and even after cutting out as many additional features from the Pythia8 parton showering options, I am still unable to reduce it to a simple quark emitting gluons from some initial scale to a final scale.

Therefore, there is no reason to directly numerically compare the results from the distributions and determine any sort of relative errors. The only possible thing that can be done is a qualitative comparison of the shapes of the output distributions, and ensure that the distributions that I generate with ColSim match both the general structure from that of Pythia8 as well as the intuitive physics understanding.

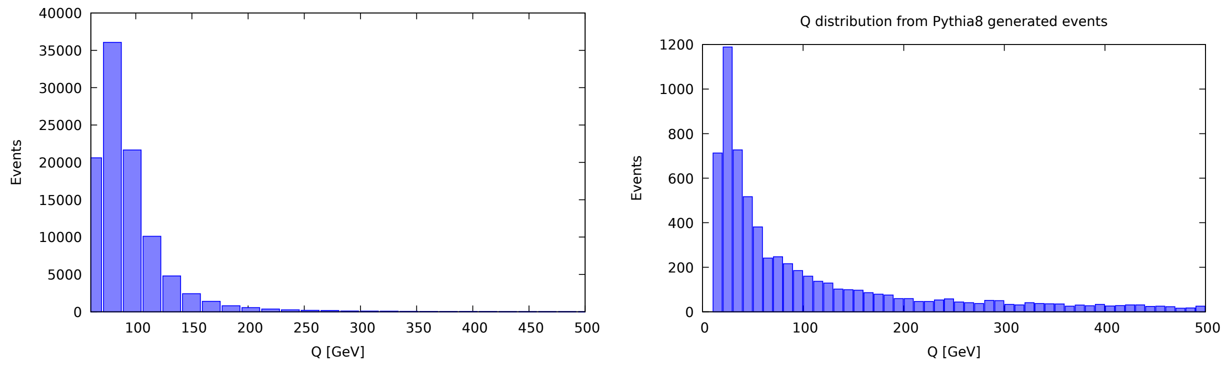


Figure 1: Comparison between parton showering invariant masses between ColSim(left) and Pythia8(right).

In Figure 1 I have given the plots for the distributions of the invariant mass for the generated events. There are some discrepancies, most notably, Pythia8 contains some additional events in the higher Q range. Figure 2 contains the rapidity distributions for the generated events (rapidity is a measure of speed, roughly speaking). In my code, this value is normalized a bit differently, but we notice that it follows the same pattern, i.e. higher (absolute value of) rapidity events are less likely. Lastly, in Figure 3 are the momentum fractions the quark in the initial reaction contains. These are remarkably equivalent; evidently, the quarks involved in the showering process represent those with a small momentum fraction in the proton.

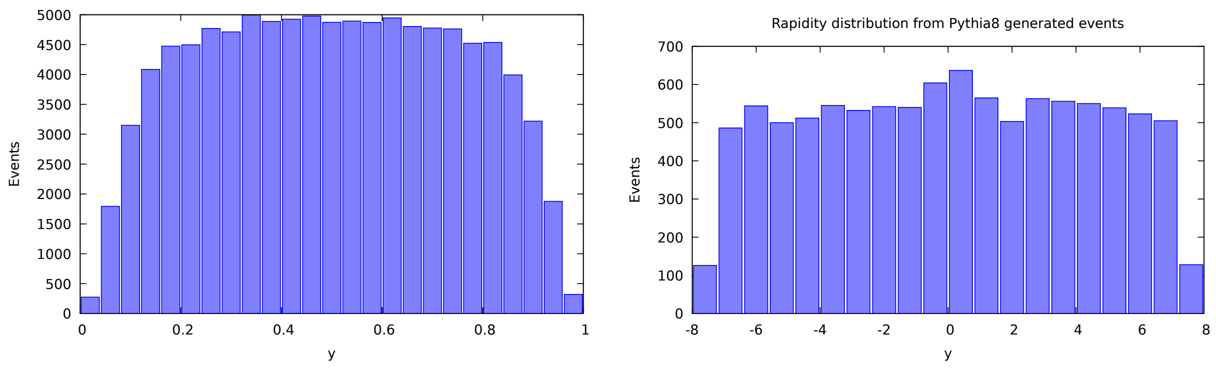


Figure 2: Comparison between rapidity between ColSim(left) and Pythia8(right).

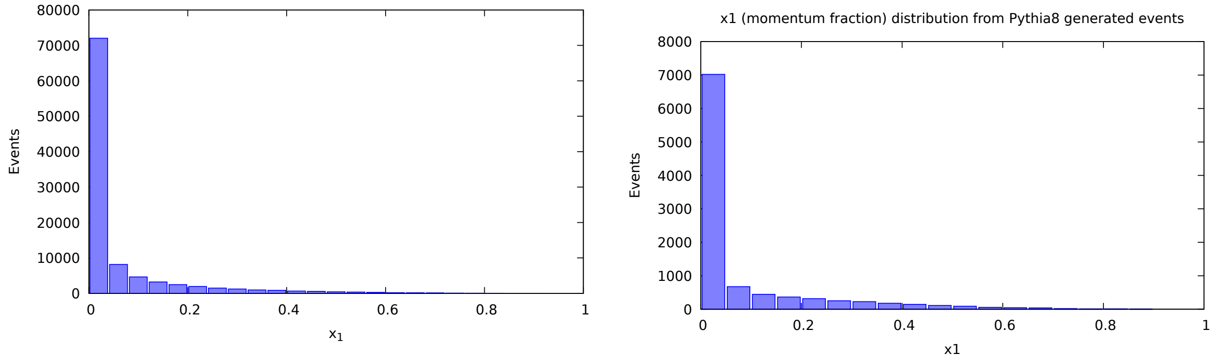


Figure 3: Comparison between the momentum fraction between ColSim(left) and Pythia8(right).

2.6 Face Validation

My understanding of this section is that half of it is meant to be comparing the model outputs with what is expected by field experts. Of course, if my model matches the behavior both quantitatively in the case of the cross section calculation and quantitatively in terms of the parton showering, and the comparisons are drawn between programs written by field experts, then I have more or less already satisfied the requirements for this section.

Further, a few implementation specifics have been done with assistance from a particle theorist here at KSU who specializes in Monte Carlo algorithms and event generators. I have nothing to cite in this regard, but considering again that the behavior of the model matches so well with current literature and existing models I think that I have satisfied this section.

2.7 Historical Validation

In the case of my model, there is not much of a reason to compare with historical data. In principle, the results themselves have not drastically changed, but rather our computational tools and theories have grown to more or less reduce the error/uncertainty with the actual calculation of values. For instance, the value of all the input parameters have more or less remained the same since their conception, but have simply had their uncertainty reduced. All I'd be doing, then, is comparing against results that effectively represent the incomplete theory and incomplete computational results, which is not instructive.

2.8 Follow-up from Milestone 6

In general, the performance of the model under the variation of the possible input parameters was great. The only item that was slightly suspicious at the time was the fact that a moderately significant increase of

```
#include <ColSim/ColSim.hpp>
using namespace ColSim;

#include <string>
using namespace std;

int main() {

  LOGGER.logMessage("This is a message.");
  LOGGER.logWarning("This is a warning.");
  LOGGER.logError("This is an error that will NOT abort the program.");

  string str = "this is a string!";
  LOGGER.logMessage("String: %s", str.c_str());

  int x = 42;
  LOGGER.logMessage("Integer: %d", x);

  double y = 5.679122;
  LOGGER.logMessage("Double: %.5lf", y);

  LOGGER.logAbort("This is an error that will abort the program.");

  return 0;
}
```

Listing 1: Test program to testing logging functionality.

the center of mass energy led to only a small shift in the curve of the transverse momentum of the generated events for the hard scattering process. After consideration, this is not very surprising: the center-of-mass energy is related to the *longitudinal* motion of the protons along the beam axis and hence the quarks. However, we were measuring the *transverse* momentum, which is perpendicular to the beam axis. Therefore, an increase in the center-of-mass energy of the quarks doesn't lead to a *direct* increase in the transverse momentum since the two planes are perpendicular, but should increase it a *little* simply due to an excess of energy in the collision; it should be more likely to create particles of higher transverse momentum, but the small shift is much more reasonable and not something to be worried about.

Apart from that output, there is nothing else to discuss regarding the performance of the model from the previous milestone.

3 Verification

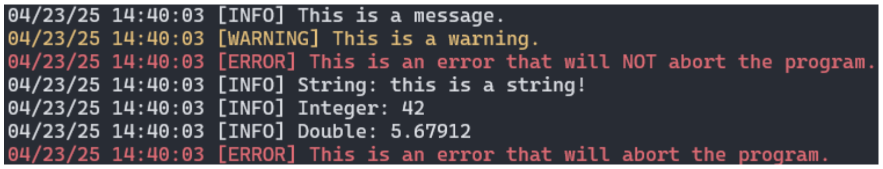
3.1 Core Components Unit Tests

Here I give some small unit tests for some of the smaller core components of the program, things like the Logger and some of the physics objects like four-vectors.

3.1.1 Logger

The unit tests here are minimal; all we need to test is that the four types of errors print as they should and handle the printing of various data types.

The code used to test the logger is given in Listing 1 and the output is given in Figure 4. Evidently, the logger is performing as expected.



```
04/23/25 14:40:03 [INFO] This is a message.
04/23/25 14:40:03 [WARNING] This is a warning.
04/23/25 14:40:03 [ERROR] This is an error that will NOT abort the program.
04/23/25 14:40:03 [INFO] String: this is a string!
04/23/25 14:40:03 [INFO] Integer: 42
04/23/25 14:40:03 [INFO] Double: 5.67912
04/23/25 14:40:03 [ERROR] This is an error that will abort the program.
```

Figure 4: Output from the code given in Listing 1

3.1.2 Settings

The Settings functionality will also be similar to test. All input variables have a default value, so the behavior for them all is that if they are neglected from the config file, no warnings/errors occur, rather the default value is set for that parameter. This can be illustrated by a simple test case as shown in Table 5. A few variables, such as the aforementioned cutoff scales, have hard limits and cannot be configured below 1.0, otherwise the program will crash. It also will emit warnings if the the cutoff is set too high, as this often reduces the accuracy of the calculation. These cases are given in Table 6.

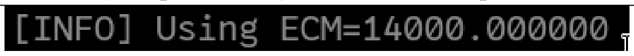
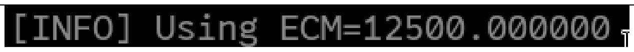
Action	Keeping/removing E_{ECM} from the config file.
Expected Output	If omitted, keeps 14 GeV, otherwise uses specified value.
Actual Output (Omitted)	 `[INFO] Using ECM=14000.000000`
Actual Output (Specifying 12.5)	 `[INFO] Using ECM=12500.000000`

Table 5: Test case for omitting E_{ECM} from the configuration file.

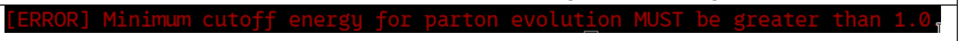
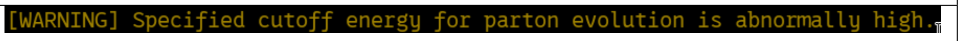
Action	Specifying a value < 1.0 and > 50.0 for the parton evolution cutoff scale.
Expected Output	For < 1.0, aborts, for > 50.0, gives warning.
Actual Output (< 1.0)	 `[ERROR] Minimum cutoff energy for parton evolution MUST be greater than 1.0.`
Actual Output (> 50.0)	 `[WARNING] Specified cutoff energy for parton evolution is abnormally high.`

Table 6: Test case for violating ranges for parton evolution cutoff energy.

Evidently, the settings interface seems to be performing as expected, and aborts upon specifying parameters that are completely incompatible with the model (and underlying theory), and warns if parameters are specified that are in principle allowed, but may lead to less accurate results.

3.1.3 Four-Vectors

The testing for four-vectors is relatively simple: the only non-trivial functionality of four-vectors is to ensure that the calculation of the transverse momentum components, i.e. $p_T^2 = p_x^2 + p_y^2$, is performed correctly, the calculation of the norm $p_\mu p^\mu \equiv p_0^2 - p_x^2 - p_y^2 - p_z^2$ is performed correctly, and boosts along the z-axis (the beam axis). A boost is defined by a parameter β which is the fraction of the speed of light to boost the particle. Defining

$$\gamma = \sqrt{\frac{1}{1-\beta^2}}, \tag{3}$$

the boost is given by

$$p^\mu \to p^{\mu\prime} = \begin{pmatrix} \gamma(p_0 - \beta p_z) & p_x & p_y & \gamma(p_z - \beta p_0) \end{pmatrix}. \tag{4}$$

We choose some random values for the components:

$$p^\mu = \begin{pmatrix} 89.0 & 5.9 & 3.4 & 9.8 \end{pmatrix}. \tag{5}$$

Table 7 contains the result from this test. Clearly, we are computing the correct values for four-vectors.

Quantity	Expected	Actual
p_T^2	7778.59	7778.59
$p_\mu p^\mu$	46.37	46.37
Boost of $\beta = 0.5$	$p^\mu = (42.05,\ 5.9,\ 3.4,\ 412.85)$	$p^\mu = (42.05,\ 5.9,\ 3.4,\ 412.85)$

Table 7: Calculated and expected values for calculated quantities involving four-vectors.

3.1.4 Gnuplot

The interface for plotting things with Gnuplot is less of a concrete test but rather that it can handle plotting some generic function. Due to the histogram format of the plotting that is done for the rest of the program, I do a mini Monte-Carlo to essentially plot $sin^2(x)$ between $0 - \pi$. The listing is a little long, so it given as Listing 2 in Appendix 8.1.

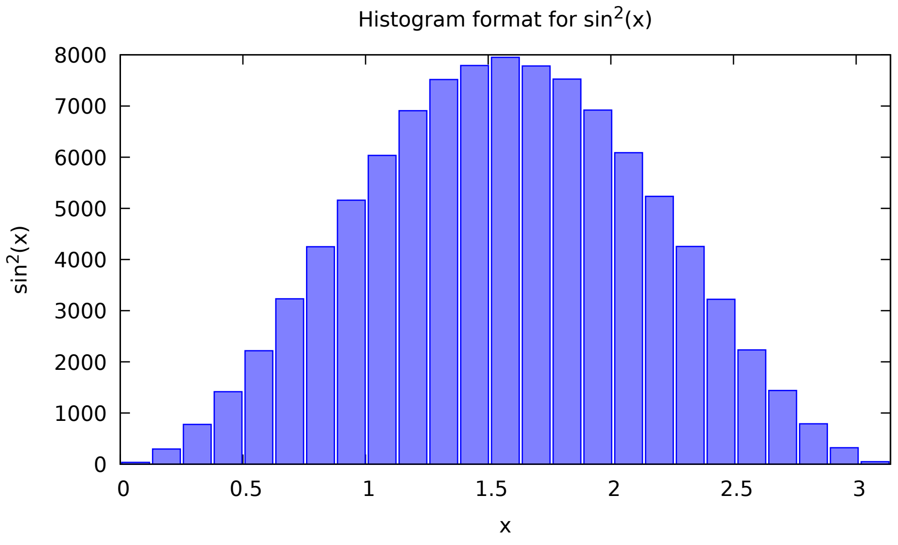


Figure 5: Histogram form for the plot of $sin^2(x)$.

The plot for the resulting distribution is given in Figure 5. The code for the Gnuplot interface therefore works as expected, and is relatively easy to use.

3.2 Monte Carlo Unit Test

A simple unit test of the Monte Carlo algorithm is conducted by running the algorithm using a few simple functions that can be calculated analytically. Since there is no reason for the algorithm to work for simple functions but fail for complex functions, I have selected a few easy functions that are easy to integrate analytically (or otherwise have exact known results):

Integral	Monte Carlo Result	Error	Exact Value	Relative Error
$\int_0^1 \mathrm{d}x\ 2x$	1.0002	$\pm 5.774 \times 10^{-4}$	1.0	$0.022\,\%$
$4\int_0^1 \frac{\mathrm{d}x}{1+x^2}$	3.14164	$\pm 6.435 \times 10^{-4}$	π	$1.42 \times 10^{-3}\,\%$
$\int_{-\infty}^{\infty} \mathrm{d}x\mathrm{d}y\mathrm{d}z e^{-(x^2+y^2+z^2)}$	5.8080	± 0.122	$\pi^{3/2}$	$6.59\,\%$

Table 8: Test functions to ensure Monte Carlo convergence.

The results are shown in Table 8. The first two functions are simple analytic functions and they yield less than a tenth of a percent relative error between the exact values and those determined from the Monte Carlo algorithm. The third function is a Gaussian, and due to the nature of the limits being $\pm\infty$, it is much harder to accurately assess the integral with smaller numbers of iterations. There are techniques to mitigate this, but the integrals that I compute do not have this feature, so I do not think it is necessary to fix anything. It is interesting to see, though, where the Monte Carlo algorithm breaks down. Despite this, like I just mentioned, the integrals I compute are finite, meaning the algorithm's convergence is very good and it works as intended.

These are the main sub-components within the project. There are a few more super minor utility functions and other things, but these are insignificant enough to not necessarily require unit testing.

3.3 Physics Components

The issue with trying to unit test the physics components is that, at the end of the day, the main part of the hard scattering process revolves directly around the Monte Carlo algorithm. We have shown that that works very well. The remaining physics component is, in the context of this milestone, implementing the equations that are calculated with the Monte Carlo algorithm. Therefore, there really is no remaining unit testing to be done with respect to the hard scattering.

With respect to the parton showering, the Sudakov form factor is the main driving component. This, however, is less of a general algorithm like Monte Carlo, but instead is implemented directly for this case. Therefore, the only testing that can be done is on the entire unit as a whole, which has already been done in the previous main section.

Therefore, this concludes the unit testing for the main sections of the progam. It is evident that the individual components work well on their own and can withstand different inputs being presented.

3.4 Regression Testing

I am unsure what to write for this section. The behavior of the above individual components have worked from the get-go, with only interface changes; i.e. the Monte Carlo algorithm has always worked, the Gnuplot interface has always plotted the functions, with, again, only minor API changes. Therefore, there is nothing to really document for this testing case, but rather, the documentation changes will be reflected in the main program documentation.

3.5 Edge Case Testing

This has already been conducted in the previous section and the previous milestone, specifically in the case of the Settings testing and varying of the input parameters. Specific limits have already been tested and described thoroughly in the previous section, which are the edge cases this section refers to. Therefore, refer to that section and Milestone 6 for the edge case behavior.

3.6 Integration Testing

Integration testing is challenging, because any integrations of the above individual components are already present in the main output that I have already extensively discussed. Therefore, considering that the model performance overall considering both the numerical results and also the simple fact that the output is readable, nothing crashes, etc., I think the integration testing is more or less satisfied.

3.7 Code Review

A big (and apparently controversial) opinion of mine is that code readability is entirely subjective. If one is working on a large project with a number of collaborators, it's must consider the practices of the collaborators, but this project is one I am programming myself. Therefore, readability is entirely my own preference, and has been written from the get-go with my own best practices I have been refining throughout my career.

In terms of actual coding practices as they relate to C++ programming, I have enabled all possible warnings and extra warnings during the compilation, i.e. via the `-Wall -Wextra` flags to the `g++` compiler.

These have ensured that improper (not *unreadable*) code is avoided as much as possible. The current state of the code compiles in its entirety with zero warning (and of course zero errors).

3.8 Version Control

The code can be found on GitHub here. The version controlling has been very poor recently, as the majority of the work has been spent running tests and whatnot. Further, my version controlling via Git has not been the best in general, which is something that I have to improve on.

4 Validation and Verification Checklist

4.1 Verification Checklist

✓ Have you compared simulation outputs with real-world data?

✓ Are all model parameters within reasonalbe and justified ranges?

? Does the model's behavior align with expert expectations?

- I didn't explicitly bring results to an expert. However, as I mentioned earlier, the program was written with help from an expert, so I feel this is more or less satisfied.

✓ Have you documented all validation reuslts using appropriate visuals?

? Have you performed historical data validation?

- As described earlier, historical data should match, since it describing the same physics, but just with more error, meaning I feel it not very instructive to compare to historical data.

✓ Did you conduct cross-model validation where applicable?

4.2 Verification Checks

✓ Have you tested the model with boundary and edge conditions?

✓ Did you verify that all input parameters are properly handed?

✓ Have you checked for mathematical and logical consistency in your code?

✓ Is the model's behavior reasonable across different time scales or scenarios?

✓ Have you systematically documented all testing results?

? Did you perform unit, integration, and regression testing.

- As mentioned earlier, my interpretation of the integration testing is the incorporation of the different elements into the main code, which is simply what the main program does, the output of which has been documented so far. Further, regression testing wouldn't be particularly beneficial since, just like with the historical data, previous results should (and do) describe the same physics with the same result, modulo some minor API changes and error changes.

× Are you following version control best practices?

- This is something that I need to improve on. At this point in the semester it's too late to fix this, since so little additional code is going to be written.

4.3 Documentation Checks

I feel these checkpoints are more or less what have already been checked, but alas:

- ✓ Is all documentation up to date and reflective of the current model?
- ✓ Have you documented all assumptions and limitations?
- ✓ Are test cases and results properly recorded?

4.4 Version Control Checks

- × Are all changes commited with descriptive messages?
  - Changes are *committed*, but not necessarily with descriptive messages.
- × Have you tagged versions appropriately?
- ✓ Is the repository synchronized with the remote server?

4.5 Data Validation Completeness Checks

- ✓ Is the data used for validation complete and reliable?
- ✓ Have you checked for data inconsistencies or anomalies?

5 Common Pitfalls

5.1 Validation Mistakes

- **Ignoring Data Discrepancies**: The one discrepancy between expectations that arose from the previous milestone was solved after further consideration of the physics/theory behind the process. Other discrepancies are impossible to quantify, specifically in the parton showering case, because it is impossible to ensure that Pythia8 is operating under identical conditions. The best we were able to do is quantitatively compare the output distributions, and we found little discrepancies.
- **Inadequate Data**: There was no out-of-data or inadequate data being compared against.
- **Overlooking Parameter Relevance**: All input parameters were considered thoroughly.
- **Over-Reliance on a Single Validation Model**: In principle, this is happening. However, when comparing to existing data or existing runs within Pythia8, there are different models for calculations of the same process, but are computed to extremely low error due to expanding to higher perturbative orders, which I cannot consider. Therefore, the comparison between these different models is not relevent for my case because my error is far higher than that done in Pythia8. I reckon, hence, that this isn't an issue.
- **Failure to Cross-Validate**: I cross-validated well for my model.
- **Misinterpretation of Validation Metrics**: I don't believe any metrics are misinterpreted.

5.2 Verification Oversights

- **Not Documenting Failed Tests**: No tests that I conducted failed.
- **Insufficient Test Coverage**: As mentioned there were some parts such as various utility functionality that were not thoroughly tested, but due to being such minor parts of the code, I felt it unncessary to extensively document. The main running parts of the code were tested/documented.

- **Lack of Systematic Approach**: I feel that the test cases were approached systematically.
- **Skipping Edge Cases**: I covered all edge cases.
- **Overlooking Regression Testing**: All the way back to several milestones ago I quotes identical values for the calculation of the cross section and basic distribution shapes. Nothing has changed, i.e. nothing has broken. This wasn't *extensively* tested, but it is clear nothing has broken from code changes.

5.3 Avoid Overfitting in Validation

A separation between different datasets for different purposes is not necessary in this case. With enough iterations for the Monte Carlo algorithm enough events generated to study the outputted kinematic distributions, we are merely converging to the actual values that would appear in the real world.

6 Documentation

6.1 Assumptions/Limitations

There are few assumptions made on the actual model itself, but rather assumptions made in the underlying theory governing the cross section calculations and the fact that we are able to model parton evolution via the parton showering algorithms. I don't want to go into detail about the specific physics, but I will describe one thing, which I described in one milestone a while back.

Physics, in the quantum scale, cannot be solved exactly in any way. Our best understanding of quantities like the cross section (which governs the rest of the process) is in terms of an infinite series of terms that, in the limit to infinity, converge to the answer. However, we cannot compute an infinite number of terms. In fact, most modern calculations only consider three; mine considers one. The main assumption behind the model is that the "leading-order" result, i.e. keeping only one term, is already a very accurate solution to the problem. Considering that our results for the cross section and qualitative picture of the parton showering match so well already to programs which consider higher-order terms in the series is indicative of the fact that despite the assumption of only keeping leading-order terms, we are still capturing enough of the fundamental physics of the process itself.

Therefore, in terms of limitations, the model does not consider all possible "intermediate" processes that would in principle physically occur in a proton-proton collision. Even in terms of the specific interaction I am considering, $pp \to Z/\gamma^* \to \ell^+\ell^-$, the intermediate Z/γ^* particles are representative of the leading order result. If I really wanted to consider more terms, there would be further interactions between the pp and $\ell^+\ell^-$. Again, though, the leading-order result is already very strong at capturing the physics of the problem.

However, as I just mentioned, there are more processes in the main pp collision that are not covered here. Quantum chromodynamical (QCD) interactions are not even considered, and they are a major part of our current understanding of high-energy interactions. However, the theory governing QCD is extremely complex, and something that is highly non-trivial to get implemented and working correctly. Because of this, it is safe to say that my model does not fully model precisely what happens in real pp collisions; but, no program *can* by virtue of how we construct our theory in the first place. So, sure, my model "failed" at modeling true pp collisions, but by capturing a leading order result in this way, I have already covered a large portion of the infinite series of terms.

Again, in terms of the model itself and the implementation, there were few limitations or assumptions made. One thing that could affect the model would be the number of events generated and number of Monte Carlo iterations to determine results. Of course, a large but finite number is chosen as a default and is used for a majority of the results presented so far, meaning there is *some* statistical error, but not enough to significantly affect the model output, specifically in the physical interpretation of the results.

The only other limitation, when it came to the V&V process, was that comparison to other models for the parton showering process was impossible due to the relatively simplistic nature of my implementation. The entire process is extremely hard and incorporates a lot of physics, and it simply wasn't possible to strip other programs down to this level. We were therefore limited to simple qualitative comparisons, but these still captures the physics of the process and we found qualitative agreement, as best we could.

6.2 Documentation Standardization

The rest of this section appears to me to be guidelines for the previous parts of the milestone that have already been covered, so I reckon there is nothing else to say.

7 Discussion and Conclusions

To conclude, in this milestone we have extensively covered the verification of our model against that from other models such as MadGraph5 and Pythia8. We found fantastic comparison with the former, but due to the impossibility of constructing identical situations within Pythia8 for the parton showering case, we were limited to only qualitative comparisons. However, despite this, these comparisons still yielded great results and indicate that the model is indeed capturing the physics it set out to capture.

Further, we also validated the model itself and ensured that it could survive against spurious input parameters, as well as ensuring that each major component of the model works as intended. With all of this, I believe that the model is ready for the final deliverable.

8 Appendix

8.1 Listing for Gnuplot Test

```
// required for setup of random number generator
ColSimMain _;

// create gnuplot object
Gnuplot plot;

// specify plot information
vector<double> min{0.0}, max{3.14}, delta{3.14};
double numBins = 25;
plot.setHistInfo(min, max, delta, numBins);

// give col/file name, open the datafile
vector<string> colNames{"sin2_x"};
plot.openDataFile("data.dat", colNames);

// grab values of sin(x)
// in general we take a vector of vectors
// for multiple variables,
// that's the reason for adding a vector
uint numPoints = 0;
while (numPoints < 100000) {
        double x = randDouble() * 3.14;
        double r = randDouble();
        double val = sin(x)*sin(x);
        double ratio =  val / 1.0;
        if (r < ratio) {
            plot.addDataPoint(vector<double>{x});
                numPoints++;
        }
}

// define the x and y labels
vector<string> xlabel{"x"}, ylabel{"sin^2(x)"}, title{"Histogram format for sin^2(x)"};
plot.setTitles(title);
plot.setXLabels(xlabel);
plot.setYLabels(ylabel);

// make the plots
plot.plot();
```

Listing 2: Listing for the plotting of $\sin^2(x)$. Given without a main function or includes.

References

[1] S. Navas et al., "Review of particle physics," *Phys. Rev. D*, vol. 110, no. 3, p. 030 001, 2024. DOI: 10.1103/PhysRevD.110.030001.

[2] M. Peskin and D. Schroeder, *An Introduction to Quantum Field Theory*. CRC Press, 2019.

[3] A. H. Hoang, O. L. Jin, S. Plätzer, and D. Samitz, *Matching hadronization and perturbative evolution: The cluster model in light of infrared shower cutoff dependence*, 2024. arXiv: 2404.09856 [hep-ph]. [Online]. Available: https://arxiv.org/abs/2404.09856.

[4] C. Bierlich et al., *A comprehensive guide to the physics and usage of pythia 8.3*, 2022. arXiv: 2203.11601 [hep-ph]. [Online]. Available: https://arxiv.org/abs/2203.11601.

[5] M. Bähr et al., "Herwig++ physics and manual," *The European Physical Journal C*, vol. 58, no. 4, pp. 639–707, Nov. 2008, ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-008-0798-9. [Online]. Available: http://dx.doi.org/10.1140/epjc/s10052-008-0798-9.