Hypatia Systems: A New Way of Learning Mathematics Online

Matthew Sahota
University of British Columbia
MATH 498

Table of Contents

Letter of Transmittal

To whom it may concern,

This report was prepared for Ladislav Stacho, a Professor of Mathematics at Simon Fraser University. It covers the work that I had done at Hypatia Systems during my co-operative work experience term in the summer of 2021.

This report is to be used as reference for any future employees or students that may use or modify the work that I had completed during my term at Hypatia Systems. The content in this report is non-confidential.

The focus of this report is on the application of LaTeX and current technology to design math curricula from Grade 8 to university level that is interactive and offers students feedback about how they can improve their understanding of mathematics. The report details the process of designing the questions and troubleshooting some of the issues encountered while setting up the questions. This report also details the usage of Python to set up the plots of Cartesian, conic, and parametric equations.

At the end of this report are suggestions for any future of students and employees that wish to work with Hypatia Systems.

Sincerely,

Matthew Sahota

UBC Science Co-op

Hypatia Systems: A New Way of Learning Mathematics Online

Matthew Sahota
University of British Columbia
MATH 498

Introduction

Mathematics is a very important topic in many aspects of life, such as cellular communications, economics, and aeronautical travel. There are also many online resources to help students with learning mathematics, however, most of them suffer from a limited scope of mathematical maturity and a lack of interactivity and ability for students to show and/or reflect on their work. This dearth in comprehensivity leads to students not fully understanding the curriculum at hand, and therefore leaves students with an incomplete learning experience.

The goal of Hypatia Systems is to develop a learning system for mathematics that is both interactive and able to cover a wide breadth of mathematical topics so students can have a complete learning experience. Hypatia Systems accomplishes this by using a system called CheckMath that checks the work that students input into the question and make sure that it makes mathematical sense, while also providing feedback if students do not get the correct answer. The usage of CheckMath allows for the creation of multi-step problems that would otherwise have to be done in one step on other learning systems. This is especially helpful for higher level problems, such as those encountered in first- or second-year calculus courses, as well as problems in linear algebra and analytical geometry that may not be doable for most learning systems.

However, CheckMath does not work with all types of problems, such as epsilon-delta proofs, polar equations, and implicit differentiation, and therefore, requires constant updating to be able to provide feedback for students. Different strategies must be used to avoid technical difficulties without sacrificing the depth of the curriculum and the analytical thinking needed for the problems. Communication between the development team and the worksheet design team, as well as within the worksheet design team, is also important to make sure that CheckMath is updated to deal with any new problems and technical difficulties that may arise from designing questions in more advanced curricula.

Discussion Part I: Incorporating CheckMath

CheckMath is a system that checks one's work in LaTeX while writing mathematical equations, and it can be used to check if the work that a student writes makes sense. CheckMath can also use answer boxes that allow for a student to write parts of a question independently or to check if the answer satisfies a certain criteria while providing specific feedback if a student does not answer a question correctly. CheckMath can be used for all sorts of mathematical questions, such as simple ones like true or false and multiple choice questions, and it can also be used for complex questions such as those involving probability, differential and integral calculus, and polar and conic equations.

Most of the work that I had done with CheckMath was setting up questions for topics in Grade 11 (Algebra and Trigonometry), Grade 12 (Precalculus) and differential calculus. I was responsible for designing questions by writing the solution first and providing any relevant figures if necessary, then designing the work area recognized by

CheckMath around the solution that I had given, as well as checking for any faults in the CheckMath system by either inserting incorrect answers to check for false positives or noting if any of the questions are not recognizable by CheckMath. Another thing I was tasked with was checking the questions to see if the answer makes sense, and if CheckMath was set up correctly for the question.

Whenever there were inconsistencies with CheckMath, I communicated with a development team through Slack to point those inconsistencies out. Some of the difficulties that I initially had, but eventually resolved, involved questions with complex numbers and implicit differentiation. CheckMath was also prone to timing out when writing more complex equations or pointing out syntactical errors when there weren't any, such as when CheckMath would expect a matrix when the question never called for one. The development team also added features to help make writing the questions easier, such as adding sections where you could solve for a certain variable or set a function as the question requires. There was also a toolbox at the bottom of the screen where question creators could add more complex symbols like limit and summation notations or Greek characters for students to use.

As stated before, CheckMath did not work with epsilon-delta problems for limits since there was no support for equations with inequalities. However, I found a way to work around the incompatibility of CheckMath with the epsilon-delta problems. What I did was put my question into title sections for text and added answer boxes whenever there was anything that needed to be filled in. Below is an example of one such question that used this new format.

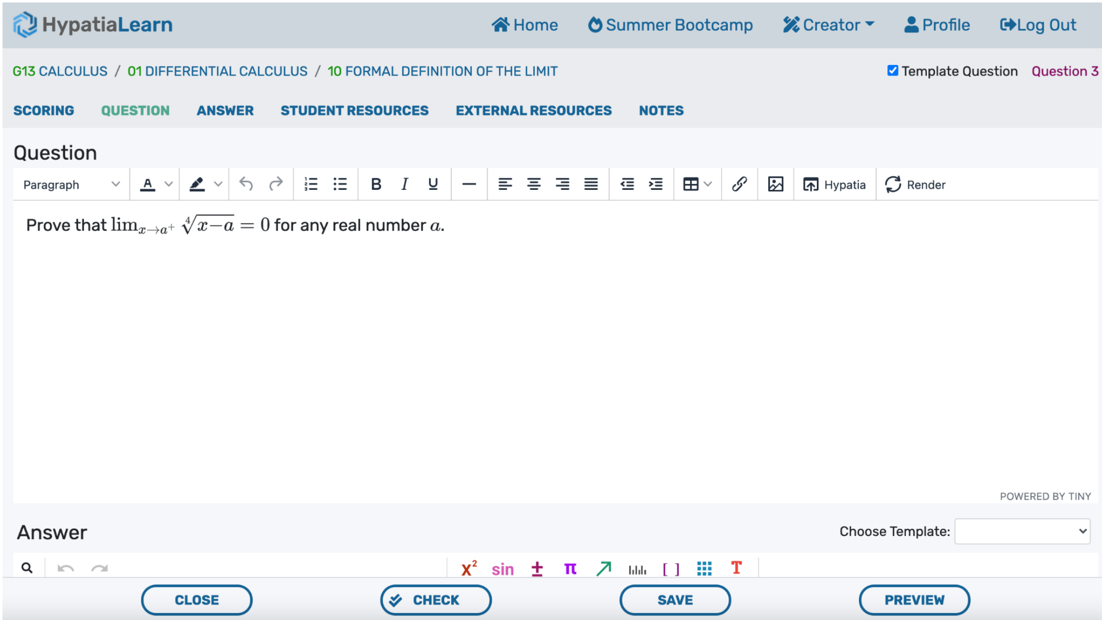

Figure 1: The question without the workplace.

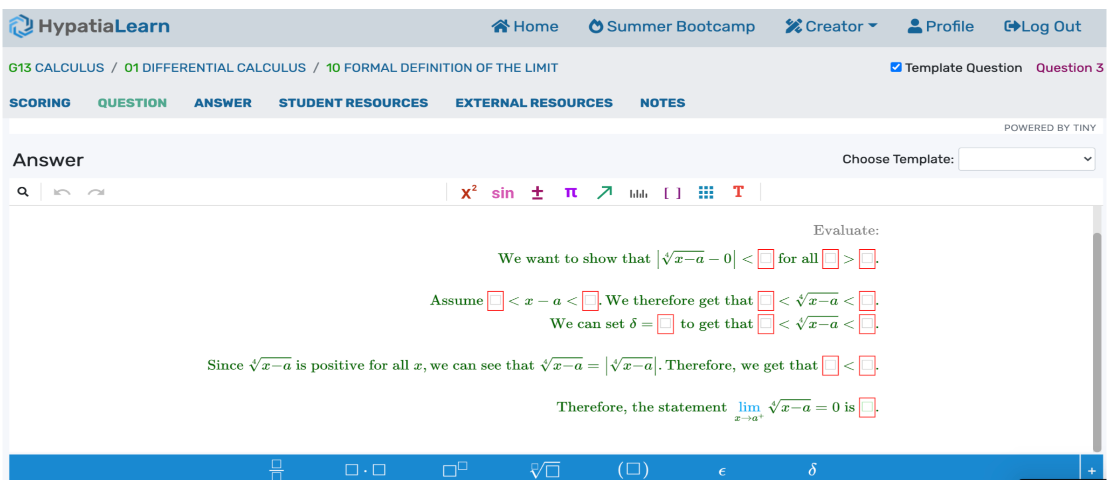

Figure 2: The workplace for the question. The red boxes are answer boxes to be filled.

The epsilon and delta characters are added below for students to use.

The green text denotes a comment section. The benefit of using this layout is that all the answers in the boxes can be recognized by CheckMath without any incompatibilities, but it does not allow you to show your work in multiple steps; you can only insert the answers in the boxes in one step. Notice that even if there are mathematical expressions in the comments, they will not be recognized, nor can the expressions or any of the green text be modified by the student. We will now attempt the question to show how CheckMath works; the first figure will have the question when one sees it the for the first time, the second figure will have one of the boxes with the incorrect answer, the third figure will have an empty box, and the final figure will have the correct answer.

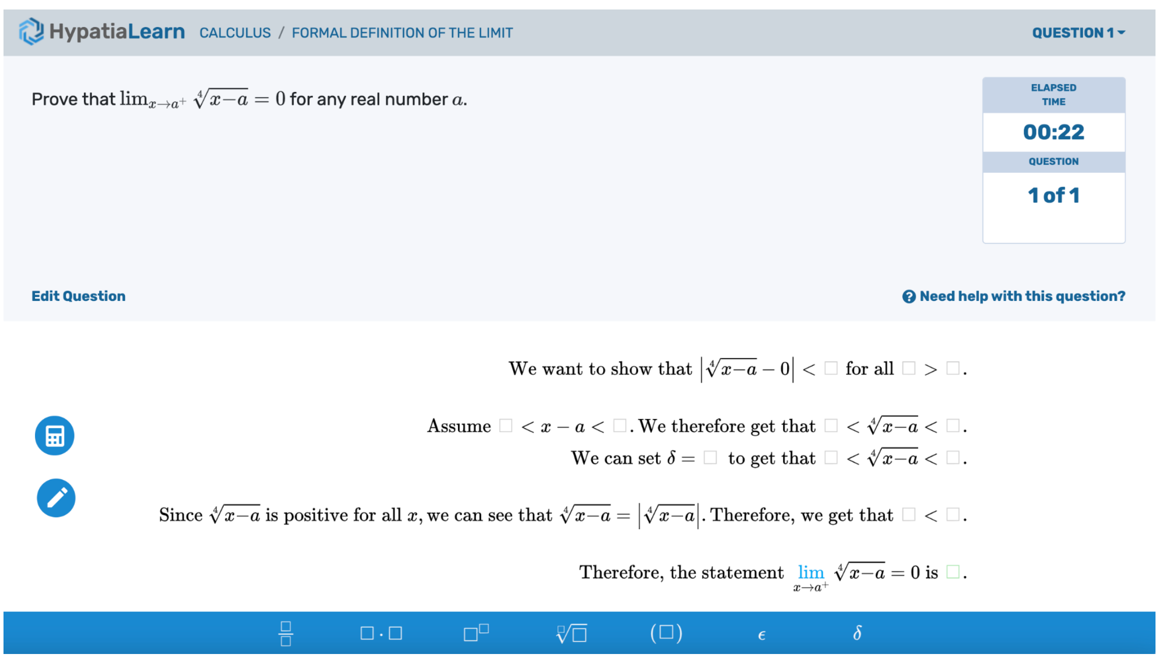

Figure 3: The question with none of the answer boxes filled.

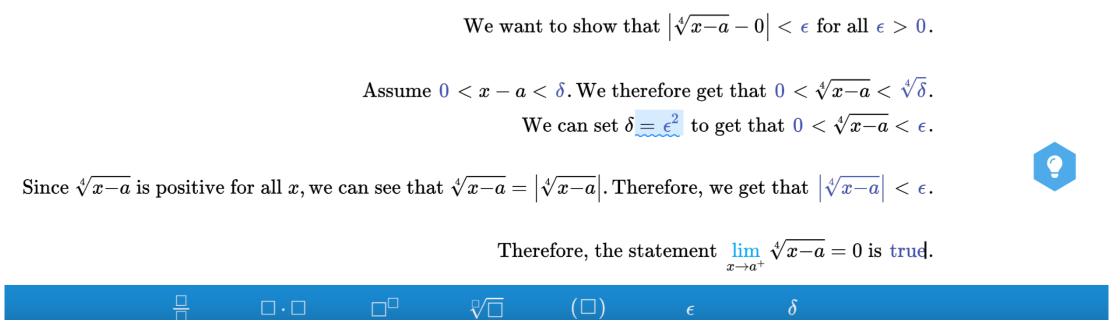

Figure 4: One of the answers is incorrect, as highlighted in blue.

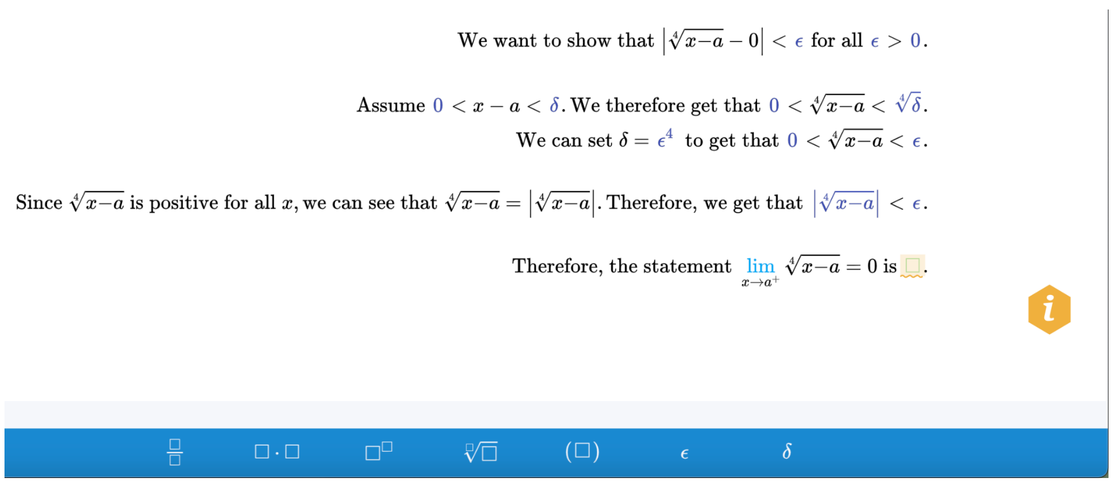

Figure 5: One of the answers is missing, as highlighted in yellow.

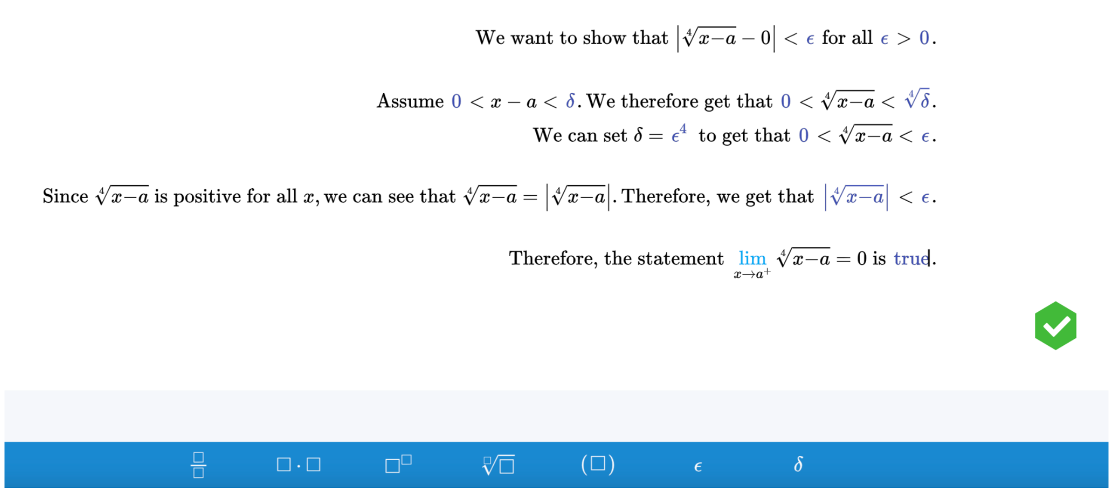

Figure 6: The correct answer for the question.

CheckMath works by checking the answer boxes and seeing if they are all correct; if one or more of the boxes is either empty or incorrect, CheckMath will prompt the user to go back and check the answer again. If all the boxes are correct, then the green check mark will appear at the bottom right of the screen. For multiple step problems that require multiple lines to show work, CheckMath will check each line for any incongruencies in the work. If there are any mistakes, CheckMath will flag the user to go review the work to correct the mistakes. Custom hints or feedback can also be added if a student answers a part of a question incorrectly, such as if points need to be entered from left to right or if a text answer requires a certain set of inputs, such as “true” or “false”, or “equals” or “does not equal”.

Discussion Part II: Designing Figures with Python

Alongside CheckMath, I also used Python to graph equations to either set up questions or to provide a figure for students to see so that they can better understand how a concept is used, and I used a writing tool to design any diagrams relevant to a question. For the most part, I worked with another coworker to update the Python plots and hand drawn images, but my coworker and I found a way to avoid needing to touch up the Python plots by adding code to make the plots transparent and the plot titles renderable by LaTeX. I had almost exclusively used Python to plot a lot of my equations, such as rotated and/or translated conic equations, piecewise equations, and vectors, however, I had particular difficulty plotting points in polar coordinates, for which I used another graphing software.

To design some of the non-rotated conic equations, such as ellipses, parabolas, and hyperbolas, I used trigonometric parametrizations to represent the conic equation. However, one issue that I had involving graphing hyperbolas was plotting both parts of the hyperbola. The Python code only allowed me to plot one part of the hyperbola, so I made two equations using different parameter values to represent both parts. Below is the Python code I used to represent a hyperbola with the equation $x^2 - 2x - y^2 - 2y = 1$ accompanied with an image of the equation.

```python
import numpy as np
import matplotlib.pyplot as plt
plt.rc('text',usetex=True)
plt.rc('font',family='serif')

# Conics
t0 = np.linspace(-2500,2500,200)
t1 = np.linspace(-1.57,1.57,2000)
t2 = np.linspace(1.58,4.71,2000)
x0 = np.zeros(len(t0))
y0 = 0*t0
x1 = 1/np.cos(t1) + 1
y1 = np.tan(t1) - 1
x2 = 1/np.cos(t2) + 1
y2 = np.tan(t2) - 1

fig = plt.figure()
plt.plot(t0,y0,color='k')
plt.plot(x0,t0,color='k')
plt.plot(x1,y1,color='b')
plt.plot(x2,y2,color='b')
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.grid()
plt.show()

fig.savefig('hyperbola.png', dpi = 80, transparent=True, format = 'png')
```

Figure 7: Code used to plot hyperbola $x^2 - 2x - y^2 - 2y = 1$.

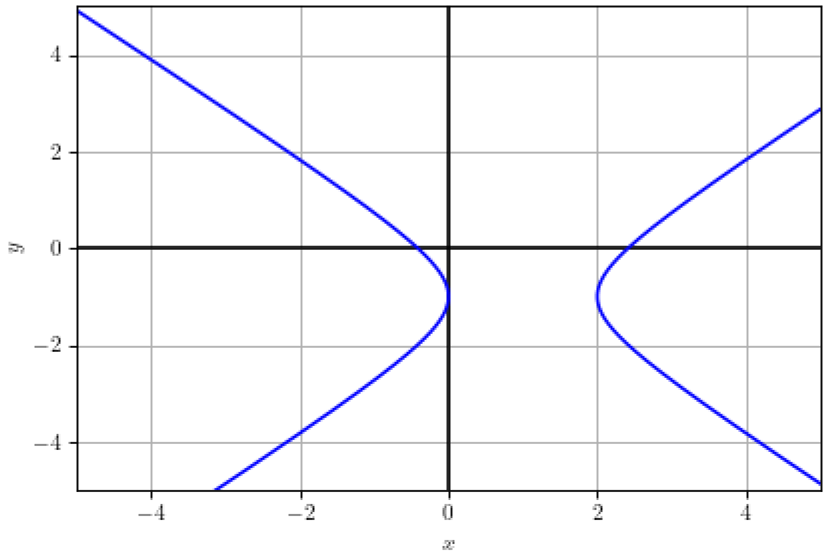

Figure 8: Plot of the hyperbola $x^2 - 2x - y^2 - 2y = 1$.

As for designing rotated conics, such as the rotated hyperbolic equation $10x^2 + 20xy - 11y^2 = 40$, the trick that I used was to set one of the variables: x or y, as the parameter t. In this example, I set y as t. From there, I solved for x using the quadratic formula, where I ended up with two solutions. Like before, I could only plot one of the parts of the hyperbola at a time, so I set two equations linked to the same variable to represent both parts. Below is the Python code I used to represent the hyperbola, as well as an image with the parabola. The red line represents the rotated x-axis, while the purple line represents the rotated y-axis.

```python
import numpy as np
import matplotlib.pyplot as plt
plt.rc('text',usetex=True)
plt.rc('font',family='serif')

# Rotated Hyperbola
t0 = np.linspace(-2500,2500,200)
t = np.linspace(-10,10,2000)
x0 = np.zeros(len(t0))
y0 = 0*t0
x1 = -t + np.sqrt(4 + 2.1*t**2)
x2 = -t - np.sqrt(4 + 2.1*t**2)
xp = 0.4*t0
yp = -2.5*t0

fig = plt.figure()
plt.plot(t0,y0,color='k')
plt.plot(x0,t0,color='k')
plt.plot(t0,xp,color='r')
plt.plot(t0,yp,color='m')
plt.plot(x1,t,color='b')
plt.plot(x2,t,color='b')
plt.xlabel("$x$")
plt.ylabel("$y$")
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.grid()
plt.show()
fig.savefig('rotated.png', dpi = 80, transparent=True, format = 'png')
```

Figure 9: Code used to plot the rotated hyperbola $10x^2 + 20xy - 11y^2 = 40$.

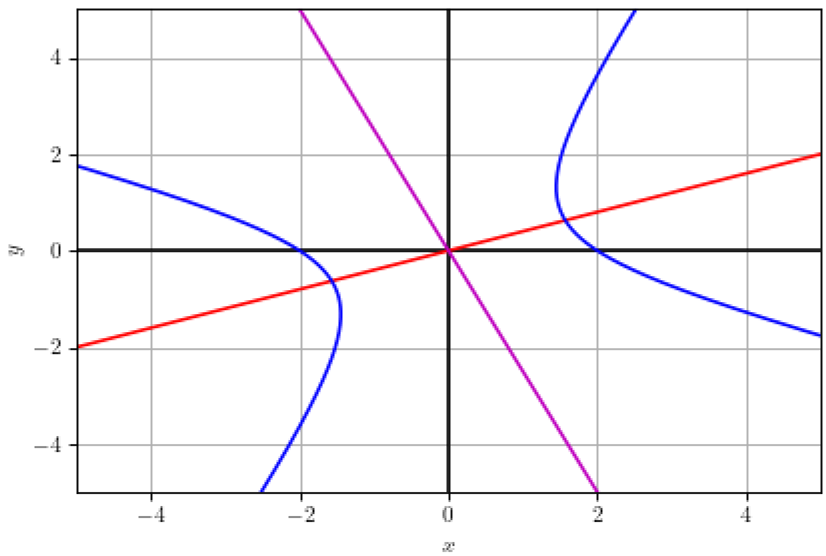

Figure 10: Plot of the rotated hyperbola $10x^2 + 20xy - 11y^2 = 40$.

For both pieces of code, the Python packages NumPy and MatPlotLib are used; NumPy is used to produce the arrays needed to plot the functions, while MatPlotLib takes the arrays and puts the functions onto a plot. Directly below the first two lines of code is the command `plt.rc`. This is used to render the plot description in LaTeX, as well as set the font for the plot. Both of the plots use a figure so that the images can be saved as .png files with a transparent background, and there are x- and y- axes that are common between both plots, which are respectively given by the commands `plt.plot(t0,y0,color='k')` and `plt.plot(x0,t0,color='k')`.

When we look at Figures 7 and 8, we can see that there are three equations for t, `t0`, which is used to generate the x- and y-axes, `t1`, which is used to generate the right arch of the hyperbola, and `t2`, which generates the left arch. The values x1 and y1 are parametric equations that depend on t1, and the values x2 and y2 are parametric equations that depend on t2, so in essence, the command `plt.plot(x1,y1,color='b')` generates the right arch in the plot, while the command `plt.plot(x2,y2,color='b')` generates the left arch in the plot, both of which are in blue. The trigonometric identity $\sec^2(t) - \tan^2(t) = 1$ is used to derive the parametrizations for the x- and y-coordinates of the plot, more specifically, we set $x = \sec(t) + 1$ and $y = \tan(t) - 1$.

As for Figures 9 and 10, we can see that there are only two equations for t, `t0`, which is used to generate the x- and y-axes, and `t`, which is used to generate the two arches. However, for this plot, a different parametric equation is used; the trick was to set y=t, where t could be any number, and solve for x using the quadratic formula. We had two solutions for x, $x_1 = -t + \sqrt{4 + 2.1t^2}$, and $x_2 = -t - \sqrt{4 + 2.1t^2}$. The first solution, which is labeled as `x1` in the code, gives the x-coordinate of the right arch, while the second solution, labeled as `x2` in the code, gives the x-coordinate of the left arch. However, since t is allowed to be any number, we only need one equation for the y-coordinate that can be used for both arches, whereas we needed two equations for the x-coordinates of the arches. There are also two rotated axes; the rotated x-axis in red, which is given by the equation $y = 0.4x$ and shown on the plot by the command `plt.plot(t0,xp,color='r')`, and the rotated y-axis in purple, which is given by the equation $y = -2.5x$ and shown on the plot by the command `plt.plot(t0,yp,color='m')`. Notice that `t0` functions as the x-value of both lines, and `xp` acts as the y-value for the equation $y = 0.4x$, whereas `yp` acts as the y-value for the equation $y = -2.5x$. These axes help aid students in drawing the rotated hyperbola.

Conclusion

Over the course of the entire four month co-op term, I have worked on designing the questions for the Grade 11, Grade 12, and Calculus I mathematics curriculum, and I have learned new skills in LaTeX and teaching in mathematics, as well as honed my current skills in Python and LaTeX. I feel that Hypatia Systems' CheckMath system is extremely helpful for offering students an interactive and comprehensive experience in learning mathematics. More features will be added to Hypatia Systems and CheckMath as more issues and incompatibilities are discovered and more advanced topics are introduced, such as multivariable calculus, differential equations, and complex analysis.