Judah Adefuye (6KNS) Candidate number: 0360

Nea programming project

Contents

[Contents 2](#_Toc132739972)

[Analysis 4](#_Toc132739973)

[Background to project 4](#_Toc132739974)

[Current system 4](#_Toc132739975)

[Identification of end-user: 6](#_Toc132739976)

[Prospective users: 6](#_Toc132739977)

[Research 7](#_Toc132739978)

[Questionnaire 7](#_Toc132739979)

[Observation 10](#_Toc132739980)

[Interview 14](#_Toc132739981)

[Diary of research collected 15](#_Toc132739982)

[Question Generation and testing students understanding 16](#_Toc132739983)

[Testing Students Understanding 18](#_Toc132739984)

[Conclusion and Proposal 21](#_Toc132739985)

[Set of objectives 23](#_Toc132739986)

[Proposed solution details 25](#_Toc132739987)

[Acceptable Limitations 25](#_Toc132739988)

[Design 26](#_Toc132739989)

[Designing Question Generation 26](#_Toc132739990)

[Question generation overview 26](#_Toc132739991)

[Line Generation 26](#_Toc132739992)

[Line generation algorithm 30](#_Toc132739993)

[Getting Nodes and finding cycles 33](#_Toc132739994)

[Graph to Graph conversion algorithm 35](#_Toc132739995)

[Cycle detection algorithm 39](#_Toc132739996)

[Inequalities 41](#_Toc132739997)

[Objective function (See objective line) 41](#_Toc132739998)

[Questions asked 45](#_Toc132739999)

[Optimizing a function subject to constraints 48](#_Toc132740000)

[#Description of data structures 54](#_Toc132740001)

[High level overview of linear programming question generation 55](#_Toc132740002)

[Design of user interface 56](#_Toc132740003)

[Line Drawing 59](#_Toc132740004)

[#System security and integrity of data 70](#_Toc132740005)

[Implementation 71](#_Toc132740006)

[lines 71](#_Toc132740007)

[lines\_for\_lp 75](#_Toc132740008)

[Linprog 77](#_Toc132740009)

[graph\_nodes 80](#_Toc132740010)

[objective\_line 84](#_Toc132740011)

[func\_plot1 86](#_Toc132740012)

[questions 89](#_Toc132740013)

[freq\_used 95](#_Toc132740014)

[main 96](#_Toc132740015)

[Testing 113](#_Toc132740016)

[Evidence of testing 120](#_Toc132740017)

[Evaluation 140](#_Toc132740018)

# Analysis

## Background to project

Among the subjects I take at A level is further maths. When revising for our mock exams I found it difficult to get extra resources on several topics in decision maths and Linear Programming in particular. Upon further investigation I discovered that I wasn’t the only one facing such a problem. This prompted me to undertake this project with its aim being to help Yr 12 further maths students revise Linear Programming.

## Current system

At the moment students majorly revise using an online resource called Integral which is limited in number of exercises (usually two per topic) and our textbooks are outdated as the specification was recently changed by AQA. There are only 3 past papers we have that test on the new specification.

My project will allow for access to many more questions and also allow students solve the questions on the computer minimalising the need for paper hence slowing climate change. All this will ensure students test their understand of concepts easier.

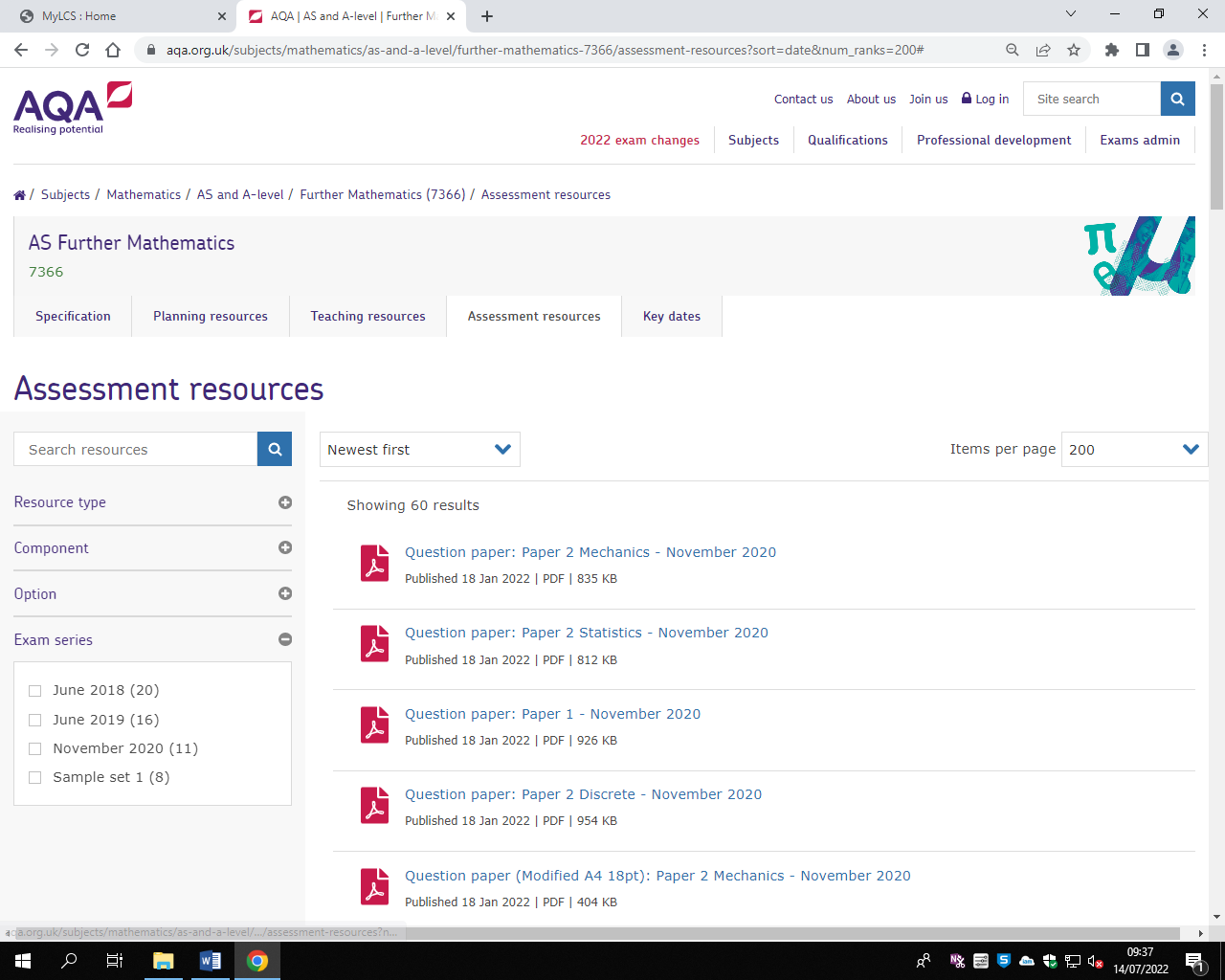


Image showing the only 3 exam series available from AQA’s website.



image showing the only two exercises integral has for a topic.

**Data Flow Diagram for Current System**

**Level 0 DFD**

Diagram

Description automatically generated

An overview of the Data flows of the way students revise

**Level 1 DFD**

**Diagram, schematic

Description automatically generated**

An overview of the ways students revise currently along with data stores.

## Identification of end-user:

Mr Atsog: An A-level maths teacher currently at Littleover community school. Having taught maths for a few years, he would be able to provide valuable feedback as to the efficacy of my tool. Moreover, it could be integrated into his lessons if deemed helpful to students.

## Prospective users:

Any Yr 12 and even Yr 13 further maths students willing to test their understanding of the basics of linear programming should find my tool useful. Teachers that teach the AQA AS Discrete Maths option and any others that would teach linear programming would also find it useful. Maths hobbyists should have fun trying out my tool.

## Research

In order to better understand the resources lacking and problems with the current system I undertook research in the form of a questionnaire sent to the current Yr 12’s , observation of the current system and an interview with my end-user.

## Questionnaire

I created a digital questionnaire using Microsoft Forms and sent it to the 11 other members of our further maths group. Though quite few in number (members of our group), the huge disparity in opinions would suggest the favoured opinions would still be chosen among a larger population. Here are the results:

Majority say that not enough resources are available to test their understanding.

Majority say working on a computer will be better - Though not what will be done in an exam, this will speed up the revision process, allow user errors to be spotted easily and most importantly help with climate change (as less paper used will lead to less trees cut and so on).

Though split, a tool that will benefit 50% of any population will be of immense benefit. Normally I would expect the number of those that believe there are not enough past papers to test understanding to be more but our class consisted mostly of people incredibly good at maths.

This shows how beneficial a tool that can generate linear programming questions will be to students.

**Conclusion**

The results suggest that the number of resources available are not enough. With only one outlier, it clearly suggests that being able to solve questions on the computer will be a good thing to have. Results also suggest that student need more questions with which to practise as they’ll feel prepared if they weren’t limited by number of questions.

## Observation

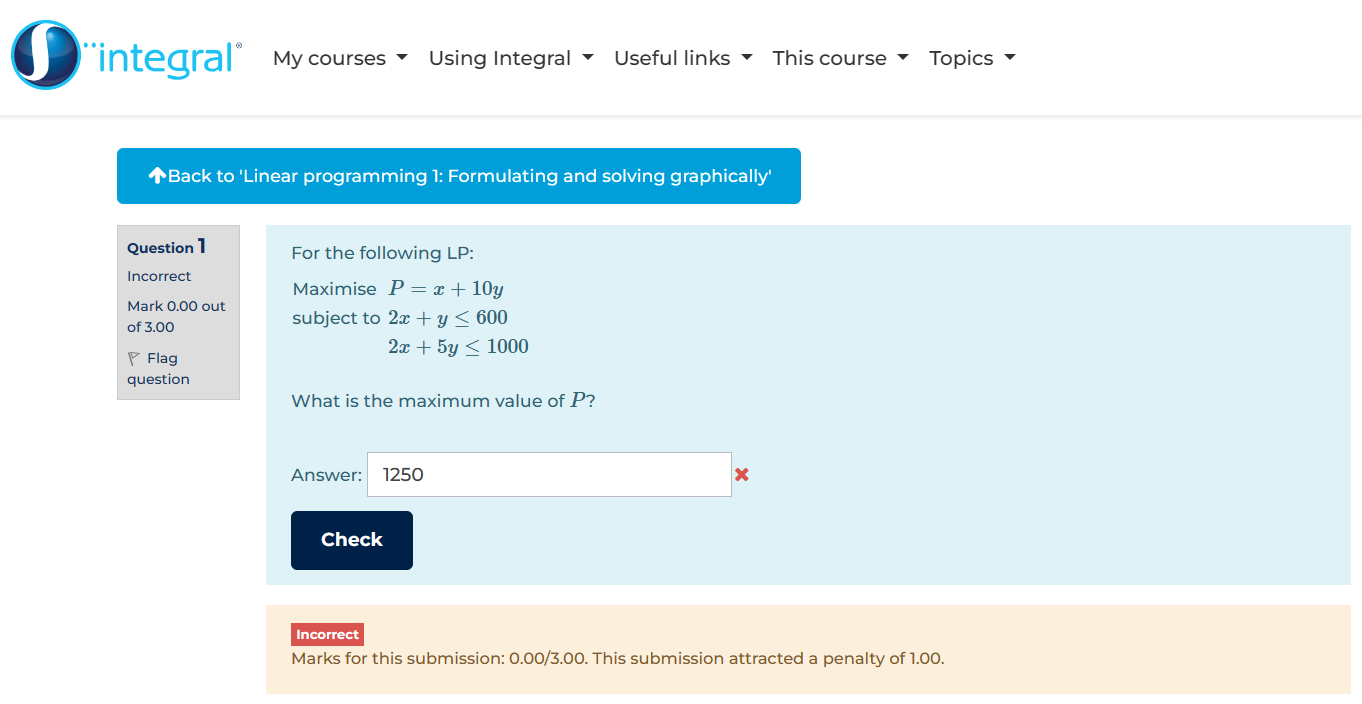
At the moment students majorly revise using an online resource called Integral which is limited in number of exercises (usually two per topic) and textbooks are outdated as the specification was recently changed by AQA. There are only 3 past papers that test on the new specification.

Currently the only tool remotely similar to mine is the test on integral.

The main problems with it and other websites online are:

1. **The underpinning knowledge of graphing the problems is not tested**. Students are expected to solve these problems on paper and just input a final answer which is checked. Students are not able to test their ability to draw graphs as the websites don’t have any functionality to test the ability of students to draw graphs and as such students might miss out on many marks, or by stroke of luck they might get the answer and assume what they are doing is right.

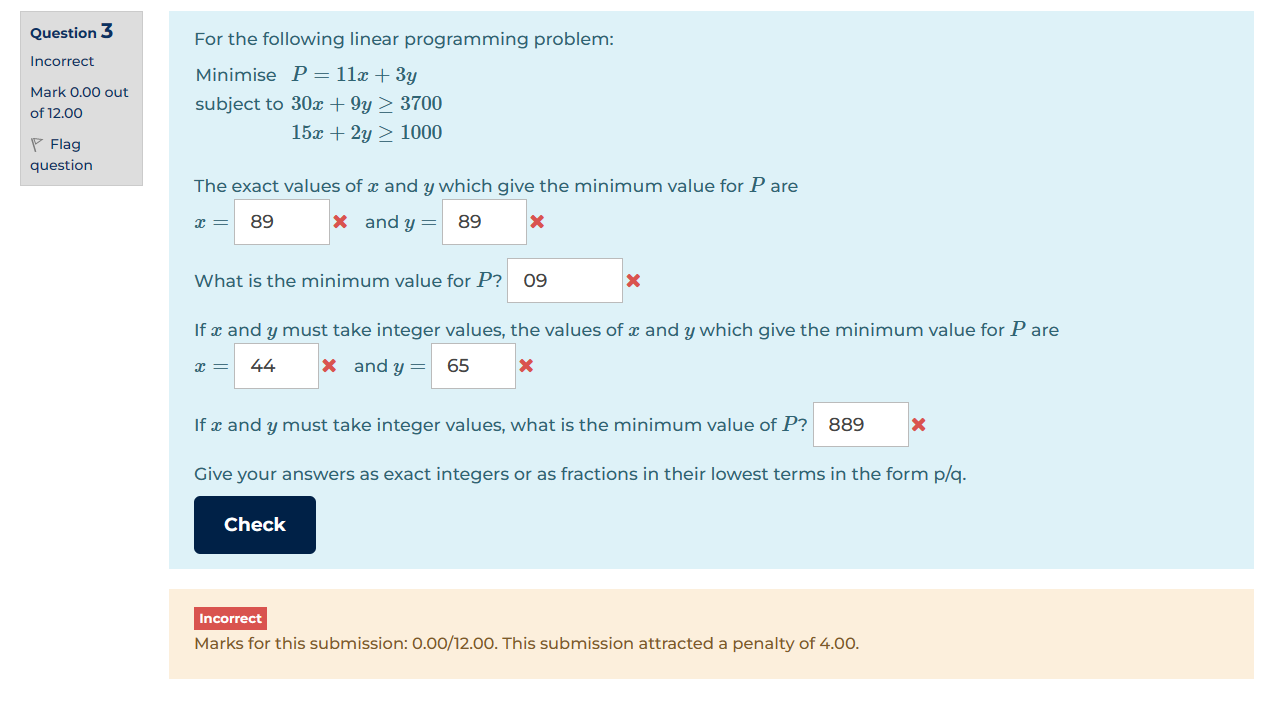
**My solution must have an adequate way(or ways) of testing students abilities to draw graphs.**



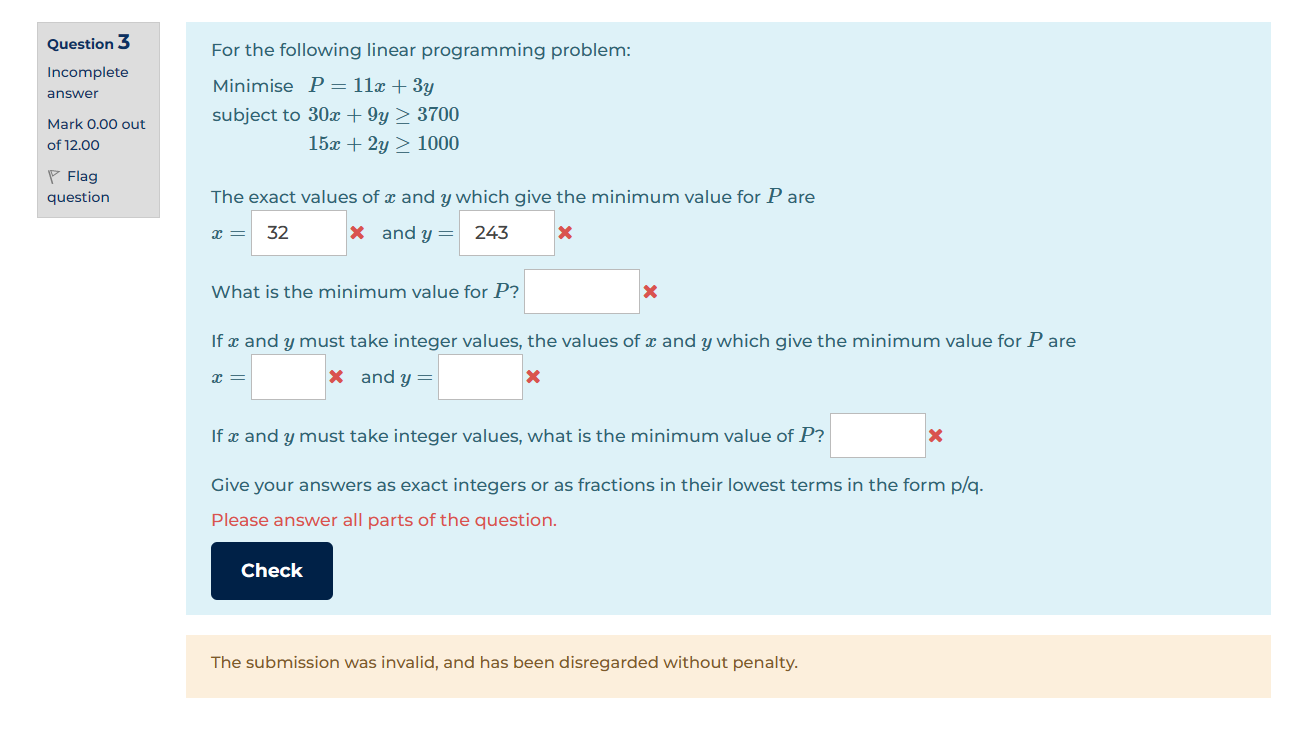
Final answer inputted without testing students ability to draw graphs

1. **The final solution is checked all at once.** This method though incredibly suspenseful could both lead to happiness and utter devastation. As evidenced by my not so perfect try (and i didnt even purposely try and fail the question :( ) ,students arent allowed to submit answers to parts of the question ie only the final answer is submitted. This is incredibly time inefficient as students might have made an error early on in solving the question and given the nature of these problems, the error cascades and affects the final answer.

**My solution should have a methodical way of answering questions allowing for user error to be spotted each step of the way**. **The final answers to linear programming problems should not be answered immediately.**

An extension could be having an option for the brave and suspense hungry students to tackle a question all in one go 

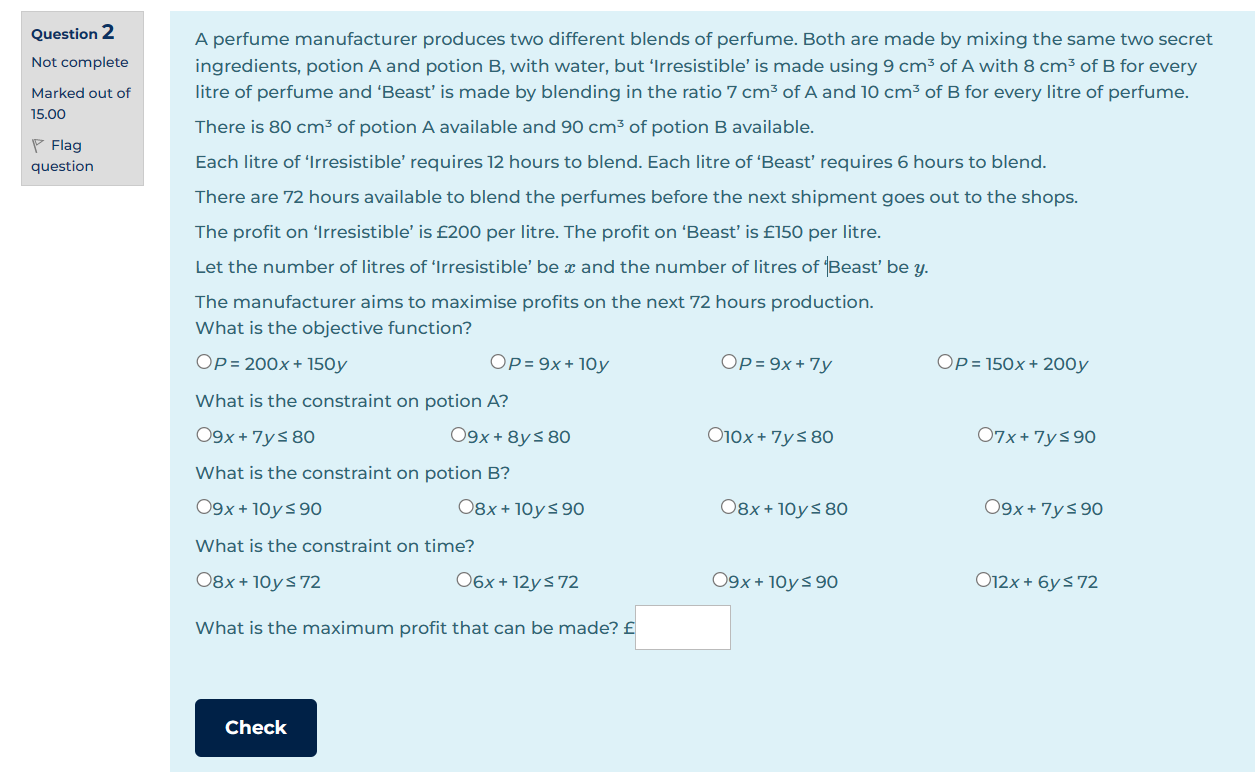
Entirely failed submission leading to me having to entirely redo the question

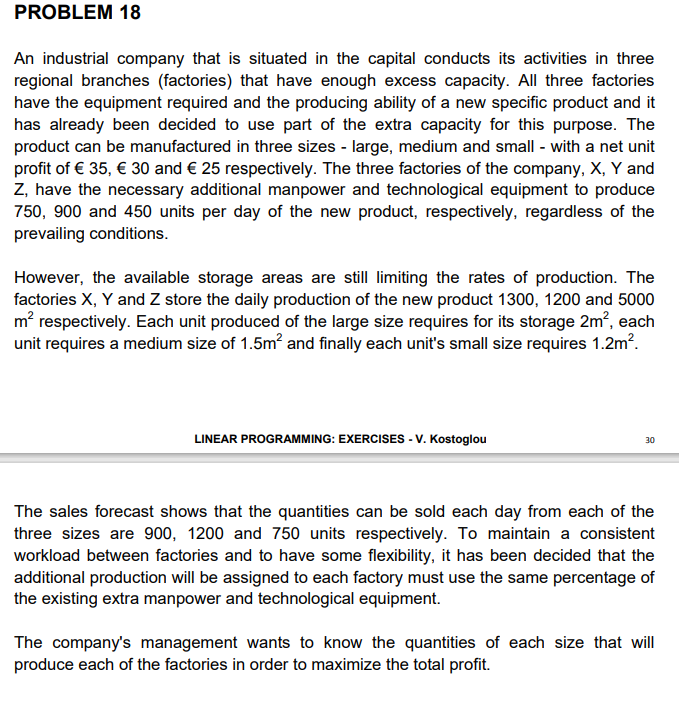


Prompt saying to answer all parts of the question before being allowed to check

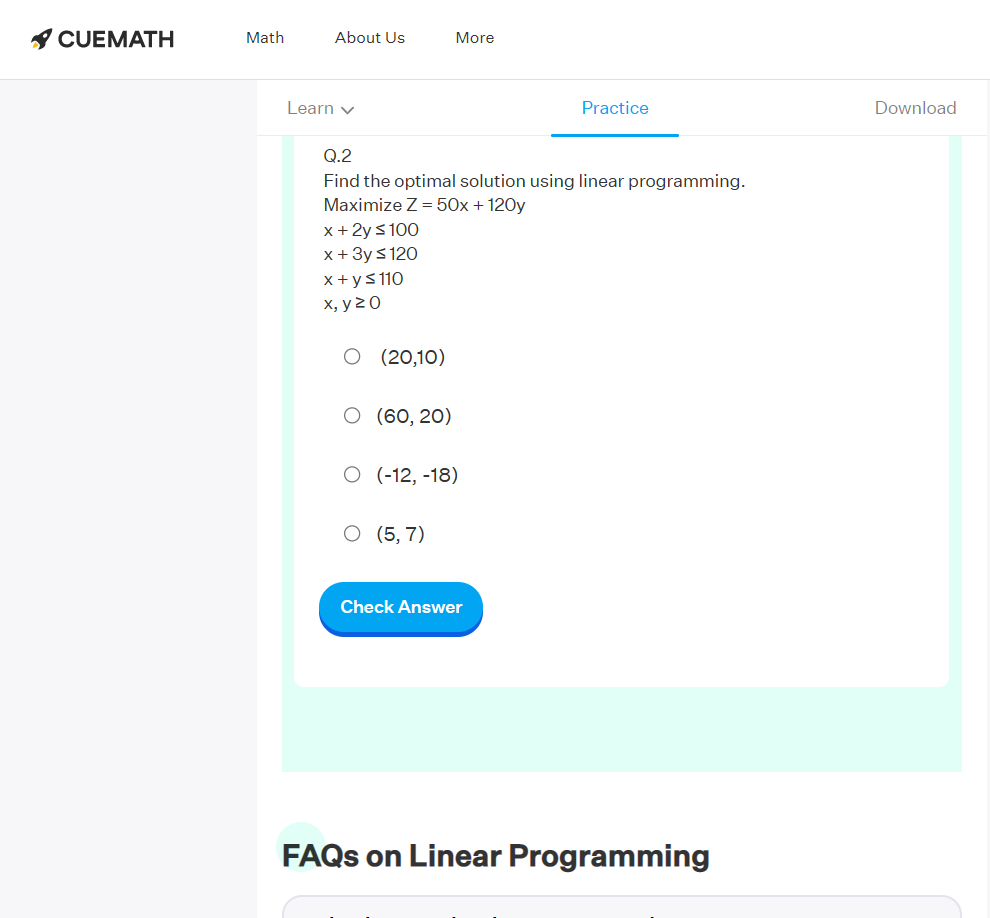
1. **Another problem will be the convoluted nature of the problems found on the internet**. On most websites, the lengthy wording of the questions will be a turn-off for most struggling students. These also don’t adequately or efficiently test the ability of students to formulate and solve the problem.

And for the few that are not long-winding they are either a victim of my second major problem or just are not enough in number to reliably test the understanding of students.



****

Convoluted problem

****

Website with only two questions

1. **The final problem is the limited number of questions**. Unlike most areas of maths, it isnt as easy to just formulate a linear programming problem that works. Though not explicitly stated online, i believe this would be the main reason there is a shortage of linear programming questions. This limit in number of questions would be the major problem I try to tackle.

**My solution should be able to generate a linear programming question.**

## Interview

I interviewed Mr Atsog asking him questions that could give insight on the current system and ways to better it.

**What are the things you would expect in a good revision tool?**

*I’d expect it to have key topics and some notes on the topics for revision. I would also want there to be questions with worked solutions as the worked solutions will enable for better understanding and would allow students to familiarise themselves with the format in which to answer exam questions. Also, it would be nice to have quickfire quiz kind of things as students would like some gaming element. It would be a plus if teachers could set questions for the students and if the tool had links to the specification so that students are sure they are doing what needs to be done. It will be good for the tool to have a modern look to it as at times clunky looking interfaces turn off students.*

**What are the downsides of the current system of revision for discrete maths and linear programming in particular?**

*If searched for properly, there are enough resources to test understanding of the topics. However, it would be better if the resources were not scattered across the internet. The current textbooks don’t cover the syllabus in totality and there are not enough past papers to fully test the understanding of students. Also, there could be more questions testing the basics of topics to ensure understanding.*

**Would it be beneficial to have a tool that generated questions on any topic in general?**

*Immensely. Not only would it save my time searching up questions, but it will also ensure students are not limited by number of questions, moreover it will lower the risk of setting repeat questions for tests. Such a tool would drastically reduce the effort needed by a teacher especially if one could extract these questions and set them as tests for students. If you will make one, I’d suggest making one with varying question types. However, such a tool will probably be limited in terms of capability as I don’t see one being able to generate exam style questions any time soon.*

**What do you think of a revision tool that automatically grades students’ answers?**

*If coupled with the ability of teachers to monitor the progress of students, such a tool will be incredibly beneficial. As of now the ones that offer such functionality are budding in terms of overall benefit. Integral for example allows students to attempt a maximum of 3 tests for each topic, which might not be enough, but more importantly, these tests are fixed and don’t change. Final answers are inputted and just compared to the answer programmed in. Students have to check through their entire work to find errors which I find highly inefficient. It also opens the possibility of cheating as questions are the same for all students.*

**Highlights of interview**

* Question generation is useful.
* Variety of questions generated is a plus.
* It will be incredibly difficult to generate exam style questions.
* There should be a modern looking GUI with ease of user input.
* Will be good If teachers are able to monitor progress of students.
* There should be a way for students to identify precise location of errors.
* Some kind of gaming element will help students.

## Diary of research collected

Interview carried out on 15/06/2022

Survey collated on 15/06/2022

System observed on 15/06/2022

## Question Generation and testing students understanding

As one of the main problems I’ll be facing is question generation, it will be right to analyze the feasibility and scope of question generation for linear programming. This analysis should help inform my Design and will also assess the viability of this project as it is on question generation that my solution hinges.

Linear programming essentially constitutes an objective function (some equation) optimized while satisfying constraints (in the form of inequalities).

To make a question i should be able to:

1. **Generate lines that should be turned into inequalities or just generate inequalities.**

Text, letter

Description automatically generated

Example inequalities from Integralmaths.org

This criterion nudges me to implement my solution with the object-oriented paradigm as I will probably need to have an “attribute” holding the coefficient of each variable that forms the line and whatever is on the right-hand side of the line.

1. **Ensure these inequalities form a closed area. This closed area will be the feasible region i.e region that satisfies all constraints.**

Chart, line chart

Description automatically generated

Feasible region (closed area)

This might prove the most challenging bit to accomplish. To do this I could potentially use algebra and geometry starting from any point to try and form some kind of 2D polygon to ensure there is a cycle. However, with my low expertise in geometry, I would be leaning towards the idea of modelling the intersections of lines as nodes and modelling the graph as the abstract data structure “graph” and maybe running some cycle detection algorithm to check if there is a cycle.

1. **Get an objective function - in this case “x + y”.**

This will be the function that will be optimized and must satisfy all inequalities. This could just be a specialized line.

1. **Finally optimize the objective function subject to some constraints (inequalities).**

Chart, line chart

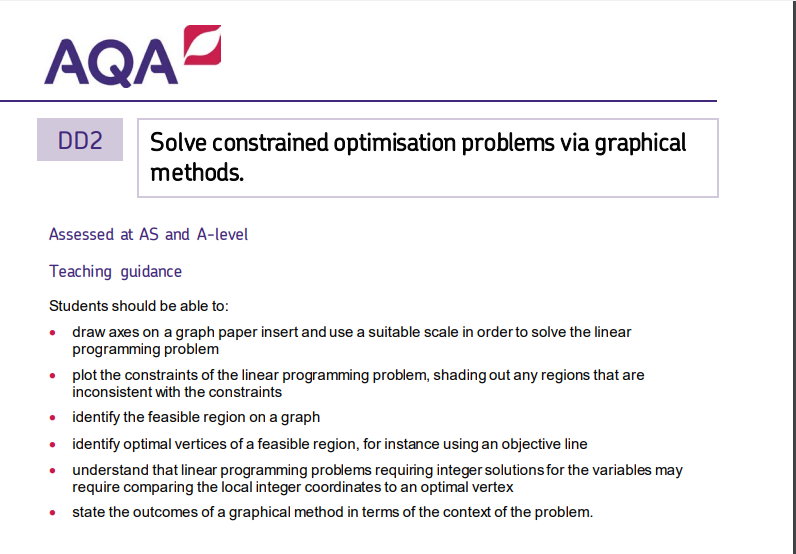
Description automatically generated

When solved the maximal value of “x + y” occurs at this vertex

Internally I would probably just evaluate the objective function at vertices of the feasible region and then pick the optimal solution.

### Testing Students Understanding

AQA teaching guidance for solving linear programming problems

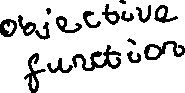


As outlined by the teaching guide for linear programming and based on prior knowledge and research, there are two ways with which students should be able to find the optimal solution.

The first will be to draw the objective function on the graph and depending on whether the function is to be minimized or maximized, shift the objective function until it meets one of the vertices of the feasible region. If to be maximized it should be shifted from above the feasible region until it meets a vertex and that will be the optimal solution if to be minimized it’s the last vertex it comes in contact with that will be optimal solution.

Chart, line chart

Description automatically generated



The second way will be to evaluate the objective function at each of the vertices and then choose the vertex of the feasible region that either maximizes or minimizes the objective function.

The essential skills needed by students to be able to do this will be:

* Drawing the lines

To draw lines students should be able to get the x and y intercept of lines and then connect them to draw a line.

* Identifying the area to shade of lines (ie graphing inequalities)
* Identifying the feasible region and the vertices that make it
* Finding the coordinates of these vertices
* Drawing the objective function
* Shifting the objective function to the optimal vertex
* Being able to evaluate the objective function at the vertices
* Obtaining the optimal solution

I can group these skills into three categories;

1. **Graphing a LP problem**

Consisting of drawing the lines (identifying the x and y intercepts of lines) and identifying the area to shade of lines.

1. **Solutions using the objective line method**

Identifying the feasible region, finding the coordinates of these vertices, drawing the objective function, Shifting the objective function to the optimal vertex and obtaining the optimal solution

1. **Solutions using the vertex method**

Identifying the feasible region, finding the coordinates of these vertices, evaluating the objective function at each vertex and obtaining the optimal solution.

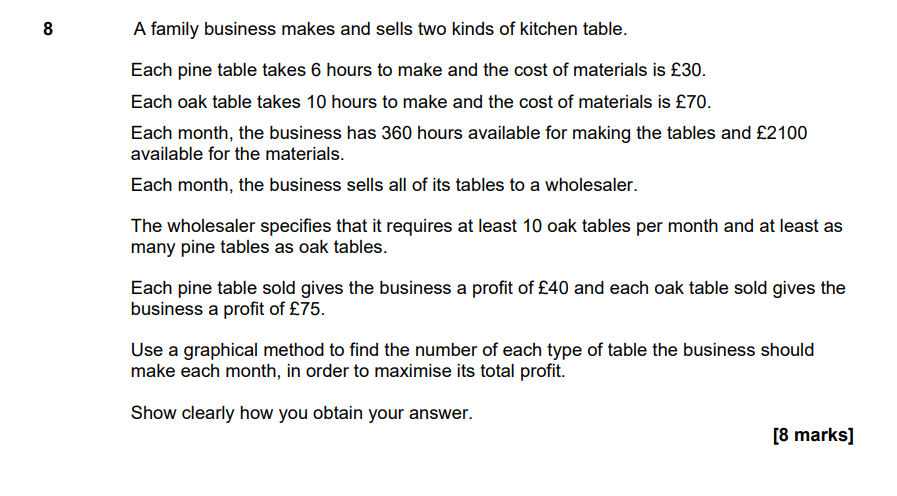
It is on these 3 “subtopics” ill base the questions asked in the tool.

**Conclusion**

This analysis has given me a clear route to take in regards to developing my solution. Question generation seems feasible enough to do with the time currently available.

Satisfied that this problem is computationally solvable, I go ahead to set objectives.

## Conclusion and Proposal



Exam question from an AQA specimen paper

At the end of the day students should be able to solve problems such as the one above. Generating questions of this nature with my current skill and level of knowledge will be impossible as correctly highlighted by Mr Atsog as such I will leave out formulating a linear programming problem.

The focus of this tool will be on a revision tool for the basics of linear programming helping a teacher to test the understanding of students of inequalities, their ability to graph them and solving a linear programming problem either with the vertex method or the objective line method. This will be used primarily preliminary to tackling exam questions.

Questions generated should be of sufficient complexity to be useful in testing students.

It will be nice to have a teacher being able to set tests and to monitor students. However, as the primary purpose of my tool is to help the teacher prepare students before taking on exam questions, I’ll consider adding a database and all associated functionality i.e teacher setting a practice test and monitoring students’ progress as extension objectives.

**New System DFD level 0**

An overview of the data flow if my tool will be implemented in the current system

Diagram

Description automatically generated

**New System DFD level 1**

Diagram, schematic

Description automatically generated

## Set of objectives

Drawing on the conclusions of my analysis and research i have settled on the following objectives (**Black**-Critical, Green – Extension):

1. **Tool should be able to Generate lines and inequalities.**

* Lines generated should contain values in a specific range eg lines could have a maximum x being 100 and as such beyond that the line should no have no other value.
* Lines should be displayed properly in the form ay + bx = c
* Lines that won’t be adequately represented with conventional format should be catered to.
  + Lines with either a or b being negative, zero or one should be displayed optimally eg -1x + 0y = -30 should be more adequately displayed as

-x = -30.

* Inequalities should be represented the same as lines but with appropriate inequality symbol ie one out of; >=, <= , > , <.

1. **Inequalities used in questions must form a closed cycle.**
   * When inequalities are drawn on graph there should be at least one cycle.
   * If more than one cycle, cycle with highest number of edges should be chosen for the question.
2. **Tool should be able to make an Objective function (function that would be optimised subject to constraints)**

* Objective function should be draggable when drawn on graph.
* Program should be able to “decipher” when objective function is at the optimal vertex to linear programming problem ie there should be some way to know when objective function is at optimal vertex.

1. **Program should be able to minimize or maximize an objective function subject to constraints (inequalities), returning optimal solution and coordinate at which solution occurs.**
2. **Program should be able to plot lines and inequalities on a graph accurately.**

* Inequalities drawn must be labelled properly.

1. **User should be able to select a topic of choice to study (under Linear Programming) or general revision to practice.**

* One out of:
  + Graphing a LP problem
  + Solutions using objective method
  + Solutions using the vertex method.
* Graphing a LP problem should be done first by user as the other sections build on what is done on the graph that is drawn.
* When any of them Is chosen the corresponding screen should be displayed and questioning on the topic should begin.
  + Questions on Graphing a LP problem should involve; Drawing the lines (identifying the x and y intercepts of lines) and identifying the area to shade of lines.
  + Questions on Solutions using the vertex method should involve; Identifying the feasible region, finding the coordinates of these vertices and obtaining the optimal solution.
  + Questions involving the objective solution Identifying the feasible region, finding the coordinates of these vertices, drawing the objective function, Shifting the objective function to the optimal vertex and obtaining the optimal vertex.

1. **Tool should be able to test students with various formats of questions.**

* Multiple choice, fill in the blank and questions that require the user to click on a graph to answer should be generated by the tool.

1. **User should be taken step by step in answering the questions.**

* Questions should be broken into smaller parts that will be solved to allow students work through questions logically and to easily trace sources of student error ie final answer to LP problem shouldn’t be inputted immediately all other aspects should be done first.
* User should be able to submit answers when done by pressing a submit button when done answering.
* User should be let known if answer is correct or wrong when done answering any question.

1. **My program should have a GUI that allows for graphs to be displayed without affecting other elements onscreen.**

* This GUI should seamlessly incorporate graph plotting functionality.
* Lines and objective function drawn should be labelled on the graph.
* There should be tools to aid in viewing graphs.
  + A zoom tool allowing to magnify sections, and a shifting tool allowing plots on axis to be shifted.

1. **Questions should be displayed on the GUI allowing for users to input answers in appropriate manner relevant to the question (e.g radiobuttons for Multiple choice questions)**
2. **Teacher should be able to monitor the progress of students and set practice tests.**

## Proposed solution details

Python 3.9.6 will be used to develop tool.

NumPy, Matplotlib and CustomTkinter will be external libraries I will use in my solution.

NumPy will be used for a variety of mathematical calculations.

Matplotlib will be primarily used for graphing and all related functionality on graphs.

CustomTkinter will be used to develop the Graphical User interface.

System on which tool will run will need at least a windows 8 computer or macOS 11 Big Sur

No other hardware is necessary.

## Acceptable Limitations

* Minimizing problems with unbound areas

It is possible for minimizing problems to have unbound areas when drawn out graphically. Unfortunately, at this point in time I can’t think up an algorithm to adequately extrapolate unbound areas and use them to generate inequalities.

# Design

The design will be broken into two parts - designing the question generation and functionality behind it and then designing the user interface.

My aim will be to make modules that primarily tackle one problem and will easily integrate with others in the final program. This will ensure my program is easy to maintain and debug while removing the likelihood of having to correct entire swaths of code to fix an error.

## Designing Question Generation

### Question generation overview

As highlighted in the analysis section in order to get a question to give to students I should be able to:

* Get some lines or inequalities
* Ensure they form a closed cycle
* Get an objective function
* Be able to optimize the objective function subject to the inequalities

### Line Generation

I decided to deal with the inequalities first. Inequalities are just conventional lines but with a comparison symbol rather than equal to. Lines are generally expressed in the form “ *y = mx + c* “ with m being the gradient of the line and c being the lines y-intercept . They can also be expressed in the form “ *ax + by = c*” where a,b and c can be any number. I decided to choose the latter as it is what students will make use of in linear programming. I realise i would need to find a way to uniquely hold the values of a, b and c for each line and decided to make a lines class where each line object will have:

* an x attribute (coefficient of its x variable that of the line above will be a ),
* a y attribute (coefficient of its y variable that of above will be b)
* A rhs (right hand side) attribute (that of above will be c)
* And finally a status attribute -This will be either “=”,”>=”,”<=” and so on depending on inequality or equation

I later decide to add in more attributes in order to work around problems and make my solution more robust. These are:

* m - gradient of line
* c - yintercept of lines
* xc -xintercept of line

Graphical user interface, application

Description automatically generated

This class will be in a module called lines.

Lines will be initalised with parameters x,y, status and rhs. The rest will be calculated as shown by following pseudocode (IN RED):

**class Lines**

**subroutine new**(x,y,rhs,status)

self.x ← x

self.y ← y

self.rhs ← rhs

self.status ← status

**if** self.y = 0 **then**

self.m ← infinity

self.c = None

**else**

self.m ← -self.x/self.y

self.c ← self.rhs/self.y

**endif**

**if** self.x ≠ 0 **then**

self.xc ← self.rhs/self.x

**else**

self.xc ← None

**endif**

**endsubroutine**

**subroutine output**(self,x)

**if** self.y ≠ 0 **then**

y ← (self.rhs - self.x \* x)/self.y

**return** y

**else**

**return**"NA"

**endif**

**endsubroutine**

**endclass**

I just rearrange the equations and get a line’s gradient and y intercept. I also take care of lines that don't have a y or x intercept.

I will further the class by adding an output() method to return the given y of the line for any input x. It will return “NA” for lines such as x = 80 as there is no y outuput

I also and importantly as stated in objective 1 made a method \_\_str\_\_() to ensure lines are displayed appropriately. An easy solution will be just to have such as the following:

subroutine \_\_str\_\_()

**return** REAL\_TO\_STRING(self.x) + "x + "+ REAL\_TO\_STRING(self.y) + "y"+ self.status + INT\_TO\_STRING(self.rhs)

end subroutine

The problem with this is there will be lines that could have negative attributes or attributes that are 0 which could result in a line such as **0x + -2y = 120** . This should be more appropriately displayed as  **-2y = 120**

**Subroutine \_\_str\_\_**(self):

y ← REAL\_TO\_STRING(self.y)

x ← REAL\_TO\_STRING(self.x)

#taking care of 0

**if** x = "0"and y = "0" **then**

**return**"Invalid line"

**elseif** x = "0"and y ≠ "0" **then** #**if** x part of equation = 0

**if** y.strip("-") ≠ "1" **then**

**if**"-"not **in** y **then**

**return** y+"y"+ self.status + self.rhs

**elseif** "-"**in** y **then**

**return** -y.strip('-') + "y "+ self.status + self.rhs

**endif**

**elseif** y.strip("-") = "1" **then**

**if** "-"not **in** y **then**

**return** y.strip('1') + "y "self.status + self.rhs

**elseif** "-"**in** y **then**

**return** -y.strip('-1') + "y "self.status + self.rhs

**endif**

**elseif** y.strip("-") ≠ "1" **then**

**if** "-"not **in** y **then**

**return** y + "y "+self.status + self.rhs

**elseif** "-"**in** y **then**

**return** -y.strip('-') + "y "+ self.status + self.rhs

**endif**

**elseif** y.strip("-") = "1" **then**

**if** "-"not **in** y **then**

**return** y.strip('1') + "y "+ self.status + self.rhs

**elseif** "-" **in** y **then**

**return** -y.strip('-1') + "y "+ self.status + self.rhs

**endif**

**endif**

**elseif** x ≠ "0"and y = "0" **then** #y part of equation = 0

**if** x.strip("-") ≠ "1" **then**

**return** x + "x "+ self.status + self.rhs

#

**elseif** x.strip("-") ≠ "1" **then**

**return** x + "x "+ self.status + self.rhs

**elseif** x.strip("-") = "1" **then**

**return** x.strip('1') + "x "+ self.status + self.rhs

**elseif** x.strip("-") = "1" **then**

**return** x.strip('1') + "x "+ self.status + self.rhs

**endif**

**elseif** x ≠ "0"and y ≠ "0" **then** #x and y are not = 0

#taking care of 1s and -

**if** x.strip("-") ≠ "1"and y.strip("-") ≠ "1" **then**

**if** "-"not **in** y **then**

**return** x + "x + "+ y + "y "+ self.status + self.rhs

**elseif** "-" **in** y **then**

**return** x + "x - "+ y.strip('-') + "y "+ self.status + self.rhs

**endif**

**elseif** x.strip("-") ≠ "1"and y.strip("-") = "1" **then**

**if** "-"not **in** y **then**

**return** x + "x + "+ y.strip('1') + "y "+ self.status + self.rhs

**elseif** "-" **in** y **then**

**return** x + "x - "+ y.strip('-1') + "y "+ self.status + self.rhs

**endif**

**elseif** x.strip("-") = "1"and y.strip("-") ≠ "1" **then**

**if** "-" not **in** y **then**

**return**x.strip('1') + "x + "+ y + "y "+ self.status + self.rhs

**elseif** "-" **in** y **then**

**return** x.strip('1') + "x - "+ y.strip('-') + "y "+ self.status + self.rhs

**endif**

**elseif** x.strip("-") = "1"and y.strip("-") = "1" **then**

**if** "-" not **in** y **then**

**return** x.strip('1') + "x + "+ y.strip('1') + "y "+ self.status + self.rhs

**elseif** "-" **in** y **then**

**return** x.strip('1') + "x - "+ y.strip('-1') + "y "+ self.status + self.rhs

**endif**

**endif**

**endSubroutine**

Though quite cumbersome to write this method would allow for lines to be adequately represented throughout the program. Another class will inherit from this class this discussed later.

All attributes and methods of this class will be public as data of lines will constantly need to be checked and used for other calculations.

#### Line generation algorithm

This will be a function returning lines that will be turned to inequalities and used for the linear programming problem

* **Generate\_pointsline(x,number)** would be a function with parameters x - an array with the domain of a line and number- the number of lines points should be generated for. This method will generate random points which will be used to draw a line. Returns a list whose elements are lists with a pair of points for **number** number of lines.
* **drawtwopoints(point\_one, point\_two)** would be the function used to draw a line. Using basic maths, i can get the gradient and y intercept of lines and then initialise a line.
* **get\_lineLP(num\_lines, x)** would be the function tying the other two together returning num\_lines, number of lines with a domain of x. If the parameter is only one line, it should just return one line. I did this in order to efficiently get an objective function.Flow chart below.

**Diagram

Description automatically generated**

It will be logical at this point to focus on how to generate inequalities that will form a closed cycle. But i decided to make a function that will plot inequalities and another to optimize a function subject to constraints as at the time i wasnt sure as to how to proceed.

**Plotting Inequalities (See func\_plot1)**

I will use Matplotlib, a visualization library in python, primarily for graphing. Its relative simplicity to use, extensive functionality and my prior experience with it will allow me comparatively easily implement my solution.

To plot lines will be relatively straightforward in matplotlib.

### Getting Nodes and finding cycles

The penultimate thing i will need for a linear programming problem will be a way to generate inequalities that form a closed cycle.

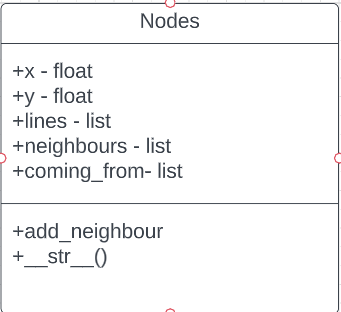
I decided to model the intersections of lines as nodes on a graph (abstract data strutcture) as I believed it will be possible at the time to run a cycle detection algorithm on a graph.

What i will need to do would be to run a cycle detection algorithm on the graph containing the nodes. If any nodes form a cycle the nodes that form the cycle should be gotten (as I could use this as the feasible region of a Linear programming problem). The mean of the coordinates of the nodes can be gotten and used to generate inequalities.

These nodes will be objects of a class **Node**

The nodes are efficiently gotten with the same subroutine **maxobj** above.

#### Nodes class



methods and attributes are public as they will be used in calculations and by other classes

* **X** and **Y** will be the x and y coordinate of the Node
* **Lines** - a list containing lines that meet to form node
* **Neighbours**  - list of neighbouring nodes
* **coming\_from** - list that will be used in cycle detection algorithm. holds the nodes that have been visited immediately before nodes

Example

Lettered Nodes at intersections of lines



For **Node G** its attributes will have the following values:

**X -** 90.9090909

**Y –** 136.363636

**Neighbours –** [B,C,I,H]

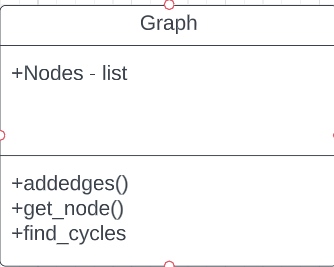
**Lines -** [x +3y <= 500, 5x + 4y <= 1000]

**Class Node\_neighbours**

This will be nodes but with an extra attribute **weight,** the weight of an edge between the neighbour and **node.** This will aid in calculations involving distances between nodes.

#### Graph Class

The graph is essentially stored as an adjacency matrix. As graphs will generally be sparse, this will be a suitable representation.



**Nodes** is the list of all nodes that form the graph. Each of these nodes have a **neighbour’s** attribute holding the neighbours of the node and t heir weights.

### Graph to Graph conversion algorithm

Using the nodes gotten from the max\_obj subroutine, ill then form a graph (abstract data structure) and then run a cycle detection algorithm on it to ensure that inequalities form a closed cycle. The graph will be an instance

This algorithm would be used to set up the graph that the cycle detection algorithm will run on.

class Graph:

    subroutine new(self,nodes)

        self.Nodes ← nodes #nodes that make up graph

        #add neighbours between all nodes that have an intersection between lines attributes

**for** node **in** self.Nodes

**for** vert **in** self.Nodes

**if** node = vert then

**continue** #skip comparing yourself

**else**:

**if** set(node.lines) & set(vert.lines) then

                        node.add\_neighbour(vert)

**endif**

**endif**

**endfor**

**endfor**

        self.addedges()

**endsubroutine**

**subroutine** addedges(self):

      #cuts neighbours to only those immediately to the left or right or above or below node

**for** node **in** self.Nodes #for each node

            new\_neighbs ← [] #new neighbours without others

**for** line **in** node.lines #for each line the node is a part of

                #positive weighted for to the left and above and vice-versa

                pos\_weights ← []

                pos\_neighbs ← []

                neg\_weights ← []

                neg\_neighbs ← []

**for** vertex **in** node.neighbours #for each neighbouring node

**if** line **in** vertex.lines then #if i have the intial line

                        #check the weights and determine if weight is positive or negative

**if** vertex.edge\_weight > 0 **then**

                            pos\_neighbs.append(vertex)

                            pos\_weights.append(vertex.edge\_weight)

**elseif** vertex.edge\_weight < 0 **then**

                            neg\_neighbs.append(vertex)

                            neg\_weights.append(vertex.edge\_weight)

**endif**

**endfor**

                #node to left or above will be node with minimum positive weight and vice

**if** pos\_weights **then**

                    min\_pos ← min(pos\_weights)

**else**

                    min\_pos ← None

**endif**

**if** neg\_weights then

                    min\_neg ← max(neg\_weights)

**else**

                    min\_neg ← None

**endif**

                right\_node ← None #restarting them

                left\_node ← None

                #find the min\_pos node and min\_neg node and call them right node and left node respectively

**for** good\_node **in** pos\_neighbs

**if** good\_node.edge\_weight = min\_pos **then**

                        right\_node ← good\_node

**endif**

**endfor**

**for** other\_good\_node **in** neg\_neighbs

**if** other\_good\_node.edge\_weight = min\_neg **then**

                        left\_node ← other\_good\_node

**endif**

**endfor**

**if** right\_node **then**

                    new\_neighbs.append(right\_node)

                endif

**if** left\_node **then**

                    new\_neighbs.append(left\_node)

**endif**

**endfor**

**endfor**

            #giving the node new neighbours

            node.neighbours ← new\_neighbs

**endsubroutine**

### Cycle detection algorithm

Though seemingly simple, without this my solution is amiss. This function is based on a traditional depth first search, however a traditional DFS is limited to only determining whether a cycle exists or not in a graph. I build on it by finding all the cycles and returning the nodes that form all the cycles. I implement it recursively instead of with a stack to save time, energy and brainpower.

subroutine find\_cycles(self,Node, List)

    Cycles ← [] #will contain the cycles

**if** len(List) > len(self.Nodes) then

**return**   #if the length of the list is greater than the number of nodes no cycle was found

**endif**

**for** neighbour **in** Node.neighbours #for each neighbour

        new\_neighb ← self.get\_node((neighbour.x,neighbour.y)) #get the node on the graph

        new\_neighb.coming\_from.append(Node)

**if** new\_neighb **in** Node.coming\_from and new\_neighb **in** List **then** #if new\_neighbour is the node we are coming #from go to another node

**continue**

**endif**

**if** new\_neighb **in** List **then**:

       #if the neighbour is already in the list then there is a cycle found

            index ← List.position(new\_neighb)

            cycle ← List[index:]  #cycle will be from the first occurrence of node to the end

            Cycles.append(cycle) #put the cycle in a list

**return** Cycles

**endif**

        #if node being visited isnt the one that isnt the one im coming from

        #and if it hasnt already been visited

        new\_list ← List[:]

        new\_list.append(new\_neighb) #add it to a list

        cycle  ← self.find\_cycles(new\_neighb,new\_list) #find\_cycles on node being visited

        Cycles.extend(cycle)

**endfor**

**return** Cycles

**endsubroutine**

### Inequalities

The nodes that form cycles are gotten with the cycle detection algorithm. I will filter for the cycle with maximum number of nodes. I use the **Statistics** module to find the mean of the coordinates of these nodes. I then compare the y values of the lines for the same x and then use this to generate inequailities.



Mean of nodes will be somewhere there. I can compare the lines with this point to generate inequalities

Nodes that form cycle with maximum number of nodes is [C, D, F, I, G]

## Objective function (See objective line)

This will be a randomly generated line with the subroutine **get\_lineLP,** but with additional functionality. Inheriting both from **Line2dcop** which inherits from both a class from matplotlib.lines **Line2d**  and from **Lines**

**class** DraggableLine(Line2dcop)

**subroutine \_\_init\_\_**(self,line2d,line, optimum\_sol)

        self.optimum\_sol ← optimum\_sol

        Line2dcop.\_\_init\_\_(self,line2d,line) #self.Line2D = line2d, self.line = line

        self.press ← None

**endSubroutine**

**subroutine** connect(self)

        #connect to all events needed

        self.cidpress ← self.line2d.figure.canvas.mpl\_connect("button\_press\_event", self.on\_press)

        self.cidrelease ← self.line2d.figure.canvas.mpl\_connect("button\_release\_event", self.on\_release)

        self.cidmotion ← self.line2d.figure.canvas.mpl\_connect("motion\_notify\_event", self.on\_motion)

**endsubroutine**

**subroutine** on\_press(self,event)

        #Check whether the mouse is over line;if so store some data.

**if** event.inaxes ≠ self.line2d.axes **then**

**return**

**endif**

        contains, attrd ← self.line2d.contains(event) #checks to see if event happened in line

**if** not contains **then**

**return**

**endif**

        self.press ←  (event.xdata, event.ydata) #where line was clicked

**endsubroutine**

**subroutine** on\_motion(self,event)

        #Move the line if the mouse is over line

**if** self.press is None or event.inaxes ≠ self.line2d.axes then

**return** # if no event occurs in line or wrong axes return

**endif**

        (xpress, ypress) ← self.press

        dx ← event.xdata - xpress #amount to shift line by in x direction

        dy ← event.ydata - ypress #amount to shift line by in y direction

        #get line data and add or subtract from the line depending on dx and dy

        xdata ← self.line2d.get\_xdata()

        ydata ← self.line2d.get\_ydata()

        self.line2d.set\_xdata(xdata + dx)

        self.line2d.set\_ydata(ydata + dy)

        self.line2d.figure.canvas.draw()

        self.press ← event.xdata, event.ydata #new press point to keep dx and dy constant

        #we want to stop when we hit  optimum solution

        #line is essentially changing while being shifted

        #in short extrapolate new line made and see if the optimum solution will be on it if so disconnect

        place\_to\_stop ← self.optimum\_sol[1]-np.interp(self.optimum\_sol[0], self.line2d.get\_xdata(),self.line2d.get\_ydata())

**if** -1< place\_to\_stop <1 **then** #making solution more robust

            self.disconnect()

**endif**

**endsubroutine**

**subroutine** on\_release(self, event)

        #Clear button press information

        self.press ← None

        self.line2d.figure.canvas.draw()

**endsubroutine**

**subroutine** disconnect(self)

        #Disconnect all callbacks

        self.line2d.figure.canvas.mpl\_disconnect(self.cidpress)

        self.line2d.figure.canvas.mpl\_disconnect(self.cidrelease)

        self.line2d.figure.canvas.mpl\_disconnect(self.cidmotion)

**endsubroutine**

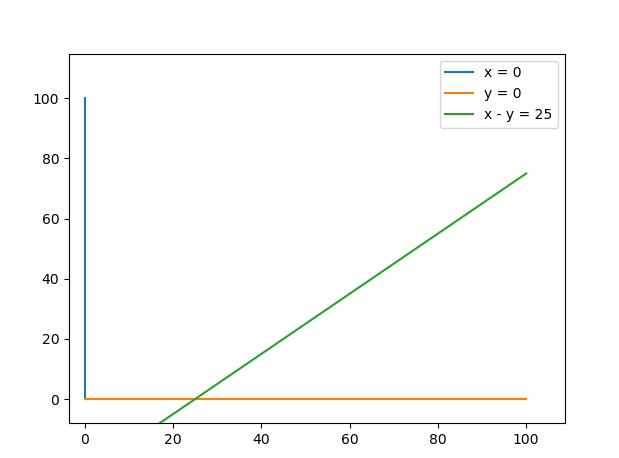
**endclass**

### Questions asked

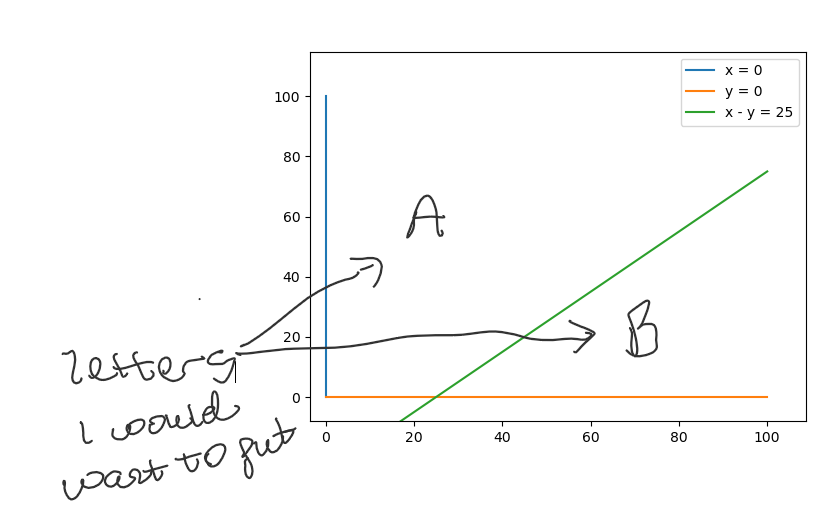
Throughout the tool questions will either be fill in the blanks, multiple choice or questions which user needs to click on the graph to answer eg user can be asked to click on the part of the graph that is to be shaded (click type).

**Area to shade questions**

For click type question all i will need to do is get the coordinate the user clicked and verify it satisfies an inequality as such i wont focus on its design. Here i focus on the design of fill in the blanks. The design will be similar for MCQ



Say i had a graph such as the above, i would want to be able to put letters



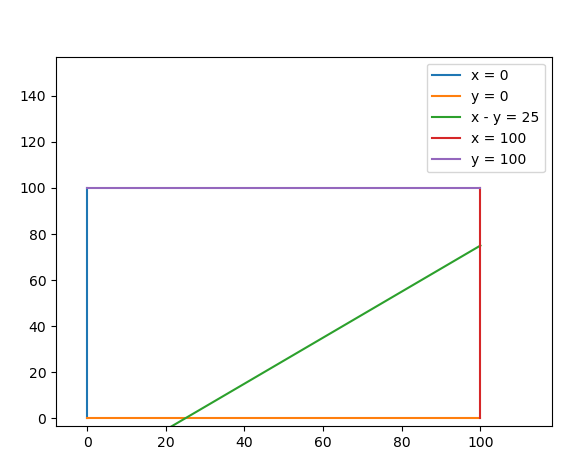
on alternating sides of the graph. One of the letters will be the answer to the question.

One point should satisfy the inequality plotted and the other should not.

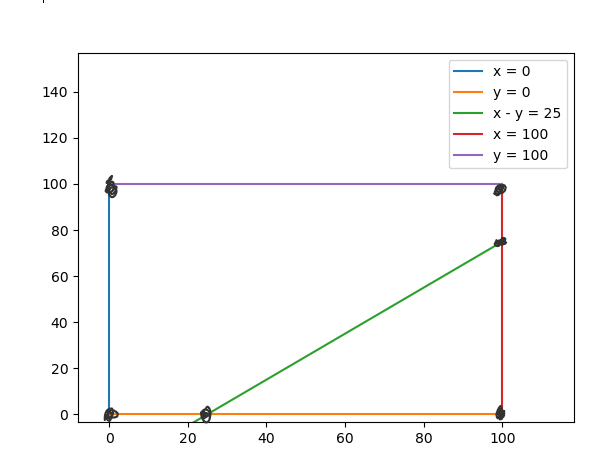
An easy solution will be to generate random points and keep testing until one generated satisfies it and another doesn’t. Theoretically speaking a computer should have no trouble doing this. However, there is still the possibility that a user is stuck waiting for the computer to generate a point. Also point generated can be right on the edge of the line or close enough to the line that text might be covering other parts of the graph.

To try and centralize text and eliminate the possibility of the user waiting I designed another solution.

As per the design of my graphs, they have a finite domain with the last element being the maximum value on the x axis. Hence lines are not plotted outside this maximum values. On the above graph for example the maximum x value is 100 as such the lines seem to stop there.

Knowing the maximum values x and y can take i introduce lines at this boundaries. 

Modelling the graph with nodes:



black circles are nodes

If a cycle detection algorithm is run for any possible line, 3 cycles should be found;1 being the cycle that runs round the square and the other two the alternate sides of the line. I can get the nodes that form the other two cycles and average them out to find the central point in that cycle. I can then confirm which of the points satisfies the inequality and the one that doesn’t, allowing me to appropriately place letters which the user can choose from.

An entry box can then be placed near a question and user will have to put in the right letter to get the answer correct. MCQ questions will have options with the answers instead.

Upon further testing there will be some lines that won’t explicitly work with these lines. I however make sure this is not a problem as with these lines ill make the only option for “area to shade questions”, “click type” ie user will have to click on the area to shade.

### Optimizing a function subject to constraints

Initially i researched libraries that could do this for me. However, they were all complex with certain restrictions. I then decided to make an algorithm myself. Researching algorithms to use to solve LP problems, I found the simplex algorithm. The only problem was with my level of skill it would have been to difficult to code. Moreover, it was like blowing out a candle with a wind turbine as AS linear programming is restricted to 2 variables.

I finally made an algorithm drawing on inspiration from what we had been taught in further maths to solve it.

My plan will be to find all possible intersections of generated lines.

I will use a recursive function to find all possible combinations of all lines. This method proved most effective when compared to similar functionality of modules. Also, given how few the number of lines will be, it wont burden any processor. Pseudocode below:

**Subroutine** combinations(iterable):

**if** len(iterable) = 0 **then**

**return**[[]]

**endif**

combos ← []

**for** combo in combinations(iterable[1:])

combos ← combos + [combo, combo + [iterable[0]]]

**endfor**

**return** combos

**endSubroutine**

The combinations of lines are then filtered to those with two elements modelling the intersection of two lines. Intersections that don't satisfy any of the inequalities are removed. The objective function is then evaluated at the remaining intersections and the optimal solution is found. I use numpy to solve the lines simultaneously and find the intersections. The resulting x and y solution will be the x and y coordinate of a Node

#**function** to solve two simultaneous equations

**Subroutine simultaneous**(line1,line2):

    a ← np.array([line1.x,line1.y])

    b ← np.arra**y**([line2.x,line2.y])

    lhs ← np.array([a,b])

    rhs ← np.array([line1.rhs, line2.rhs])

**try**

        solution ← np.linalg.solve(lhs,rhs)

    #**for** lines that cant be solved simultaneously eg x = 10 **and** x = 80

**except** np.linalg.LinAlgError

**return** None

**else**

**return** solution # (xcoordinate,ycoordinate) of vertex

**endSubroutine**

**subroutine max\_obj**(lines, minimax, lenaxis = None, objective= None)

#Function that either get nodes **for** graph **or** maximises **or** minimises an objective **function**

#lines - lines or inequalities to **use**

#minimax - mode ("high", "low", "get\_nodes")

#lenaxis - length of axis (xlength,ylength)

#objective – **function** to be maximised

rows ← combinations(lines)

temp\_rows ← []

#filtering **for** those with only two lines

**for** row in rows

**if** len(row) = 2 **then**

temp\_rows.append(row)

**endif**

**endfor**

#putting solutions of all possible two lines into a list

solutions ← []

new\_rows ← []

#removing combinations of lines that dont give solutions

**for** row **in** temp\_rows

ans ← simultaneous(row[0], row[1])

#adding solutions of lines to solutions

**if** type(ans) == np.ndarray **then**

new\_rows.append(row)

solutions.append(ans)

**endif**

**endfor**

rows ← new\_rows

#putting corresponding values of objective using solutions into a list

tests ← []

#defualt objective is none **as** im more often using this **function** to #get the nodes

**if** objective **then**

**for** obj in solutions

sol ← obj[0] \* objective.x + obj[1] \* objective.y

tests.append(sol)

**endfor**

**endif**

#going to **use** this **function** to also get the nodes from a graph

#removing solutions not in axis from list dont want the user to be shifting the graph to find regions

**if** minimax = "get\_nodes" **then**

filtered\_rows ← [] #lines of intersections i need

filtered\_list ← [] #solutions ill **usefor**nodes

gotten\_nodes ← [] #stores the nodes

**for** i,solution **in** enumerate(solutions)

**if** solution[0] <= lenaxis[0] **and** solution[1]<= lenaxis[1] **and** solution[0] >= 0 **and** solution[1] >= 0 **then**

filtered\_list.append(solution) #getting only nodes in range

filtered\_rows.append(rows[i]) #getting lines that make nodes

**endif**

**endfor**

**for** node, row **in** zip(filtered\_list,filtered\_rows)

gotten\_nodes.append(Node(node,row)) #creating nodes based on #solution **and**lines that make node

**endfor**

**return** gotten\_nodes

#**for** each sol:

#**for** each line: **if** it doesnt satisfy the inequality remove solution from solutions(removing answers that are not #in the feasible region)

**elseif** minimax = "high" **th1en #maximizing an objective function**

**for** i **in** range(len(tests))

**for** line **in** lines

**if** line.status = "<=" **then**

#i use round to make solution more robust as there might be some minor drops in accuracy

**if** round(solutions[i][0] \* line.x + solutions[i][1] \* line.y,10) > line.rhs **then**

tests[i] ← -np.inf

**endif**

**elseif** line.status = ">=" **then**

**if** round(solutions[i][0] \* line.x + solutions[i][1] \* line.y,10) < line.rhs **then**

tests[i] ← -np.inf

**endif**

**endif**

**endfor**

**endfor**

#optimum things

optimum\_index ← tests.position(max(tests))

optimum\_intersection ← rows[optimum\_index]

optimum\_solution ← tests[optimum\_index]

optimum\_x\_and\_y ← solutions[optimum\_index]

**elseif** minimax = "low" **then #minimizing an objective function**

**for** i in range(len(tests))

**for** line **in** lines

**if** line.status = "<=" **then**

**if** round(solutions[i][0] \* line.x + solutions[i][1] \* line.y,) > line.rhs **then**

tests[i] ← np.inf

**endif**

**elseif** line.status = ">=" **then**

**if** round(solutions[i][0] \* line.x + solutions[i][1] \* line.y,7) < line.rhs **then**

tests[i] ← np.inf

**endif**

**endif**

**endfor**

**endfor**

#optimum things

optimum\_index ← tests.position(min(tests))

optimum\_intersection ← rows[optimum\_index]

optimum\_solution ← tests[optimum\_index]

optimum\_x\_and\_y ← solutions[optimum\_index]

**endif**

**return** optimum\_solution,optimum\_intersection,optimum\_x\_and\_y

**endsubroutine**

## Description of data structures

Here I highlight the key data structures I use and their purposes in my solution and if applicable its implementation details.

**Graphs**

Though extensively described already (see Getting Nodes and finding Cycles), Line Graphs will be converted into **Graphs** the abstract data structure in order to run a cycle detection algorithm and ensure inequalities made form a closed cycle.

The graph will have an attribute **nodes** – a list holding the nodes that the graph consists of. Each of the nodes has a **neighbours** attribute – a list of all the nodes neighbours. Therefore the graph is represented essentially using an adjacency matrix. This representation will be suitable as graphs will generally be sparse.

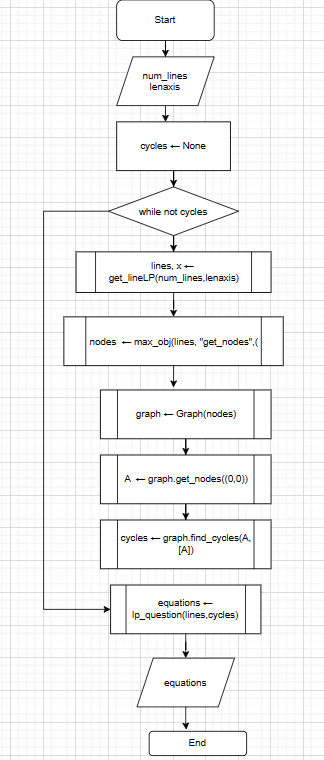
**Sets**

Sets as part of how the graph of lines is converted into the graph (abstract data structure), I turn each of the nodes’ lines attributes (lines being a list of the lines that intersects to form the node) to sets and then check if there is an intersection between it and the set of any other node. If there is, these two nodes that have an intersection between lines attributes could potentially be neighbours. I implement the checking of shared line attributes this way as the selecting the two nodes to check for intersections already has a Big O of (n2). Comparing individual values of lines in between the nodes’ lines’ attribute could potentially make the Big O of (n4)or worse.

**Dictionaries**

When pages are instantiated, the page name is held as a key in a dictionary while the page object is the value of the key. This allows for easy removal, replacement, addition and access of page objects.

## High level overview of linear programming question generation



In actuality i will implement this in a class **LP\_question (see question)** with an attribute of the class being equations.

## Design of user interface

I use customtkinter to design the GUI.

**All pages will occupy at least 3/4th of the horizontal width of any screen.** I do this as I will embedding graphs in so. All elements can automatically scale as such enlarging the screen doesnt have any negative effects. For best display it will be best the solution is on full screen.

**The background color and foreground colour of widgets is consistent throughout *#EFEFE0*** this color is light on the eyes and will allow for good contrast with the various color lines that will be plotted.

**The text color is also consistent throughout and is, *#7A513A***providing good contrast with the background and aiding easy visibility. The text font is “Arial” with varying sizes depending on purpose of widget.

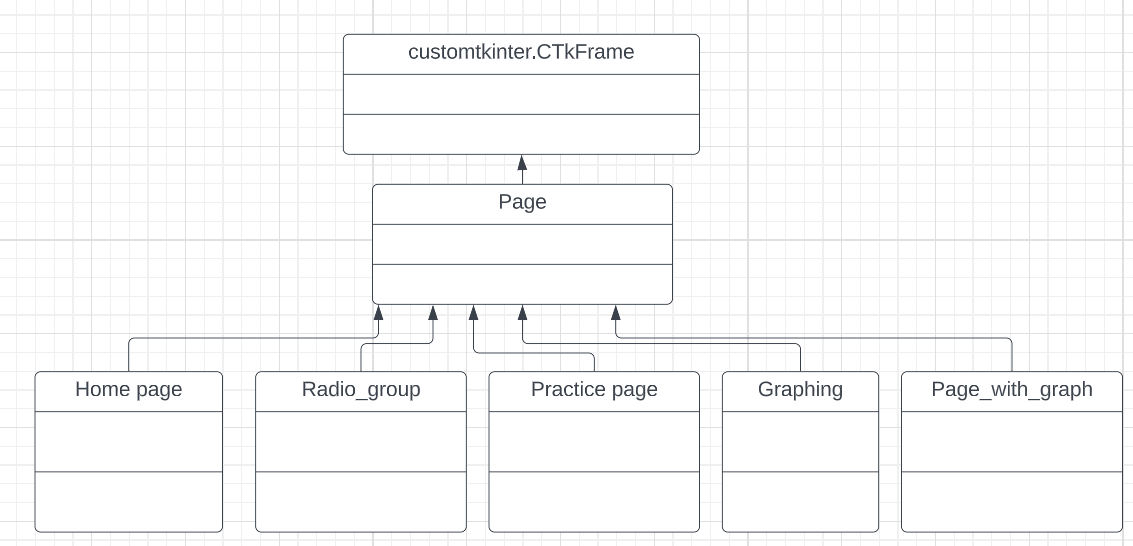
**The border color around all buttons is white and the hover color for all buttons is “*grey92”***

Pages in my solution will be classes inheriting from a Page class inheriting from Customtkinter frames.

**Page** will be an abstract class, never actually instantiated but It will be the parent class of the various pages used throughout my solution. I use inheritance to my advantage allowing for attributes of my pages to be consistent throughout without repeating code

Page objects (ie objects of the child classes of **Page**) will then be instantiated and placed on the root window.

I do the same with buttons and labels ensuring easy formatting of all widgets easily and eliminating the need to repeat a lot of code



**Home page**

This will be the page that will be seen immediately the tool is opened

Graphical user interface, application, PowerPoint

Description automatically generated

Border Colour: white

Font Colour: #7A513A

Background Colour :#EFEFE0

**Pages with graphs**

Pages with graphs will have the graph on the left and tools to aid with viewing the graph on the bottom.

Graph on left with its background color being white to provide good contrast for various color lines that will be plotted

Chart

Description automatically generated

Questions of varying types asked.

Tools to aid with viewing graph if needed.

Questions of varying types asked.

### Line Drawing

**2d array , connect four**

**Design of Graphing a LP problem page question switching (refer to linear programming gui)**

I pick on the deign of this page as it shows how most of the question displaying will be done. This page will be similar to other pages.

The pseudocode and design here will be highly simplified focusing majorly on the interactions with the user and how question switching will be done. Widget placement is entirely left out as this is done to show how interactions occur.

(I will edit this for clarity in future)

from questions import LPquestion

**class** Graphing(Page)

    #This page handles Graphing Linear Problems

    subroutine(self,master)

        super().\_\_init\_\_(master)

        self.question ← LPquestion(3,100) #LP question object. main **use** will be its equations attribute

       #title

        title ← def\_lbl(self,text ← "Graphing a LP problem",font ← ("Arial",40))

        #opening text displaying all the inequalities

        opening\_text ← "Lets Graph these inequalities:\n"

**for** equation in self.question.equations:

            opening\_text ← opening\_text+ "\n" + equation.\_\_str\_\_()

**endfor**

        #label with opening text

        large\_lbl ← def\_lbl(self,text ← opening\_text,font ← ("Arial",25))

        #button to start graphing the ineqaulities

        go\_btn ← def\_button(self,text ←"Lets GOOO",font ← ("Arial",25), command ← self.change\_in())

        self.current ← -1 #initialised to -1 so intial call will be 0 **and** so on

        self.max\_current ← len(self.question.equations)- 1 #max number of equations

**endsubroutine**

##when go button is clicked change\_question **function** is run"""

**subroutine** change\_question(self):

        #subroutine changing inequalities when done graphing one

        self.clear() #clear frame

        self.current ← self.current + 1

        #current equation

        ##load **new** Page that has graph **and** put it on

        new\_page ← Page\_with\_Graph(self,self.question.equations[self.current],100)

**endsubroutine**

**endclass**

##**for** each inequlty a Page\_with\_Graph object is generated **and** put on the screen"""

**class** Page\_with\_Graph(Page)

  #on this page user is tested on yintercept, xintercept **and** the area to be shaded on graph of

  #inequalities

**subroutine**(self,master,equation,end\_x):

        ##all widgets here shouldnt change **for** current equation

        super().\_\_init\_\_(master,)

        #values intialised to 0

        self.yintercept ← 0

        self.xintercept ← 0

        self.areashade ← 0

        self.equation ← equation

        part1 ← ("y-intercept","x-intercept","area to shade")

        self.generator ← (element **for** element in  part1) #generator object - can invoke next() to bring up next part of question

        #when a user has gotten a part correct we get a next part of equation to **do**

        #using the next element from the generator

        #title at top

        title ← def\_lbl(self,text ← "Graphing a LP problem",font ← ("Arial",40))

        #equation currently being worked on

        self.main\_label ← def\_lbl(self,text ← "Graphing " + equation.\_\_str\_\_(),font ← ("Arial",30,"underline"))

        self.x ← np.arange(0,end\_x+1) #end of x-axis

        self.graph\_frame ← ctk.CTkFrame(self,fg\_color ←"#EFEFE0") #frame that will hold graph

        self.fig, self.axes ← empty\_plot(self.x)

        self.axes.grid()

##graph that will be used is gotten **and** is placed on the left side of the screen"""

        #canvas on which graph will be drawn

        self.graph\_canvas ← FigureCanvasTkAgg(self.fig,self.graph\_frame)

        #navigation toolbar

        self.toolbar ← NavigationToolbar2Tk(self.graph\_canvas,self.graph\_frame,pack\_toolbar←False)

        #this frame will hold questions user will be asked

        self.question\_frame ← Page(self)

        #contents of this frame changes based on question part

        self.refresh()

**endsubroutine**

##I put a circle at the x **and** y intercept when done by user **and** shade the area to be shaded when done

##by user.Plot is a subroutine that plots the line on the graph (see func\_plot1) **for** the x **and** y intercept

##questions can either be multiple choice **or** fill in the blanks. **for** area to shade user can also click on the area

##to shade denoted by touch"""

**subroutine** refresh(self)

        #putting dots at x **and** y intercept when done by user

**if** self.yintercept **and** self.equation.c then #**if** the line has a yintercept

            c1 ← plt.Circle([0,self.equation.c],radius ← 1,picker ← True, color ← "black")

            self.axes.add\_artist(c1)

            self.graph\_canvas.draw()

**endif**

**if** self.xintercept **and** self.equation.xc then

            c1 ← plt.Circle([self.equation.xc,0],radius ← 1,picker ← True, color ← "black")

            self.axes.add\_artist(c1)

            plot(self.equation,self.x,self.axes,self.graph\_canvas)

**endif**

        self.question\_frame.clear() #clear the question frame **new** question will be put in here

        question\_types ← ["fill\_blank","MCQ",]

        question\_types1 ← ["fill\_blank","MCQ","touch"]

##If there is no more element in the generator object then user is done with all parts of graphing the equation

##so move on to next equation. Access\_mode being either the x intercept, yintercept **or** area to shade"""

**try**:

            #access mode of the question would be the next element from the generator

            self.access\_mode ← next(self.generator)

            #including touch type questions **for** area to shade questions

**if** self.access\_mode ≠ "area to shade" then

                self.question\_type ← random.choice(question\_types)

**else**

                self.question\_type ← random.choice(question\_types1)

**endif**

**if** self.question\_type = "fill\_blank" then

##Graphing\_question.get\_question() returns a question object **and** its question stem (see questions)"""

                self.question\_obj,self.question\_text, ← Graphing\_question(self.access\_mode,nec\_info← self.equation).get\_question(self.question\_type) #question object,question prompt

                self.entry ← ctk.CTkEntry(self.question\_frame) #entry box **for** fill blank questions

##fill blank have two different styles i cater to both"""

**if** self.question\_obj.style = 1 then # \_\_\_\_ is the blah blahblah - entry before

                    #question stem

                    self.question\_lbl ← def\_lbl(self.question\_frame,text ← self.question\_text)

**elseif** self.question\_obj.style = 0 then # what is the blah blah blah --entry is after

                    self.question\_lbl ← def\_lbl(self.question\_frame,text ← self.question\_text)

**endif**

**elseif** self.question\_type = "MCQ"

##MCQ questions have an extra attribute containing the options to be picked from"""

                self.question\_obj,self.question\_text,self.options ← Graphing\_question(self.access\_mode, self.equation).get\_question(self.question\_type) #question object,question prompt, options

                #question stem

                self.question\_lbl ← def\_lbl(self.question\_frame,text = self.question\_text)

##**class** that makes radiobuttons sharing same variable with options"""

                self.radio\_group ← Radio\_group(self.question\_frame,len(self.options), self.options) #radio buttons

**elseif** self.question\_type = "touch" then

                #user has to click on graph to answer this type of question

                self.question\_obj,self.question\_text ← Graphing\_question(self.access\_mode, self.equation).get\_question(self.question\_type) #question object,question prompt, options

                self.question\_lbl ← def\_lbl(self.question\_frame,text ← self.question\_text)

##connecting graph to mouse press events """

                self.cidpress ← self.graph\_canvas.mpl\_connect("button\_press\_event", self.submit)

             #submit button , will be configured to show next part **if** answer is correct **and**  next equation

             #**if** equation is done

            self.next\_button ← def\_button(self,text ← "Submit",command ← self.submit)

**endif**

        except StopIteration #then equation is done

            #all elements in generator have been used up ==> we are done with all parts of equation

##**function** to plot inequality see func\_plot1"""

            plot\_linobj([self.equation], self.x, self.axes)

            self.graph\_canvas.draw()

            #configure the button to move on to next equation

            self.next\_button.configure(command← self.master.change\_question,text ← "Next Equation")

**endsubroutine**

**subroutine** submit(self,event ← None)

        """Validating user entry regardless of question type"""

        # **if** dealing with x intercept **or** y intercept part is 1

**try**:

**if** self.question\_type = "fill\_blank" then

                self.user\_entry ← self.entry.get() #get user input

**if** self.question\_obj.part = 1 then #we are dealing with x **or** y intercept

##Validating lines that have gradient = infinity **or** 0 is different from other lines"""

**if** self.equation.m ≠ 0 **and** self.equation.m ≠ np.inf then

**if** STRING\_TO\_REAL(self.user\_entry) = self.question\_obj.answer then #compare float with answer

                            print("CorrecT")

                            #set part of question done to 1

**if** self.access\_mode = "y-intercept" then

                                self.yintercept ← 1

**elseif** self.access\_mode = "x-intercept" then

                                 self.xintercept ← 1

**endif**

                            self.next\_button.configure(command ← self.refresh,text ← "Next Part") #go to next part **if** correct

**else**:

                            print("Wrong")

**endif**

**elseif** self.equation.m = np.inf then

**if** self.access\_mode  = "y-intercept" then

**if** self.user\_entry = "NA" then

                                print("CorrecT")

                                self.next\_button.configure(command ← self.refresh,text ← "Next Part") #go to next part **if** correct

**else**:

                                print("Wrong")

**endif**

**elseif** self.access\_mode = "x-intercept" then

**if** STRING\_TO\_REAL(self.user\_entry) = self.question\_obj.answer then #compare float with answer

                                print("CorrecT")

                                self.next\_button.configure(command ← self.refresh,text ← "Next Part") #go to next part **if** correct

**else**:

                                print("Wrong")

**endif**

**endif**

**elseif** self.equation.m = 0

**if** self.access\_mode = "x-intercept" then

**if** self.user\_entry = "NA" then

                                print("Correct")

                                self.next\_button.configure(command ← self.refresh,text ← "Next Part") #go to next part **if** correct

**else**:

                                print("Wrong")

**endif**

**elseif** self.access\_mode ="y-intercept" then

**if** float(self.user\_entry) = self.question\_obj.answer then #compare float with answer

                                print("CorrecT")

                                self.next\_button.configure(command ← self.refresh,text ← "Next Part") #go to next part **if** correct

**else**:

                                print("Wrong")

**endif**

**endif**

**endif**

**elseif** self.question\_obj.part = 2 then

**if** self.user\_entry = self.question\_obj.answer then

                        print("CorrecT")

                        self.areashade ← 1

                        self.next\_button.configure(command ← self.refresh,text ← "Next Part") #go to next part **if** correct

**else**:

                        print("Wrong")

**endif**

**elseif** self.question\_type = "MCQ" then

                self.user\_entry ← self.radio\_group.variable.get() #get option chosen by user

**if** self.user\_entry = self.question\_obj.answer then

                    print("CorrecT")

**if** self.access\_mode = "y-intercept" then

                        self.yintercept ← 1

**elseif** self.access\_mode = "x-intercept" then

                         self.xintercept ← 1

**elseif** self.access\_mode = "area to shade" then

                        self.areashade ← 1

**endif**

                    self.next\_button.configure(command ← self.refresh,text ← "Next Part")# go to next part **if** correct

**else**:

                    print("Wrong")

**endif**

**elseif** event.xdata **and** event.ydata then #means graph was clicked- an area to shade question

##**function** to check **if** user clicked on right area to shade of graph see func\_plot1

                test  ← validate(self.equation,event)

**if** test = "CorrecT" then

                    print("CorrecT")

                    #done with question disconnect callbacks

                    self.graph\_canvas.mpl\_disconnect(self.cidpress)

                    self.next\_button.configure(command ← self.refresh,text ← "Next Part") # go to next part **if** correct

**else**:

                    print("Wrong")

**endif**

**endif**

**except** ValueError

            print("Invalid output")

**endsubroutine**

**endclass**

## System security and integrity of data

As of now, no user data Is stored by the tool. As such system security is uneccessary

# Implementation

Below are the modules containing the implementation of my solution.

## lines

This module contains the classes of the lines that will be instantiated throughout the program except the objective lines. I exclude the Draggable lines from this module as it will cause circular imports. However, the Draggable lines still inherit from the classes in this module.

**import** **numpy** **as** **np**

**from** **matplotlib.lines** **import** Line2D

*#pretty self explanatory*

*#x, y , right hand side and status attribute*

**class** **lines**(Line2D):

valid\_statuses = ["=","<=",">="]

**def** \_\_init\_\_(self,x=0,y=0,rhs=0,status = "=", test = None):

self.x = x

self.y = y

self.rhs = rhs

self.status = status

self.test = test

*# going to use these to work around matplotlib fill\_between*

*#need the value of gradient...*

**if** self.rhs == "O": *#also need the value of gradient of objective function to ensure it isnt parallel to other inequalities*

**if** self.y == 0:

self.m = np.inf

**else**:

self.m = -self.x/self.y

**else**:

**if** self.y == 0:

self.m = np.inf

self.c = None

**else**:

self.m = -self.x/self.y

self.c = self.rhs/self.y

**if** self.x != 0:

self.xc = float(f"{self.rhs/self.x:.3f}")

**else**:

self.xc = None

**def** output(self,x):

*#will mainly be used to generate inequalities by comparing mean value of nodes with value of line at that point*

*#but also see what a lines y will be for a given x*

**if** self.y != 0:

y = (self.rhs - self.x \* x)/self.y

**return** y

**else**:

**return** "NA"

*#function to print line im working with*

*#takes into account coefficent of x and y so i dont have 1x +... or x + -4y.used logic gates making it way easier to follow*

*#also takes away 0 so dont have 0x + or 0y...*

**def** \_\_str\_\_(self):

y = str(self.y)

x = str(self.x)

*#taking care of 0*

**if** x == "0" **and** y == "0":

**return** ("Invalid line")

*###*

**elif** x == "0" **and** y != "0":

*###*

**if** y.strip("-") != "1":

**if** "-" **not** **in** y:

**return** f"{y}y {self.status} {self.rhs}"

**elif** "-" **in** y:

**return** f"-{y.strip('-')}y {self.status} {self.rhs}"

*####*

**elif** y.strip("-") == "1":

**if** "-" **not** **in** y:

**return** f"{y.strip('1')}y {self.status} {self.rhs}"

**elif** "-" **in** y:

**return** f"-{y.strip('-1')}y {self.status}{self.rhs}"

**elif** y.strip("-") != "1":

**if** "-" **not** **in** y:

**return** f"{y}y {self.status} {self.rhs}"

**elif** "-" **in** y:

**return** f"-{y.strip('-')}y {self.status} {self.rhs}"

**elif** y.strip("-") == "1":

**if** "-" **not** **in** y:

**return** f"{y.strip('1')}y {self.status} {self.rhs}"

**elif** "-" **in** y:

**return** f"-{y.strip('-1')}y {self.status} {self.rhs}"

*###*

**elif** x != "0" **and** y == "0":

**if** x.strip("-") != "1" :

**return** f"{x}x {self.status} {self.rhs}"

*####*

**elif** x.strip("-") != "1":

**return** f"{x}x {self.status} {self.rhs}"

**elif** x.strip("-") == "1":

**return** f"{x.strip('1')}x {self.status} {self.rhs}"

**elif** x.strip("-") == "1" :

**return** f"{x.strip('1')}x {self.status} {self.rhs}"

*####*

**elif** x != "0" **and** y != "0":

*###taking care of 1's and -*

**if** x.strip("-") != "1" **and** y.strip("-") != "1": *#0,0*

**if** "-" **not** **in** y:

**return** f"{x}x + {y}y {self.status} {self.rhs}"

**elif** "-" **in** y:

**return** f"{x}x - {y.strip('-')}y {self.status} {self.rhs}"

*####*

**elif** x.strip("-") != "1" **and** y.strip("-") == "1": *#0,1*

**if** "-" **not** **in** y:

**return** f"{x}x + {y.strip('1')}y {self.status} {self.rhs}"

**elif** "-" **in** y:

**return** f"{x}x - {y.strip('-1')}y {self.status} {self.rhs}"

**elif** x.strip("-") == "1" **and** y.strip("-") != "1": *#1,0*

**if** "-" **not** **in** y:

**return** f"{x.strip('1')}x + {y}y {self.status} {self.rhs}"

**elif** "-" **in** y:

**return** f"{x.strip('1')}x - {y.strip('-')}y {self.status } {self.rhs}"

**elif** x.strip("-") == "1" **and** y.strip("-") == "1": *#1,1*

**if** "-" **not** **in** y:

**return** f"{x.strip('1')}x + {y.strip('1')}y {self.status} {self.rhs}"

**elif** "-" **in** y:

**return** f"{x.strip('1')}x - {y.strip('-1')}y {self.status} {self.rhs}"

**def** \_\_repr\_\_(self):

**return** self.\_\_str\_\_()

*#if both are 0 print invalid line*

*#major logic is if either the y or x attribute is 0, dont print the variable that is 0*

*#as for the variables that arent 0, check if the coefficient is 1 if it is 1, remove the 1 and print only the variable*

*#for y variables also check if it is negative and deal with it*

*#finally if neither 0,1,or negative just print it how it is*

**class** **Line2dcop**(lines,Line2D):

*"""Class inheriting from both matplotlib lines and my own lines"""*

**def** \_\_init\_\_(self,Line2d,line):

Line2D.\_\_init\_\_(self,Line2d.get\_xdata(),Line2d.get\_ydata())

lines.\_\_init\_\_(self,line.x,line.y,line.rhs)

self.line2d = Line2d

self.line = line

## lines\_for\_lp

This module will be used for the generation of the lines that will later be turned to inequalities.

*###module to generate lines for linear programming*

**from** **lines** **import** lines **as** l

**import** **numpy** **as** **np**

**import** **random**

*#first generate points*

**def** generate\_pointsline(x,number):

*"""generate points for lines based on x axis*

*A pair of points per number"""*

points = []

**for** \_ **in** range(number):

point\_oney = random.randrange(0,np.amax(x))

point\_onex = random.randrange(0,np.amax(x))

point\_twoy = random.randrange(0,np.amax(x))

point\_twox = random.randrange(0,np.amax(x))

coordinates = [(point\_onex,point\_oney),(point\_twox,point\_twoy)]

points.append(coordinates)

**return** points

*#using two points to draw a line*

**def** drawtwopoints(point\_one, point\_two):

*"""uses two points to draw a line"""*

l\_onex, l\_oney = point\_one

l\_twox, l\_twoy = point\_two

*#y = mx + c*

**try**:

m = (l\_twoy-l\_oney)/(l\_twox-l\_onex)

c = l\_twoy - m\*l\_twox

line = l(round(-m,2),1,round(c,2))

**return** line

**except** **ZeroDivisionError**:*#gradient of line is infinity line of form x = k*

m = 1 *#coefficient of x is 1*

c = l\_twox *#rhs is going to be x value*

line = l(round(m,2),0,round(c,2))

**return** line

*#main function generating and returning lines for linear programming*

**def** get\_lineLP(num\_lines,end\_x,objective = None):

*"""returns the lines to be plotted"""*

x = np.arange(0,end\_x+1,10)

**if** num\_lines > 1: *#then i need linear programming lines include y and x = 0*

points = generate\_pointsline(x,num\_lines)

lines = [] *#array of lines*

**for** point **in** points:

line = drawtwopoints(point[0],point[1])

lines.append(line)

lines.extend([l(1,0,0),l(0,1,0)]) *#adding x and y = 0 to lines*

**return** lines ,x

**elif** num\_lines ==1:

*#one objective line*

point = generate\_pointsline(x,num\_lines)[0]

line = drawtwopoints(point[0],point[1])

intercept\_test = line.rhs

**if** **not** objective:

**return** line

**if** objective:

*#objective function must not be x = blah or y = blah*

**while** line.m == np.inf **or** line.m == 0:

line = drawtwopoints(point[0], point[1])

intercept\_test = line.rhs

line = l(line.x,line.y,"O",test = intercept\_test)

**return** line

## linprog

This module will contain the optimizing algorithm. As there is a lot in common between the process of wrangling the data ill use to convert graphs and the preliminary stages of the optimizing algorithm, I also include the retrieval of nodes in this module.

**import** **numpy** **as** **np**

**from** **graph\_nodes** **import** Node

*#function to solve two simultaneous equations*

**def** simultaneous(line1,line2):

a = np.array([line1.x,line1.y])

b = np.array([line2.x,line2.y])

lhs = np.array([a,b])

rhs = np.array([line1.rhs, line2.rhs])

**try**:

solution = np.linalg.solve(lhs,rhs)

*#for lines that cant be solved simultaneously eg x = 10 and x = 80*

**except** np.linalg.LinAlgError:

**return** None

**else**:

**return** solution

*# recursive algorithm to bring total number of possible combinations from a list*

**def** combinations(iterable):

*"""recursive algorithm to bring total number of combinations from an iterable"""*

**if** len(iterable) == 0:

**return**[[]]

combos = []

**for** combo **in** combinations(iterable[1:]):

combos += [combo, combo + [iterable[0]]]

**return** combos

*#receives an objective function and solutions from lines and returns solutions put in function*

**def** max\_obj(lines, minimax, lenaxis = None, objective= None):

*"""Function that either get nodes for graph or maximises or minimises an objective function*

*lines - lines to use*

*minimax - mode ("high", "low", "get\_nodes")*

*lenaxis - length of axis (x,y)*

*objective - function to be maximised*

*"""*

rows = combinations(lines)

*#filtering for those with only two lines*

rows = list(filter(**lambda** a: len(a) == 2,rows))

solutions = []

new\_rows = []

*#removing combinations of lines that dont give solutions*

**for** row **in** rows:

ans = simultaneous(row[0], row[1])

*#adding solutions of lines to solutions*

**if** type(ans) == np.ndarray:

new\_rows.append(row)

solutions.append(ans)

rows = new\_rows

*#putting corresponding values of objective using solutions into a list*

tests = []

*#defualt objective is none as for now im more often using this function to get the nodes*

**if** objective:

**for** obj **in** solutions:

sol = obj[0] \* objective.x + obj[1] \* objective.y

tests.append(sol)

*#going to use this function to also get the nodes from a graph*

*#removing solutions not in axis from list dont want the user to be shifting the graph to find regions*

**if** minimax == "get\_nodes":

filtered\_rows = [] *#lines of intersections i need*

filtered\_list = [] *#solutions ill use for nodes*

gotten\_nodes = [] *#stores the nodes*

**for** i,solution **in** enumerate(solutions):

**if** (solution[0] <= lenaxis[0]) **and** (solution[1] <= lenaxis[1]) **and** solution[0] >= 0 **and** solution[1] >= 0:

filtered\_list.append(solution) *#getting only nodes in range*

filtered\_rows.append(rows[i]) *#getting lines that make nodes*

**for** node, row **in** zip(filtered\_list,filtered\_rows):

gotten\_nodes.append(Node(node,row)) *#creating nodes based on solution and lines that make node*

**return** gotten\_nodes

*#trying to go for each sol:*

*#for each line: if it doesnt satisfy it remove solution from solutions(removing answers that are not in the feasible region)*

**elif** minimax == "high":

**for** i **in** range(len(tests)):

**for** line **in** lines:

**if** line.status == "<=":

**if** round(solutions[i][0] \* line.x + solutions[i][1] \* line.y,10) > line.rhs:

tests[i] = -np.inf

**elif** line.status == ">=":

**if** round(solutions[i][0] \* line.x + solutions[i][1] \* line.y,10) < line.rhs:

tests[i] = -np.inf

*#optimum things*

optimum\_index = tests.index(max(tests))

optimum\_intersection = rows[optimum\_index]

optimum\_solution = tests[optimum\_index]

optimum\_x\_and\_y = solutions[optimum\_index]

*## round bcuz of an error python gave me*

*## instead of evaluating 25 it did 24.99999999999999996 and that evaluated to false*

*## looked everywhere for the drop in accuracy but couldnt find it python or maybe my laptop just isnt smart enough*

**elif** minimax == "low":

**for** i **in** range(len(tests)):

**for** line **in** lines:

**if** line.status == "<=":

**if** round(solutions[i][0] \* line.x + solutions[i][1] \* line.y,10) > line.rhs:

tests[i] = np.inf

**elif** line.status == ">=":

**if** round(solutions[i][0] \* line.x + solutions[i][1] \* line.y,10) < line.rhs:

tests[i] = np.inf

*#optimum things*

optimum\_index = tests.index(min(tests))

optimum\_intersection = rows[optimum\_index]

optimum\_solution = tests[optimum\_index]

optimum\_x\_and\_y = solutions[optimum\_index]

**return** optimum\_solution,optimum\_intersection,optimum\_x\_and\_y

## graph\_nodes

This module contains most classes and functions related to graphs (the abstract data structure). It also contains the cycle detection algorithm.

**class** **Node**:

**def** \_\_init\_\_(self,coordinate,lines):

*"""*

*Parameters*

*----------*

*coordinate : Tuple*

*Contians x and y coordinate for node to be placed on graph*

*lines : Tuple*

*Contains lines that intersected to get node*

*"""*

self.x = coordinate[0] *#xcoordinate of node*

self.y = coordinate[1] *#ycoordinate*

self.lines = lines *#lines that intersect to form node*

self.neighbours = [] *#list containing neighbouring nodes*

self.coming\_from = [] *#the node could be gotten from several ways*

**def** add\_neighbour(self,neighbour):

*"""add a neighbouring node to a list of neighbouring nodes"""*

**if** neighbour **not** **in** self.neighbours:

self.neighbours.append(node\_neighbour(neighbour,(self.x,self.y)))

**def** \_\_str\_\_(self):

**return** f"Node({self.x},{self.y})"

**def** \_\_repr\_\_(self):

**return** f"{self.\_\_str\_\_()}**\n**"

**class** **node\_neighbour**(Node):

*"""i need some kind of weight function trying to use this to get over it"""*

*#same as a node but this just carries extra weight :)*

**def** \_\_init\_\_(self,node,incoming):

self.x = node.x

self.y = node.y

self.lines = node.lines

*##distance between the nodes is the weight*

*#negative if neighbour is to the left of node or if neighbour is below node (if and only if line they lie on doesnt have changes in x)*

incoming\_x,incoming\_y = incoming

self.edge\_weight = ((incoming\_x-self.x)\*\*2 + (incoming\_y-self.y)\*\*2)\*\*0.5

**if** incoming\_x > self.x:

self.edge\_weight = -self.edge\_weight

*#deals with node that lie on a line x = blah*

**elif** incoming\_x == self.x:

**if** incoming\_y > self.y :

self.edge\_weight = -self.edge\_weight

**def** \_\_str\_\_(self):

**return** f"Neighbour({self.x},{self.y})"

**def** \_\_repr\_\_(self):

**return** f"Neighbour({self.\_\_str\_\_()})**\n**"

**class** **Graph**:

**def** \_\_init\_\_(self,nodes):

self.Nodes = nodes *#nodes that make up graph*

*#add neighbours between all nodes that have an intersection between lines attributes*

**for** node **in** self.Nodes:

**for** vert **in** self.Nodes:

**if** node == vert:

**continue** *#skip comparing yourself*

**else**:

**if** set(node.lines) & set(vert.lines):

node.add\_neighbour(vert)

self.addedges()

**def** addedges(self):

*"""cuts neighbours to only those immediately to the left or right or above or below node"""*

**for** node **in** self.Nodes: *#for each node*

new\_neighbs = [] *#new neighbours without others*

**for** line **in** node.lines: *#for each line the node is a part of*

*#positive weighted for to the left and above and vice-versa*

pos\_weights = []

pos\_neighbs = []

neg\_weights = []

neg\_neighbs = []

**for** vertex **in** node.neighbours: *#for each neighbouring node*

**if** line **in** vertex.lines: *#if i have the intial line*

*#check the weights and determine if weight is positive or negative*

**if** vertex.edge\_weight > 0:

pos\_neighbs.append(vertex)

pos\_weights.append(vertex.edge\_weight)

**elif** vertex.edge\_weight < 0:

neg\_neighbs.append(vertex)

neg\_weights.append(vertex.edge\_weight)

*#node to left or above will be node with minimum positive weight and vice*

min\_pos = min(pos\_weights) **if** pos\_weights **else** None

min\_neg = max(neg\_weights) **if** neg\_weights **else** None

right\_node = None *#restarting them*

left\_node = None

*#find the min\_pos node and min\_neg node and call them right node and left node respectively*

**for** good\_node **in** pos\_neighbs:

**if** good\_node.edge\_weight ==min\_pos:

right\_node = good\_node

**for** other\_good\_node **in** neg\_neighbs:

**if** other\_good\_node.edge\_weight == min\_neg:

left\_node = other\_good\_node

new\_neighbs.append(right\_node) **if** right\_node **else** None

new\_neighbs.append(left\_node) **if** left\_node **else** None

*#giving the node new neighbours*

node.neighbours = new\_neighbs

**def** print\_graph(self):

**for** node **in** self.Nodes:

**print**(f"{node} - {node.neighbours}**\n**")

**def** get\_node(self,coordinate):

*#get the corresponding node to a coordinate*

**for** node **in** self.Nodes:

**if** coordinate == (node.x,node.y):

**return** node

**def** find\_cycles(self,Node, List):

Cycles = [] *#will contain the cycles*

**if** len(List) > len(self.Nodes):

**return** *#if the length of the list is greater than the number of nodes no cycle was found*

**for** neighbour **in** Node.neighbours: *#for each neighbou*

new\_neighb = self.get\_node((neighbour.x,neighbour.y)) *#get the node*

new\_neighb.coming\_from.append(Node)

**if** new\_neighb **in** Node.coming\_from **and** new\_neighb **in** List: *#if new\_neighbour is the node we are coming from go to another node*

**continue**

**if** new\_neighb **in** List:

*#if the neighbour is already in the list then there is a cycle found*

index = List.index(new\_neighb)

cycle = List[index:] *#cycle will be from the first occurence of node to the end*

Cycles.append(cycle) *#put the cycle in a list*

**return** Cycles

*#if node being visited isnt the one that isnt the one im coming from*

*#and if it hasnt already been visited*

new\_list = List[:]

new\_list.append(new\_neighb) *#add it to a list*

cycle = self.find\_cycles(new\_neighb,new\_list) *#find\_cycles on node being visited*

Cycles.extend(cycle)

**return** Cycles

## objective\_line

In this module I implement functions and the class to make a draggable line ie the objective function.

**from** **lines** **import** Line2dcop

**from** **lines\_for\_lp** **import** get\_lineLP

**from** **linprog** **import** max\_obj

**from** **func\_plot1** **import** plot\_line

*#class for objective line that can be moved to determine optimal solution*

**class** **DraggableLine**(Line2dcop):

**def** \_\_init\_\_(self,line2d,line, optimum\_x\_y,optimum\_sol):

self.optimum\_sol = optimum\_sol

self.optimum\_x\_y = optimum\_x\_y *#coordinates of optimum solution*

Line2dcop.\_\_init\_\_(self,line2d,line) *#self.Line2D = line2d, self.line = line*

self.press = None

**def** connect(self):

*"""connect to all the events needed"""*

self.cidpress = self.line2d.figure.canvas.mpl\_connect(

"button\_press\_event", self.on\_press)

self.cidrelease = self.line2d.figure.canvas.mpl\_connect(

"button\_release\_event", self.on\_release)

self.cidmotion = self.line2d.figure.canvas.mpl\_connect(

"motion\_notify\_event", self.on\_motion)

**def** on\_press(self,event):

*"""Check whether the mouse is over line;if so store some data."""*

**if** event.inaxes != self.line2d.axes:

**return**

contains, attrd = self.line2d.contains(event) *#checks to see if event happened in line*

**if** **not** contains:

**return**

self.press = (event.xdata, event.ydata) *#where line was clicked*

**def** on\_motion(self,event):

*"""Move the line if the mouse is over line."""*

**if** self.press **is** None **or** event.inaxes != self.line2d.axes:

**return** *# if no event occurs in line or wrong axes return*

(xpress, ypress) = self.press

dx = event.xdata - xpress *#amount to shift line by in x direction*

dy = event.ydata - ypress *#amount to shift line by in y direction*

*#get line data and add or subtract from the line depending on dx and dy*

xdata = self.line2d.get\_xdata()

ydata = self.line2d.get\_ydata()

self.line2d.set\_xdata(xdata + dx)

self.line2d.set\_ydata(ydata + dy)

self.line2d.figure.canvas.draw()

self.press = event.xdata, event.ydata *#new press point to keep dx and dy constant*

*# #we want to stop when we hit optimum solution*

*# #line is essentially changing while being shifted*

*# #in short extrapolate new line made and see if the optimum solution will be on it if so disconnect*

*# place\_to\_stop = self.optimum\_x\_y[1] - np.interp(self.optimum\_x\_y[0], self.line2d.get\_xdata(),self.line2d.get\_ydata())*

*# if -1< place\_to\_stop <1:#making solution more robust*

*# print("We are there")*

*# self.disconnect()*

**def** on\_release(self, event):

*"""Clear button press information"""*

self.press = None

self.line2d.figure.canvas.draw()

**def** disconnect(self):

*"""Disconnect all callbacks"""*

self.line2d.figure.canvas.mpl\_disconnect(self.cidpress)

self.line2d.figure.canvas.mpl\_disconnect(self.cidrelease)

self.line2d.figure.canvas.mpl\_disconnect(self.cidmotion)

**def** get\_objective\_line(question,x\_range,axis,len\_axis, chosen = None, minimax = None):

*"""Function to return objecitve line"""*

**if** **not** chosen:

line = get\_lineLP(1,len\_axis)

**else**:

line = chosen

*#maximizing objective function to constraints*

optimum\_solution,optimum\_intersection, optimum\_x\_y = max\_obj(question,minimax,objective = line)

optimum\_x\_y = list(optimum\_x\_y)

*#getting an objective line*

Line2Dcop = plot\_line(line,x\_range, axis,"objective line",objective = True)

ob\_line = DraggableLine(Line2Dcop,line, optimum\_x\_y,optimum\_solution)

ob\_line.connect()

**return** ob\_line

## func\_plot1

This module will contain most functions related to graph plotting.

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

*#making a line in terms of x*

*# if zerodivision error => gradient is infinity, return value of x as an array(solution done to workaround matplotlib limitations)*

**def** in\_terms\_y(equation,x):

**try**:

y = -equation.x/equation.y\*x + equation.rhs/equation.y

**return** y

**except** **ZeroDivisionError**:

new\_arr = np.array([])

**for** i **in** range(len(x)):

new\_arr = np.append(new\_arr,equation.rhs)

**return** new\_arr

*#plots a set of lines on graph*

*#also shades regions*

**def** plot\_linobj(equations,x, ax):

*"""Main function for plotting inequalities"""*

ready\_plots = []

**if** ax:

*#fig,ax = plt.subplots(figsize = (5,5),facecolor = "#EFEFE0")*

*#ax.set\_facecolor("#EFEFE0")*

**for** line **in** equations:

y = in\_terms\_y(line,x) *# create an array in terms of x (ie what we will plot)*

ready\_plots.append(y) *# append array to list*

*# determining region to shade based on lines gradient attribute*

*#if gradient is not infinity use fill\_between*

*#figured its because of intercept that determines side of line 0,0 lies*

*#using that to determine where to shade*

**if** line.m != np.inf **and** line.m != 0:

ax.plot(x,y,label = line.\_\_str\_\_(),picker = True) *#plot line on graph, label will be string representation i defined in lines module*

test = line.y \* 0 + line.x \* 0 *# new proposed. testing a value of in region then shading based on that*

**if** line.status == ">=":

**if** line.c > 0:

**if** test > line.rhs:

ax.fill\_between(x,y,9999999,alpha = 0.2)

**elif** test < line.rhs:

ax.fill\_between(x,y,alpha = 0.2)

*#nb we wont want lines that are >= with c < 0*

**elif** line.c < 0:

**if** test > line.rhs:

ax.fill\_between(x,y,alpha = 0.2)

**elif** test < line.rhs:

ax.fill\_between(x,y,9999999,alpha = 0.2)

*# cant use 0,0 as a test for lines whose intercept is the origin*

**elif** line.c == 0:

test = line.x \* 1 + line.y \* 0

**if** test > line.rhs:

plt.fill\_between(x,y,9999999,alpha = 0.2)

**elif** test < line.rhs:

ax.fill\_between(x,y,alpha = 0.2)

*######*

**elif** line.status == "<=":

**if** line.c > 0:

**if** test < line.rhs:

ax.fill\_between(x,y,9999999,alpha = 0.2)

**elif** test > line.rhs:

ax.fill\_between(x,y,alpha = 0.2)

**elif** line.c < 0:

**if** test < line.rhs:

ax.fill\_between(x,y,alpha = 0.2)

**elif** test > line.rhs:

ax.fill\_between(x,y,9999999,alpha = 0.2)

*#cant use 0,0 as a test for lines whose intercept is the origin*

**elif** line.c == 0:

test = line.x \* 1 + line.y \* 0

**if** test < line.rhs:

ax.fill\_between(x,y,9999999,alpha = 0.2)

**elif** test > line.rhs:

ax.fill\_between(x,y,alpha = 0.2)

**elif** line.m == 0:

ax.plot(x,y,label = line.\_\_str\_\_(),picker = True) *#plot line on graph, label will be string representation i defined in lines module*

**if** line.status == "<=":

ax.fill\_between(x,y,9999999,alpha = 0.2)

**elif** line.status == ">=":

ax.fill\_between(x,y,-9999999,alpha = 0.2)

*#if gradient is infinity use fill\_betweenx*

**elif** line.m == np.inf:

ax.plot(y,x,label = line.\_\_str\_\_(),picker = True

) *#plot line on graph, label will be string representation i defined in lines module*

**if** line.status == ">=":

ax.fill\_betweenx(x,y,-9999999,alpha = 0.2)

**elif** line.status == "<=":

ax.fill\_betweenx(x,y,9999999,alpha = 0.2)

*#adding legend, setting axis*

ax.axis([-2,np.amax(x),-2,np.amax(x)])

ax.legend()

**def** plot\_line(line, x, ax, label,objective=False):

*""" function majorly to return line with matplotlib properties"""*

new\_x = np.arange(-(np.amax(x)\*10),np.amax(x)\*10,10)

*#px + qy = c*

**if** objective: *#for objective line i want it to always be on screen so its length will be way longer*

y = in\_terms\_y(line,new\_x)

line = ax.plot(new\_x,y, label = label)[0]

**return** line

**def** plot(line,x,ax,canvas, ):

*"""*

*Function to just plot a single line*

*Parameters*

*----------*

*line : Line object*

*DESCRIPTION.*

*x : array of x points*

*ax : axes*

*axes to plot line on*

*Returns*

*-------*

*None.*

*"""*

y = in\_terms\_y(line, x)

**if** line.y == 0:

ax.plot(y,x, )*#label = line.\_\_str\_\_())*

**else**:

ax.plot(x,y,) *#label = line.\_\_str\_\_())*

canvas.draw()

**def** empty\_plot(x):

fig,ax = plt.subplots(figsize = (6,6),facecolor = "#EFEFE0")

ax.set\_facecolor("white")

ax.axis([-2,np.amax(x),-2,np.amax(x)])

**return** fig,ax

## questions

This module contains most functions and questions and classes related to questioning. It also most methods for validating user entries.

**import** **random**

**from** **graph\_nodes** **import** Graph

**from** **lines** **import** lines **as** l

**import** **numpy** **as** **np**

**class** **Graphing\_question**():

**def** \_\_init\_\_(self, access\_mode, nec\_info):

*#access\_modes - y-intercept,x-intercept,area to shade, click*

*#nec\_info is majorly a line*

*"""*

*Parameters*

*----------*

*access\_mode : STRING*

*SUB AREA OF GRAPHING LINEAR PROGRAMMING PROBLEMS*

*nec\_info : ANY*

*INFO NEEDED TO MAKE QUESTION*

*part:int*

*Phase of question- 1 x or y intercept ,2 - drawing and shading*

*Returns*

*-------*

*None.*

*"""*

self.access\_mode = access\_mode

self.nec\_info = nec\_info

**if** self.access\_mode =="area to shade":

self.part = 2

**elif** self.access\_mode == "y-intercept" :

self.part =1

**elif** self.access\_mode == "x-intercept":

self.part = 1

**if** self.part:

self.style = random.randint(0,1)

self.question\_stems = {"c":[f"What is the {access\_mode} of the line {nec\_info}?",

f"is the {access\_mode} of the line {nec\_info}"]}

self.question\_stem = self.question\_stems["c"][self.style]

**def** get\_question(self, question\_type, axes = None, lenaxis = None):

self.question\_type = question\_type

**if** question\_type == "fill\_blank":

**if** self.access\_mode == "y-intercept" :

self.answer = self.nec\_info.c

**elif** self.access\_mode == "x-intercept":

self.answer = self.nec\_info.xc

**elif** self.access\_mode == "area to shade":

places, letters, correct = put\_letters(self.nec\_info, lenaxis, axes)

self.answer = letters[places.index(correct)]

*#print(self.answer)*

**return** self,self.question\_stem

**elif** question\_type == "MCQ":

**if** self.part ==1:

a = f"({self.nec\_info.xc},0)"

b = f"({self.nec\_info.c},0)"

c = f"(0,{self.nec\_info.xc})"

d = f"(0,{self.nec\_info.c})"

e = f"(0,0)"

self.options = [a,b,c,d,e]

**if** self.access\_mode == "y-intercept":

self.answer = d

**elif** self.access\_mode == "x-intercept":

self.answer = a

**return** self,self.question\_stem,self.options

**elif** self.part == 2:

places, letters, correct = put\_letters(self.nec\_info, lenaxis, axes)

a,b = letters

self.options = [a,b]

self.answer = self.options[places.index(correct)]

**return** self, self.question\_stem,self.options

**elif** question\_type == "touch":

self.question\_stem = "Click on the part of graph that should be shaded out"

**return** self, self.question\_stem

**def** validate(equation,event):

*"""function to validate where user clicks"""*

**if** equation.status == ">=":

**if** (event.xdata \* equation.x + event.ydata \* equation.y) > equation.rhs:

**return** "Wrong"

**else**:

**return** "CorrecT"

**elif** equation.status == "<=":

**if** event.xdata \* equation.x + event.ydata \* equation.y < equation.rhs:

**return** "Wrong"

**else**:

**return** "CorrecT"

**def** validates(equations,event):

*"""Validates user click in feasible region"""*

**for** equation **in** equations:

lhs = equation.x \*event.xdata + equation.y \* event.ydata

**if** equation.status == "<=":

**if** lhs < equation.rhs:

**continue**

**else**:

**return** False

**elif** equation.status == ">=":

**if** lhs > equation.rhs:

**continue**

**else**:

**return** False

**return** True

**def** suit\_y(ob\_func,entry, lenaxis):

*"""Testing for a suitable yintercept for objective function"""*

*#creating a new line and testing to see if c will be suitable*

equation = l(x = ob\_func.x,y = ob\_func.y,rhs =ob\_func.test)

**if** equation.m >0 :

new\_eq = l(equation.x,equation.y,entry)

**if** lenaxis>new\_eq.c > 0 **or** 0<new\_eq.xc < lenaxis:

hold = True

**else**:

hold = False

**elif** equation.m < 0:

new\_eq = l(equation.x,equation.y, entry)

**if** new\_eq.c > 0 **and** new\_eq.output(lenaxis) < lenaxis:

hold = True

**else**:

hold = False

**elif** equation.m == 0:

**if** entry > 0 **and** entry < lenaxis:

hold = True

**else**:

hold = False

**elif** equation.m == np.inf:

**if** entry == "NA":

hold = True

**else**:

hold = False

new\_eq = l(equation.x,equation.y,entry)

**return** hold, new\_eq

**from** **lines\_for\_lp** **import** get\_lineLP

**from** **linprog** **import** max\_obj

**import** **statistics** **as** **stats**

**def** filter\_cycles(cycles):

*"""filters the cycles returning cycle with greatest number of nodes"""*

lengths = [len(cycle) **for** cycle **in** cycles]

**for** cycle **in** cycles:

**if** len(cycle) == max(lengths):

**return** cycle

**def** lp\_question(lines, cycles):

*"""*

*Parameters*

*----------*

*lines : List*

*has lines that form LP problem*

*nodes : List*

*has nodes that form cycle on graph*

*Returns*

*-------*

*lines : list*

*lines have been given inequality symbols*

*cop\_nodes : list containing vertices of feasible region*

*"""*

nodes = filter\_cycles(cycles)

xs = [node.x **for** node **in** nodes] *#xcoordinate of all nodes*

ys = [node.y **for** node **in** nodes] *#ycoordinate of all nodes*

xbar = stats.mean(xs) *#mean xcoordinate*

ybar = stats.mean(ys) *#mean ycoordinate*

ytests = [line.output(xbar) **for** line **in** lines] *#y value of lines at xbar*

**for** ytest ,line **in** zip(ytests,lines):

**if** ytest == "NA": *#for lines such as x = blah*

**if** xbar < line.rhs:

line.status = "<="

**elif** xbar > line.rhs:

line.status = ">="

*#set inequality of line based on comaprison with ybar*

**elif** ybar < ytest:

line.status = "<="

**elif** ybar > ytest:

line.status = ">="

*#also want to get the nodes that form feasible region need to filter nodes that form cycle to those in feasible region*

cop\_nodes = nodes[:]

**for** node **in** nodes:

**for** line **in** lines:

**if** line.status == "<=":

*#another computer issue gave -59.6200000000000000005 when i wanted -59.62*

**if** round(node.x \* line.x + node.y \* line.y,10) > line.rhs:

cop\_nodes.remove(node)

**elif** line.status == ">=":

**if** round(node.x \* line.x + node.y\* line.y,10) < line.rhs:

cop\_nodes.remove(node)

**return** lines, cop\_nodes

**class** **LPquestion**():

*"""Linear programming Ineqaulties"""*

**def** \_\_init\_\_(self,num\_of\_lines,lenaxis):

*#getting the lines*

cycles = None

*#getting the nodes*

*#finding the cycles*

**while** **not** cycles:

lines, x = get\_lineLP(num\_of\_lines ,lenaxis)

nodes = max\_obj(lines, "get\_nodes",(lenaxis,lenaxis))

graph = Graph(nodes)

A = graph.get\_node((0,0))

cycles = graph.find\_cycles(A,[A])

*#equations that make the linear programming question*

self.equations, self.nodes = lp\_question(lines, cycles)

self.minimax = random.choice(("high","low"))

**def** put\_letters(line, lenaxis, axes):

*"""function to return points at which i can put letters i want """*

line = line

lenaxis\_tup = (lenaxis,lenaxis)

**if** line.rhs == lenaxis:

**return** None

**else**:

lines\_add = [l(1,0,100),l(0,1,100), l(1,0,0),l(0,1,0)]

*#putting in the boundaries*

lines\_add.append(line)

*#getting nodes for graph*

nodes = max\_obj(lines\_add, "get\_nodes", lenaxis = lenaxis\_tup)

graph = Graph(nodes)

b = graph.get\_node((0,100))

new\_cycles = []

cycles = graph.find\_cycles(b,[b]) *# found cycles*

*###if the cycle contains node whose borders are the boundaries of the square remove it*

**for** cycle **in** cycles:

**if** len(cycle) != len(nodes):

new\_cycles.append(cycle)

*#print(new\_cycles)*

places = [] *#places to put letters with correspoding letters*

*#we've gotten new cycles*

**if** len(new\_cycles) == 2:

*#want to recieve an axis and put letters at the centre of the cycles*

**for** cycle **in** new\_cycles:*#*

xs = [node.x **for** node **in** cycle] *#xcoordinate of all nodes*

ys = [node.y **for** node **in** cycle] *#ycoordinate of all nodes*

xbar = stats.mean(xs) *#mean xcoordinate*

ybar = stats.mean(ys) *#mean ycoordinate*

places.append((xbar,ybar))*#places*

correct = None

**for** place **in** places:

**if** line.status == ">=":

**if** place[0] \* line.x + place[1] \* line.y < line.rhs:

correct = place

**elif** line.status == "<=":

**if** place[0] \* line.x + place[1] \* line.y > line.rhs:

correct = place

letters = ["A","B","C","D","E"]

letters = letters[:len(places)]

**for** letter, place **in** zip(letters, places):

axes.text(place[0],place[1],letter)

**return** places,letters, correct

## freq\_used

This module contains the classes of the widgets I will frequently use in the GUI.

**import** **customtkinter** **as** **ctk**

**class** **Page**(ctk.CTkFrame):

*"""All pages inherit from this class"""*

**def** \_\_init\_\_(self,master,fg\_color = "#EFEFE0",\*\*kw):

super().\_\_init\_\_(master,corner\_radius=0,fg\_color = fg\_color,\*\*kw)

self.master = master

**def** clear(self):

**for** child **in** self.winfo\_children():

child.destroy()

**class** **def\_button**(ctk.CTkButton):

*"""All buttons inherit from this class"""*

**def** \_\_init\_\_(self,master,border\_width=2,fg\_color ="#EFEFE0", border\_color="white",

hover\_color="grey92",font=("Arial",11),text\_color="#7A513A",text="Place\_holder",\*\*kw):

super().\_\_init\_\_(master,border\_width =border\_width,border\_color =border\_color,hover\_color = hover\_color,

font=font,text\_color = text\_color,fg\_color = fg\_color, text = text,\*\*kw)

**class** **def\_lbl**(ctk.CTkLabel):

*"""All labels will inherit from this class."""*

**def** \_\_init\_\_(self,master, fg\_color ="#EFEFE0",text\_color="#7A513A",font = ("Arial",20),\*\*kw):

super().\_\_init\_\_(master,fg\_color=fg\_color, text\_color=text\_color,font=font, \*\*kw)

**class** **Radio\_group**(Page):

*"""Mini frame that holds multiple radiobuttons sharing same variable"""*

**def** \_\_init\_\_(self,master,number,options):

super().\_\_init\_\_(master,)

self.master = master

self.variable = ctk.StringVar(self,"")

self.rad\_btns = []

self.options = options

**for** i **in** range(number):

radio\_button = ctk.CTkRadioButton(self,variable = self.variable, value = options[i],text = options[i],

font = ("Arial",20), text\_color="#7A513A" ,border\_color = "white")

radio\_button.grid(row = i,column = 0,sticky = "w")

self.rad\_btns.append(radio\_button)

## main

Main module. Ties everything together to produce the working solution.

**import** **customtkinter** **as** **ctk**

**from** **questions** **import** LPquestion

**from** **freq\_used** **import** Page, def\_button,def\_lbl,Radio\_group

**import** **random**

**from** **matplotlib** **.backends.backend\_tkagg** **import**(FigureCanvasTkAgg,NavigationToolbar2Tk)

**import** **matplotlib.pyplot** **as** **plt**

**from** **func\_plot1** **import** plot ,empty\_plot,plot\_linobj

**import** **numpy** **as** **np**

**from** **questions** **import** Graphing\_question, validate, validates,suit\_y

**from** **objective\_line** **import** get\_objective\_line

**from** **lines\_for\_lp** **import** get\_lineLP

**class** **Done\_Section**(Page):

*"""Page showing user has completed a section"""*

**def** \_\_init\_\_(self,master):

super().\_\_init\_\_(master)

self.grid\_columnconfigure(0, weight = 1)

self.grid\_rowconfigure(0, weight = 1)

message\_lbl = def\_lbl(self, text = "You have completed this section", font = ("Arial",40))

message\_lbl.grid(row = 0,column = 0)

return\_lbl = def\_button(self, text = "return to Practice Page", command = **lambda**:self.master.show\_page(Practice\_Page))

return\_lbl.grid(row = 0, column = 0,sticky = "se")

**class** **Home\_Page**(Page):

*"""Home page- First one that will be seen"""*

**def** \_\_init\_\_(self,master):

super().\_\_init\_\_(master)

self.grid\_columnconfigure(0, weight = 1)

*##heading\_frame ie frame that contains linear programming*

heading\_frame = ctk.CTkFrame(self,height = 200, fg\_color="#EFEFE0", border\_width=2, border\_color = "white")

heading\_frame.grid(row = 0, column = 0, sticky = "ew", padx = 50, pady = 90)

heading\_frame.grid\_propagate(False)

heading\_frame.grid\_columnconfigure(0, weight = 1)

heading\_frame.grid\_rowconfigure(0, weight = 1)

lbl\_heading = ctk.CTkLabel(heading\_frame,text = "LINEAR PROGRAMMING",fg\_color ="#EFEFE0",font=("Arial",45, "bold"),

text\_color= "#7A513A")

lbl\_heading.grid(row=0,column = 0, pady = 20, sticky = "ew", padx = 10)

*##lower frame*

lower\_frame = ctk.CTkFrame(self,fg\_color ="#EFEFE0", height = 150,)

lower\_frame.grid(row = 1,column = 0, sticky = "ew", padx = 50,pady = 50)

lower\_frame.grid\_columnconfigure(0, weight = 1)

*##buttons*

*#practice button*

self.practice\_button = def\_button(lower\_frame, text = "Practice",border\_width= 2,font=("Arial",40), width = 300,

height = 100,

command = **lambda**:self.master.show\_page(Practice\_Page))

self.practice\_button.grid(row = 0,column = 0,pady = 10, sticky = "nse")

*#revision button*

self.revision\_button = def\_button(lower\_frame, text = "Revision", border\_width= 2,font=("Arial",40), width = 300, height = 100)

self.revision\_button.grid(row = 0, column = 0,pady =10, sticky = "nsw")

**class** **Practice\_Page**(Page):

*"""Page shown after Practice button is clicked"""*

**def** \_\_init\_\_(self,master):

super().\_\_init\_\_(master)

self.grid\_columnconfigure(1, weight = 1)

self.grid\_rowconfigure(1, weight = 1)

self.gen\_revision\_btn = def\_button(self,text = "General Revision", font= ("Arial",65))

self.gen\_revision\_btn.grid(row = 1,column = 1)

self.sol\_vertex\_btn = def\_button(self,text = "Solutions via vertex method",font= ("Arial",25),anchor = "centre",

command = **lambda**: self.master.show\_page(Sol\_vertex))

self.sol\_vertex\_btn.grid(row = 1, column =0, sticky = "w")

self.forming\_lp\_btn = def\_button(self,text = "Formulating a LP problem",font = ("Arial",25))

self.forming\_lp\_btn.grid(row = 0,column = 1,sticky = "s")

self.graphing\_lp\_btn = def\_button(self,text = "Graphing a LP problem",font = ("Arial",25),

command = **lambda**: self.master.show\_page(Graphing))

self.graphing\_lp\_btn.grid(row = 1, column = 2)

self.objective\_line\_btn = def\_button(self, text = "Solutions via objective line",font = ("Arial",25),

command = **lambda**:self.master.show\_page(Sol\_obj))

self.objective\_line\_btn.grid(row = 2, column = 1,sticky = "n")

**class** **Main**(ctk.CTk):

**def** \_\_init\_\_(self):

super().\_\_init\_\_(fg\_color= "#EFEFE0")

self.question = None

self.width = self.winfo\_screenwidth()

self.height = self.winfo\_screenheight()

self.height \*=0.75\*0.75

self.width \*=0.75\*0.75

self.resizable(True,True)

self.minsize(self.width,self.height)

self.geometry(f"{self.width}x{self.height}")

self.pages = {}

self.grid\_rowconfigure(0, weight = 1)

self.grid\_columnconfigure(0, weight =1)

self.graphing = 0

self.rest = 0

**for** F **in** (Home\_Page,Practice\_Page, Graphing, Done\_Section,Sol\_obj, Sol\_vertex): *#pages dictionary will have Pages name as keys and page object as value*

frame = F(self)

self.pages[F] = frame

self.show\_page(Home\_Page)

**def** show\_page(self,New\_Page):

*#ensuring graphing problems are done first*

**if** self.graphing **or** New\_Page == Home\_Page **or** New\_Page == Practice\_Page **or** New\_Page == Graphing **or** New\_Page == Sol\_obj **or** New\_Page == Sol\_vertex **or** New\_Page == Done\_Section:

**for** page **in** self.pages.keys():

self.pages[page].grid\_remove() *#remove all pages and put on new page*

**if** New\_Page != Graphing **or** self.graphing == 0:

self.pages[New\_Page].grid(row =0, column = 0, sticky = "nsew",padx=20, pady = 20)

*#getting new graphing page as previous one will be solved already*

**else**:

self.pages[Graphing] = Graphing(self)

self.pages[Graphing].grid(row =0, column = 0, sticky = "nsew",padx=20, pady = 20)

**else**:

**print**("Practice Graphing problems First")

**def** replace\_page(self,rep\_Page):

*"""repalce a page we have done"""*

new\_page = rep\_Page(self)

self.pages[rep\_Page] = new\_page

**def** get\_new\_question(self):

*"""get a new overhead question"""*

self.question = LPquestion(3, 100)

**class** **Graphing**(Page):

*"""This page handles Graphing Linear Problems"""*

**def** \_\_init\_\_(self,master):

super().\_\_init\_\_(master)

self.question = LPquestion(3,100) *#top level question*

self.master.question = self.question

self.grid\_columnconfigure(0, weight = 1)

self.grid\_rowconfigure(0 ,weight =1)

*#arranged them all this way to avoid changing row and column configure*

title = def\_lbl(self,text = "Graphing a LP problem",font = ("Arial",40))

title.grid(row = 0,column = 0,sticky = "n",pady = 25)

*#opening text displaying all the inequalities*

opening\_text = "Lets Graph these inequalities:**\n**"

**for** equation **in** self.question.equations:

opening\_text+= "**\n**" + equation.\_\_str\_\_()

self.large\_lbl = def\_lbl(self,text = opening\_text,font=("Arial",25))

self.large\_lbl.grid(row = 0, column =0)

go\_btn = def\_button(self,text = "Lets' GOOO",font = ("Arial",25), command =**lambda**: self.subroutine())

go\_btn.grid(row = 0, column = 0, sticky = "se",padx = 250,pady = 80)

self.current = -1 *#initialised to -1 so intial call will be 0 and so on*

**def** subroutine(self):

*"""Changing equations"""*

self.clear() *#clear frame*

self.current += 1

*#current equation*

self.max\_current = len(self.question.equations) *#max number of equations*

**if** self.current != self.max\_current: *#there are still more equations to solve*

*##load new Page that has graph*

new\_page = Page\_with\_Graph(self,self.question.equations[self.current],100)

new\_page.grid(row = 0,column =0,sticky = "nsew")

**else**:

self.master.graphing +=1

self.master.show\_page(Done\_Section)

self.master.replace\_page(Graphing)

**class** **Page\_with\_Graph**(Page):

*"""This Page will be where users graph equations one by one"""*

**def** \_\_init\_\_(self,master,equation,end\_x):

*##all widgets here shouldnt change for current equation*

super().\_\_init\_\_(master,)*#fg\_color="white",border\_color = "black",border\_width = 2)*

*#values intialised to 0*

self.yintercept = 0

self.xintercept = 0

self.areashade = 0

self.equation = equation

part1 = ("y-intercept","x-intercept","area to shade")

self.generator = (element **for** element **in** part1) *#generator object - can invoke next() to bring up next part of question*

*#when a user has gotten a part correct we get a new question using the next element from the generator*

self.grid\_columnconfigure((0,1),weight = 1)

self.grid\_rowconfigure(1,weight = 1)

*#title at top*

title = def\_lbl(self,text = "Graphing a LP problem",font = ("Arial",40))

title.grid(row = 0,column = 0,sticky = "e", pady = 30)

*#equation currently being worked on*

self.main\_label = def\_lbl(self,text = f"Graphing {equation}",font = ("Arial",30,"underline"))

self.main\_label.grid(row =1,column = 0,sticky = "nw")

self.x = np.arange(0,end\_x+1) *#end of x-axis*

self.lenaxis = end\_x

self.graph\_frame = ctk.CTkFrame(self,fg\_color ="#EFEFE0") *#frame that will hold graph*

self.graph\_frame.grid(row =1,column = 1)

self.fig, self.axes = empty\_plot(self.x)

self.axes.grid()

self.graph\_canvas = FigureCanvasTkAgg(self.fig,self.graph\_frame)

self.graph\_canvas.get\_tk\_widget().grid(row=0,column = 0, sticky = "e")

self.toolbar = NavigationToolbar2Tk(self.graph\_canvas,self.graph\_frame,pack\_toolbar=False)

self.toolbar.grid(row=1, column = 0)

self.question\_frame = Page(self)*#,fg\_color = "grey")*

*#contents of this frame changes based on question part*

self.question\_frame.grid(row = 1,column = 0,sticky = "nsew",pady= 100)

self.question\_frame.grid\_columnconfigure(0, weight = 1)

self.question\_frame.grid\_rowconfigure((0,1)

,weight = 1)

self.refresh()

**def** refresh(self):

*#putting dots at x and y intercept when done by user*

**if** self.yintercept **and** self.equation.c: *#if the line has a yintercept*

c1 = plt.Circle([0,self.equation.c],radius = 1,picker = True, color = "black")

self.axes.add\_artist(c1)

self.graph\_canvas.draw()

**if** self.xintercept **and** self.equation.xc:

c1 = plt.Circle([self.equation.xc,0],radius = 1,picker = True, color = "black")

self.axes.add\_artist(c1)

**if** self.yintercept **and** self.xintercept:

plot(self.equation,self.x,self.axes,self.graph\_canvas)

self.graph\_canvas.draw()

self.question\_frame.clear() *#clear the question frame new question will be put in here*

question\_types = ["fill\_blank","MCQ",]

question\_types1 = ["fill\_blank","MCQ","touch"]

**try**:

*#access mode of the question would be the next element from the generator*

self.access\_mode = next(self.generator)

*#including touch type questions for area to shade questions*

self.question\_type = random.choice(question\_types) **if** self.access\_mode != "area to shade" **else** random.choice(question\_types1)

**if** self.equation.rhs == self.lenaxis **or** self.equation.x==1 **and** self.equation.y ==1 **and** self.equation.rhs == self.lenaxis:

**if** self.access\_mode == "area to shade":

self.question\_type = "touch"

**elif** (self.equation.c == None **or** self.equation.xc == None) **and** self.access\_mode == "area to shade":

self.question\_type = "touch"

**if** self.question\_type == "fill\_blank": *#should try to optimize this more*

self.question\_obj,self.question\_text, = Graphing\_question(self.access\_mode,nec\_info= self.equation).get\_question(self.question\_type,axes = self.axes,lenaxis = self.lenaxis) *#question object,question prompt*

self.entry = ctk.CTkEntry(self.question\_frame)

**if** self.question\_obj.style == 1: *# \_\_\_\_ is the blah blahblah - entry before*

*#entry widget for user answer*

self.entry.grid(row=0,column=0,sticky = "nw")

*#question stem*

self.question\_lbl = def\_lbl(self.question\_frame,text = self.question\_text)

self.question\_lbl.grid(row=0,column = 0,sticky = "n" ,padx = 20)

**elif** self.question\_obj.style == 0: *# what is the blah blah blah --entry is after*

self.entry.grid(row = 0,column = 0,sticky = "ne",padx = 70)

self.question\_lbl = def\_lbl(self.question\_frame,text = self.question\_text)

self.question\_lbl.grid(row=0,column = 0,sticky = "nw" )

**elif** self.question\_type == "MCQ":

self.question\_obj,self.question\_text,self.options = Graphing\_question(self.access\_mode, self.equation).get\_question(self.question\_type,axes=self.axes,lenaxis=self.lenaxis) *#question object,question prompt, options*

*#question stem*

self.question\_lbl = def\_lbl(self.question\_frame,text = self.question\_text)

self.question\_lbl.grid(row=0,column = 0,sticky = "nw")

self.radio\_group = Radio\_group(self.question\_frame,len(self.options), self.options) *#radio buttons*

self.radio\_group.grid(row = 0, column = 0,sticky = "w")

**elif** self.question\_type == "touch":

self.question\_obj,self.question\_text = Graphing\_question(self.access\_mode, self.equation).get\_question(self.question\_type) *#question object,question prompt, options*

self.question\_lbl = def\_lbl(self.question\_frame,text = self.question\_text)

self.question\_lbl.grid(row=0,column = 0,sticky = "nw")

self.cidpress = self.graph\_canvas.mpl\_connect("button\_press\_event", self.submit)

*#submit , next part, next equation button*

self.next\_button = def\_button(self,text = "Submit",command =self.submit)

self.next\_button.grid(row = 2,column = 1)

self.graph\_canvas.draw()

**except** **StopIteration**:

*#all elements in generator have been used up ==> we are done with all parts of equation*

plot\_linobj([self.equation], self.x, self.axes)

*#self.axes.legend()*

self.graph\_canvas.draw()

*#configure the button to move on to next equation*

self.next\_button.configure(command= self.master.subroutine,text = "Next Equation")

**def** submit(self,event = None):

*"""Validating user entry regardless of question type"""*

*###we need to deal with value errors*

*# if dealing with x intercept or y intercept part is 1*

**try**:

**if** self.question\_type == "fill\_blank":

self.user\_entry = self.entry.get() *#get user input*

**if** self.question\_obj.part ==1: *#we are dealing with x or y intercept*

**if** self.equation.m != 0 **and** self.equation.m != np.inf:

**if** float(self.user\_entry) == self.question\_obj.answer: *#compare float with answer*

**print**("CorrecT")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

*#set part of question done to 1*

**if** self.access\_mode == "y-intercept":

self.yintercept = 1

**elif** self.access\_mode == "x-intercept":

self.xintercept = 1

self.next\_button.configure(command = self.refresh,text = "Next Part") *#go to next part if correct*

**else**:

**print**("Wrong")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**elif** self.equation.m == np.inf:

**if** self.access\_mode == "y-intercept":

**if** self.user\_entry == "NA":

**print**("CorrecT")

self.next\_button.configure(command = self.refresh,text = "Next Part") *#go to next part if correct*

self.yintercept =1

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

**else**:

**print**("Wrong")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**elif** self.access\_mode == "x-intercept":

**if** float(self.user\_entry) == self.question\_obj.answer: *#compare float with answer*

**print**("CorrecT")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.xintercept = 1

self.next\_button.configure(command = self.refresh,text = "Next Part") *#go to next part if correct*

**else**:

**print**("Wrong")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**elif** self.equation.m == 0:

**if** self.access\_mode == "x-intercept":

**if** self.user\_entry == "NA":

**print**("Correct")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.xintercept = 1

self.next\_button.configure(command = self.refresh,text = "Next Part") *#go to next part if correct*

**else**:

**print**("Wrong")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**elif** self.access\_mode=="y-intercept":

**if** float(self.user\_entry) == self.question\_obj.answer: *#compare float with answer*

**print**("CorrecT")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.yintercept = 1

self.next\_button.configure(command = self.refresh,text = "Next Part") *#go to next part if correct*

**else**:

**print**("Wrong")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**elif** self.question\_obj.part == 2:

**if** self.user\_entry == self.question\_obj.answer:

**print**("CorrecT")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.areashade = 1

self.next\_button.configure(command = self.refresh,text = "Next Part") *#go to next part if correct*

**else**:

**print**("Wrong")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**elif** self.question\_type == "MCQ":

self.user\_entry = self.radio\_group.variable.get() *#get option chosen by user*

**if** self.user\_entry == self.question\_obj.answer:

**print**("CorrecT")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

**if** self.access\_mode == "y-intercept":

self.yintercept = 1

**elif** self.access\_mode == "x-intercept":

self.xintercept = 1

**elif** self.access\_mode == "area to shade":

self.areashade = 1

self.next\_button.configure(command = self.refresh,text = "Next Part")*# go to next part if correct*

**else**:

**print**("Wrong")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**elif** event.xdata **and** event.ydata: *#means graph was clicked- an area to shade question*

test = validate(self.equation,event)

**if** test == "CorrecT":

**print**("CorrecT")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.graph\_canvas.mpl\_disconnect(self.cidpress)

self.next\_button.configure(command = self.refresh,text = "Next Part")*# go to next part if correct*

**else**:

**print**("Wrong")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**except** **ValueError**:

invalid = def\_lbl(self.question\_frame, text = "Invalid input!", font = ("Arial",30))

invalid.grid(row = 2,column = 0, sticky = "ew")

**print**("Invalid input")

**class** **Sol\_obj**(Page):

**def** \_\_init\_\_(self,master):

super().\_\_init\_\_(master)

self.minimax = self.master.question.minimax

**print**(self.minimax)

*#text when describing if problem should be minimized or maximized*

self.minimax\_text = "Maximize" **if** self.minimax == "high" **else** "Minimize"

self.grid\_columnconfigure((0,1), weight = 1)

self.grid\_rowconfigure(2, weight = 1)

self.equations = self.master.question.equations

*#question frame changes while user interacts*

self.question\_frame = Page(self)

self.question\_frame.grid(row = 2,column = 0,sticky = "nsew")

self.question\_frame.grid\_rowconfigure((0,2), weight = 1)

self.question\_frame.grid\_columnconfigure(0, weight = 1)

end\_x = 100

*#title*

title = def\_lbl(self, text = "Solution using objective line",font = ("Arial",40))

title.grid(row = 0,column = 0, pady = 25,sticky = "n")

self.x = np.arange(0,end\_x+1) *#end of x-axis*

self.lenaxis = end\_x

self.graph\_frame = Page(self) *#frame that will hold graph*

self.graph\_frame.grid(row =2,column = 1 )

*#graph*

self.fig, self.axes = empty\_plot(self.x)

plot\_linobj(self.master.question.equations, self.x,ax=self.axes)

self.axes.grid()

*#self.axes.get\_legend().remove()*

self.graph\_canvas = FigureCanvasTkAgg(self.fig,self.graph\_frame)

self.graph\_canvas.get\_tk\_widget().grid(row=0,column = 0, sticky = "e")

self.toolbar = NavigationToolbar2Tk(self.graph\_canvas,self.graph\_frame,pack\_toolbar=False)

self.toolbar.grid(row=1, column = 0)

*#objective function*

self.obj\_line = get\_lineLP(1,self.lenaxis, objective = True)

m\_lists = [line.m **for** line **in** self.equations]

*#to ensure objectve line is not parallel to any lines*

**while** self.obj\_line.m **in** m\_lists:

self.obj\_line = get\_lineLP(1, self.lenaxis, objective = True)

*#opening text displaying all the inequalities*

opening\_text = f"Lets {self.minimax\_text} {self.obj\_line} subject to these inequalities:**\n**"

**for** equation **in** self.equations:

opening\_text+= "**\n**" + equation.\_\_str\_\_()

*#lbl for opening text*

open\_lbl = def\_lbl(self.question\_frame, text = opening\_text)

open\_lbl.grid(row =0, column = 0, sticky = "n", pady = 90)

over\_head\_text = def\_lbl(self, text = f"{self.minimax\_text} {self.obj\_line}")

over\_head\_text.grid(row =1, column = 0 )

*#next\_button*

self.next\_button = def\_button(self, text = "Let's Gooo", command = self.refresh)

self.next\_button.grid(row = 3, column = 1)

self.graph\_canvas.draw()

*#stages of questioning*

stages = ["id\_region","suit\_y","shift\_obj", "get\_point"]

self.stages = (stage **for** stage **in** stages)

**def** refresh(self):

self.next\_button.configure(text = "Find the region",command = None)

**try**:

self.stage = next(self.stages)

self.question\_frame.clear()

*# identify the feasible region*

**if** self.stage == "id\_region":

self.cid\_press = self.graph\_canvas.mpl\_connect("button\_press\_event", self.check)

text = def\_lbl(self.question\_frame, text = "Click on the feasible region")

text.grid(row = 0, column = 0, sticky = "n", pady = 90)

*#choose a suitable y intercept:*

**elif** self.stage == "suit\_y":

text = def\_lbl(self.question\_frame, text = f"The objective function is {self.obj\_line.\_\_str\_\_()}.**\n**Suggest a suitable value for the y intercept of the objective function", )*#justify = "left")*

text.grid(row = 0, column =0, sticky = "nw",pady = 90)

self.entry = ctk.CTkEntry(self.question\_frame)

self.entry.grid(row = 0,column = 0, sticky = "ne", pady = 95, padx = 30)

self.next\_button.configure(text = "Submit",command = self.check)

*#shift objective until feasible region is reached*

**elif** self.stage == "shift\_obj":

text = def\_lbl(self.question\_frame, text = f"Drag the objective function until Optimum solution is reached")

text.grid(row = 0, column = 0, sticky = "n", pady = 90)

self.objective\_function = get\_objective\_line(self.equations, self.x, self.axes, self.lenaxis, chosen = self.obj\_line, minimax= self.minimax)

self.objective\_function.connect()

self.axes.legend()

self.graph\_canvas.draw()

self.next\_button.configure(text = "Submit",command = self.check)

**elif** self.stage == "get\_point":

text = def\_lbl(self.question\_frame, text = f"What are the coordinates of the optimal solution**\n**(xcoordinate,ycoordinate)?")

text.grid(row = 0,column = 0, sticky = "n", pady = 90)

self.entry = ctk.CTkEntry(self.question\_frame)

self.entry.grid(row = 0, column = 1, sticky = "ne", pady= 90, padx = 30)

self.next\_button.configure(text = "Submit",command = self.check)

**except** **StopIteration**:

self.master.get\_new\_question()

self.master.show\_page(Done\_Section)

self.master.replace\_page(Sol\_obj)

**print**("We done now")

**def** check(self, event = None):

**try**:

*#identify feasible region check*

**if** self.stage == "id\_region":

**if** event.xdata **and** event.ydata:

**if** validates(self.equations,event):

**print**("True")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.graph\_canvas.mpl\_disconnect(self.cid\_press)

self.next\_button.configure(text ="Next", command = self.refresh)

**else**:

**print**("False")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

self.next\_button.configure(text ="you shall not pass", command = None)

*#identify suitable y check*

**if** self.stage == "suit\_y":

entry = float(self.entry.get())

state, new = suit\_y(self.obj\_line, entry, self.lenaxis)

**if** state:

**print**("State",state)

self.obj\_line = new

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.next\_button.configure(text ="Next", command = self.refresh)

**else**:

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

*#shift objecitve line until solution*

**if** self.stage == "shift\_obj":

*#we want to stop when we hit optimum solution*

*#line is essentially changing while being shifted*

*#in short extrapolate new line made and see if the optimum solution will be on it if so disconnect*

place\_to\_stop = self.objective\_function.optimum\_x\_y[1] - np.interp(self.objective\_function.optimum\_x\_y[0], self.objective\_function.line2d.get\_xdata(),self.objective\_function.line2d.get\_ydata())

**if** -1< place\_to\_stop <1:*#making solution more robust*

self.objective\_function.disconnect()

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.next\_button.configure(text ="Next", command = self.refresh)

**else**:

**print**("Not there yet")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**if** self.stage == "get\_point":

*#input should be in form (xcoord,ycoord)*

entry = self.entry.get()

xcoordlist,ycoordlist = entry.split(",")

*#getting the x and y coordinate from entry in order to compare with answer*

*#could compare entire string but then id have to specify decimal places and it will result in 0.000*

xcoord = float(xcoordlist.split("(")[1])

ycoord = float(ycoordlist.split(")")[0])

**if** xcoord == float(f"{self.objective\_function.optimum\_x\_y[0]:.3f}") **and** ycoord == float(f"{self.objective\_function.optimum\_x\_y[1]:.3f}"):

**print**("I'm him")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.next\_button.configure(text ="Next", command = self.refresh)

**else**:

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**except** **ValueError**:

**print**("Invalid input")

invalid = def\_lbl(self.question\_frame, text = "Invalid input!", font = ("Arial",30))

invalid.grid(row = 2,column = 0, sticky = "ew")

**def** change\_string\_tup(string):

*"""Change string match from user input to tuple"""*

xcoordlist,ycoordlist = string.split(",")

*#getting the x and y coordinate from entry in order to compare with answer*

xcoord = float(xcoordlist.split("(")[1])

ycoord = float(ycoordlist.split(")")[0])

**return** (xcoord,ycoord)

**from** **linprog** **import** max\_obj

**class** **Sol\_vertex**(Page):

**def** \_\_init\_\_(self,master):

super().\_\_init\_\_(master)

self.minimax = self.master.question.minimax

**print**(self.minimax)

*#text when describing if problem should be minimized or maximized*

self.minimax\_text = "Maximize" **if** self.minimax == "high" **else** "Minimize"

self.grid\_columnconfigure((0,1), weight = 1)

self.grid\_rowconfigure(2, weight = 1)

self.equations = self.master.question.equations

*#question frame changes while user interacts*

self.question\_frame = Page(self)

self.question\_frame.grid(row = 2,column = 0,sticky = "nsew")

self.question\_frame.grid\_rowconfigure((0,2), weight = 1)

self.question\_frame.grid\_columnconfigure(0, weight = 1)

*#title*

title = def\_lbl(self, text = "Solution using Vertex method",font = ("Arial",40))

title.grid(row = 0,column = 0, pady = 25,sticky = "n")

end\_x = 100

self.x = np.arange(0,end\_x+1) *#end of x-axis*

self.lenaxis = end\_x

self.graph\_frame = Page(self) *#frame that will hold graph*

self.graph\_frame.grid(row =2,column = 1 )

*#graph*

self.fig, self.axes = empty\_plot(self.x)

plot\_linobj(self.master.question.equations, self.x,ax=self.axes)

self.axes.grid()

*#self.axes.get\_legend().remove()*

self.graph\_canvas = FigureCanvasTkAgg(self.fig,self.graph\_frame)

self.graph\_canvas.get\_tk\_widget().grid(row=0,column = 0, sticky = "e")

self.toolbar = NavigationToolbar2Tk(self.graph\_canvas,self.graph\_frame,pack\_toolbar=False)

self.toolbar.grid(row=1, column = 0)

*#objective function*

self.obj\_line = get\_lineLP(1,self.lenaxis, objective = True)

m\_lists = [line.m **for** line **in** self.equations]

*#to ensure objectve line is not parallel to any lines*

**while** self.obj\_line.m **in** m\_lists:

self.obj\_line = get\_lineLP(1, self.lenaxis, objective = True)

self.opt\_solution,\_,\_ = max\_obj(self.equations, self.master.question.minimax,lenaxis = self.lenaxis,objective=self.obj\_line) *#optimum solution*

*#opening text displaying all the inequalities*

opening\_text = f"Lets {self.minimax\_text} {self.obj\_line} subject to these inequalities:**\n**"

**for** equation **in** self.equations:

opening\_text+= "**\n**" + equation.\_\_str\_\_()

*#lbl for opening text*

open\_lbl = def\_lbl(self.question\_frame, text = opening\_text)

open\_lbl.grid(row =0, column = 0, sticky = "n", pady = 75)

over\_head\_text = def\_lbl(self, text = f"{self.minimax\_text} {self.obj\_line}")

over\_head\_text.grid(row =1, column = 0 )

*#next\_button*

self.next\_button = def\_button(self, text = "Let's Gooo", command = self.refresh)

self.next\_button.grid(row = 3, column = 1)

self.graph\_canvas.draw()

stages = ["id\_region","get\_coordinates","obtain\_opt"]

self.stages = (stage **for** stage **in** stages)

**def** refresh(self):

self.next\_button.configure(text = "Find the region",command = None)

**try**:

self.stage = next(self.stages)

self.question\_frame.clear()

*# identify the feasible region*

**if** self.stage == "id\_region":

self.cid\_press = self.graph\_canvas.mpl\_connect("button\_press\_event", self.check)

text = def\_lbl(self.question\_frame, text = "Click on the feasible region")

text.grid(row = 0, column = 0, sticky = "n", pady = 75)

**elif** self.stage == "get\_coordinates":

self.question\_frame.grid\_rowconfigure(0, weight = 0)

text = def\_lbl(self.question\_frame, text = f"Write out the coordinates of the vertices of the feasible region**\n**(xcoordinate,ycoordinate), suggest splitting over multiple lines")

text.grid(row = 0,column = 0, sticky= "n",pady = 20 )

self.question\_frame.grid\_rowconfigure(1, weight = 1)

self.text\_box = ctk.CTkTextbox(self.question\_frame)

self.text\_box.grid(row = 1,column = 0, sticky = "nsew" ,padx = 20)

self.next\_button.configure(text = "Submit",command = self.check)

**elif** self.stage == "obtain\_opt":

text = def\_lbl(self.question\_frame, text = f"What is the optimal solution")

text.grid(row = 0,column = 0, sticky = "n", pady = 75)

self.entry = ctk.CTkEntry(self.question\_frame)

self.entry.grid(row = 0, column = 1, sticky = "nw", pady= 75, padx = 30)

self.next\_button.configure(text = "Submit",command = self.check)

**except** **StopIteration**:

self.master.get\_new\_question()

self.master.show\_page(Done\_Section)

self.master.replace\_page(Sol\_vertex)

**print**("We done now")

**def** check(self, event = None):

**try**:

*#identify feasible region check*

**if** self.stage == "id\_region":

**if** event.xdata **and** event.ydata:

**if** validates(self.equations,event):

**print**("True")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.graph\_canvas.mpl\_disconnect(self.cid\_press)

self.next\_button.configure(text ="Next", command = self.refresh)

**else**:

**print**("False")

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

self.next\_button.configure(text ="you shall not pass", command = None)

**elif** self.stage == "get\_coordinates":

nodes = self.master.question.nodes

vertex\_list = [(round(node.x,3),round(node.y,3)) **for** node **in** nodes]

**print**(vertex\_list)

answer\_text = self.text\_box.get("0.0","end")

**print**(answer\_text)

**from** **re** **import** findall

*#regex to find coordinates from user input*

pattern = r"\( \*\d+\.?\d{0,3} \*, \*\d+\.?\d{0,3} \*\)"

user\_answers = findall(pattern,answer\_text)

new\_user\_answers = [change\_string\_tup(answer) **for** answer **in** user\_answers]

**print**(user\_answers)

**print**(new\_user\_answers)

**if** len(user\_answers) == len(vertex\_list):

**print**("We can continue")

**else**:

**print**("There are some you have not identified")

**if** new\_user\_answers == vertex\_list:

**print**("You got it man")

self.next\_button.configure(text ="Next", command = self.refresh)

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

**else**:

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**print**("Check again!")

**elif** self.stage =="obtain\_opt":

answer = self.entry.get()

**print**(self.opt\_solution)

**if** float(answer) == round(self.opt\_solution,3):

**print**("Youve done it")

correct = def\_lbl(self.question\_frame, text = "Correct", font = ("Arial",30))

correct.grid(row = 2,column = 0, sticky = "ew")

self.next\_button.configure(text ="Next", command = self.refresh)

**else**:

wrong = def\_lbl(self.question\_frame, text = "Wrong", font = ("Arial",30))

wrong.grid(row = 2,column = 0, sticky = "ew")

**except** **ValueError**:

invalid= def\_lbl(self.question\_frame, text = "Invalid input", font = ("Arial",30))

invalid.grid(row = 2,column = 0, sticky = "ew")

**print**("Invalid input")

app = Main()

app.mainloop()

# Testing

The following table is an overview of tests conducted on the system to ensure that all expected outcomes of “actions” are the actual outcomes of the “actions”. Testing will also show extent to which objectives have been met.

For many of the objectives, as generation of questions is random and outputs are determined on the input, I use examples with correct outputs to test the efficacy of my algorithms eg for the optimization algorithms, I use example questions from Integral and their answers to test my algorithms. Also, I repeat many of the tests multiple times to ensure that correctness is not a fluke. Satisfied that all algorithms work, I only include 2 repeats for each in my evidence of testing (where necessary). The exact tests are included in the evidence of testing section.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Objective 1: Tool should be able to Generate lines and inequalities.**   * Lines generated should contain values in a specific range eg lines could have a maximum x being 100 and as such beyond that the line should no have no other value. * Lines should be displayed properly in the form ay + bx = c * Lines that won’t be adequately represented with conventional format should be catered to.   + Lines with either a or b being negative, zero or one should be displayed optimally eg -1x + 0y = -30 should be more adequately displayed as   -x = -30.   * Inequalities should be represented the same as lines but with appropriate inequality symbol ie one out of; >=, <= , > , <. | | | | | | | | | | |
| Test No. | Test performed | | | | | Test purpose | Test data | Expected Outcome | | Actual outcome |
| 1.1 | Asked tool to generate line | | | | | Verify tool can generate lines | Number of lines =1 | 1 line should be generated | | 1 line generated  (PASS) |
| 1.2 | Asked tool to generate line | | | | | Verify tool can generate lines | Number of lines =5 | 5 lines generated | | 5 lines generated  (PASS) |
| 1.3 | Line generated displayed | | | | | Verify line generated is displayed properly | X = 2, y = 2, rhs = 100 | Line 2x + 2y = 100  generated | | Line 2x +2y = 100  Generated  (PASS) |
| 1.4 | Line generated displayed | | | | | Verify line generated is displayed properly | X = -1, y = -1, rhs = 100 | Line – x – y = 100  generated | | Line – x – y = 100  Generated  (PASS) |
| 1.5 | Line generated displayed | | | | | Verify line generated is displayed properly | X = 0, y= 1, rhs = 100 | Line y = 100 generated | | Line y = 100 generated  (PASS) |
| 1.6 | Line generated displayed | | | | | Verify line generated is displayed properly | X = 1, y = 0, rhs = 100 | Line x = 100 generated | | Line x = 100 generated  (PASS) |
| 1.7 | Line generated displayed | | | | | Verify line generated is displayed properly | X = 100, y = 100 , rhs = 1000 | Line 100x + 100y = 1000 generated | | Line 100x + 100y = 1000 generated  (PASS) |
| 1.8 | Inequality generated displayed | | | | | Verify line generated is displayed properly | X = 2, y = 2, rhs = 100, status = “>=” | Inequality  “2x + 2y >= 100”  generated | | Inequality  “2x + 2y >= 100”  Generated  (PASS) |
| 1.9 | Inequality generated displayed | | | | | Verify inequality generated is displayed properly | X = -1, y= -0, rhs = 100 | Inequality “-x = 100” displayed | | Inequality “-x = 100” displayed  (PASS) |
| Overall objective | | PASS: Lines are able to be generated and both lines and inequalities are appropriately displayed throughout tool | | | | | | | | |
| **Objective 2: Inequalities used in questions must form a closed cycle.**   * + When inequalities are drawn on graph there should be at least one cycle.   + If more than one cycle, cycle with highest number of edges should be chosen for the question. | | | | | | | | | | |
| 2.1 | Found all combinations of lines | | | | | Verify simultaneous equation solving works and all combinations of lines are found. |  | All combinations of lines found | | All combinations of lines found  (PASS) |
| 2.2 | Filtered combinations of lines to those containing two elements | | | | | Verify combinations of lines are filtered to those with two elements |  | Combinations of lines filtered to those with two elements | | Combinations of lines filtered to those with two elements  (PASS) |
| 2.3 | Removed unnecessary intersections of lines | | | | | Verify intersections of lines not in axis range are removed to ensure nodes will be in axis range |  | Intersections of lines not in specified range removed | | Intersections of lines not in specified range removed  (PASS) |
| 2.4 | Nodes that will form graph are gotten | | | | | Verify nodes corresponding to intersections of lines are returned |  | All nodes returned | | All nodes returned  (PASS) |
| 2.5 | Graph Created from gotten nodes | | | | | Verify that neighbours of nodes are correctly identified |  | Nodes correctly identified and graph is made | | Nodes are correctly identified and graph is made (PASS) |
| 2.6 | Cycle detection algorithm is run on graph | | | | | Ensure algorithm finds all cycles in graph |  | All cycles on graph and nodes that make them are returned | | All cycles on graph and nodes that make them are returned  (PASS) |
| 2.7 | Cycles gotten are filtered to the one with highest number of edges | | | | | Verify cycle with largest number of edges is chosen |  | Cycle with largest number of edges is identified and returned | | Cycle with largest number of edges is identified and returned  (PASS) |
| 2.8 | Lines that define cycle are turned to inequalities | | | | | Verify inequalities generated are correct |  | Inequalities generated define cycle | | Inequalities generated define cycle  (PASS\*\*) |
| 2.9 | Inequalities that form cycle are chosen to be used in question | | | | | Verify inequalities used in question form a cycle |  | Inequalities used in question form a closed cycle | | Inequalities used in question form a closed cycle (PASS) |
| Overall Objective: | | | PASS: Inequalities used in questions form a close cycle. | | | | | | | |
| **Objective 3: Tool should be able to make an Objective function (function that would be optimised subject to constraints)**   * Objective function should be draggable when drawn on graph. * Program should be able to “decipher” when objective function is at the optimal vertex to linear programming problem ie there should be some way to know when objective function is at optimal vertex. | | | | | | | | | | |
| 3.1 | Objective function requested to be gotten | | | | |  |  | Objective function gotten | | Objective function gotten (PASS) |
| 3.2 | Objective function plotted on graph | | | | |  |  | Objective function drawn on graph | | Objective function drawn on graph  (PASS) |
| 3.3  \*\*See explanation of test 2.9 | Objective function clicked on and tried to be dragged | | | | | Confirm objective function is draggable |  | Objective function moves with mouse | | Objective function moves with mouse  (PASS) |
| 3.4 | Output “We are there” when objective function is at optimal solution | | | | | Confirm when dragged program knows when at the optimal vertex |  | Outputs “We are there” when at optimal vertex | | “We are there” printed when at optimal vertex  (PASS) |
| Overall objective: | | | | PASS: Tool is able to get an objective function. When drawn on the graph, the objective function is draggable. Tool knows when objective function is at optimal vertex. | | | | | | |
| **Objective 4:Program should be able to minimize or maximize an objective function subject to constraints (inequalities), returning optimal solution and coordinate at which solution occurs.** | | | | | | | | | | |
| 4.1 | Ran example inequalities involving a maximizing LP problem through optimizing algorithm | | | | | Verify LP problems are solved correctly by alogrithm |  | Correct solution and coordinates returned | | Correct solution and coordinates returned  (PASS) |
| 4.2 | Ran example inequalities involving a minimizing LP problem through optimizing algorithm | | | | | Verify LP problems are solved correctly by alogrithm |  | Correct solution and coordinates returned | | Correct solution and coordinates returned  (PASS) |
| Overall objective: | | | | PASS: Tool is able to maximize and minimize objective functions subject to inequalities | | | | | | |
| **Objective 5: Program should be able to plot lines and inequalities on a graph accurately.**   * Inequalities drawn must be labelled. | | | | | | | | | | |
| 5.1 | Ran example inequalities through graph plotting algorithm | | | | | Verify graphs are plotted accurately |  | Lines are plotted correctly and inequalities are shaded correctly | | Lines are plotted correctly and inequalities are shaded correctly (PASS) |
| Overall objective: | | | | PASS: Tool is able to plot lines and inequalities accurately | | | | | | |
| **Objective 6: User should be able to select a topic of choice to study (under Linear Programming) or general revision to practice.**   * One out of:   + Graphing a LP problem   + Solutions using objective method   + Solutions using the vertex method. * Graphing a LP problem should be done first by user as the other sections build on what is done on the graph that is drawn. * When any of them Is chosen the corresponding screen should be displayed and questioning on the topic should begin.   + Questions on Graphing a LP problem should involve; Drawing the lines and identifying the area to shade of lines.   + Questions on Solutions using the vertex method should involve; Identifying the feasible region, finding the coordinates of these vertices, evaluating the objective function at each vertex and obtaining the optimal solution.   + Questions involving the objective solution Identifying the feasible region, finding the coordinates of these vertices, drawing the objective function, Shifting the objective function to the optimal vertex and obtaining the optimal solution. | | | | | | | | | | |
| 6.1 | Clicked on Graphing a LP problem | | | | |  |  | Correct screen shown and correct line of questioning | | Correct screen shown and correct line of questioning (PASS) |
| 6.2 | Clicked on Solutions using objective method | | | | | Verify user Is made to do graphing linear problems first |  | “Complete Graphing a LP problem first” shown | | Messagebox with”complete Graphing a LP problem first” pops up (PASS) |
| 6.3 | Clicked on Solutions using objective method after Graphing a LP problem done | | | | |  |  | Correct screen shown and correct line of questioning | | Correct screen shown and correct line of questioning (PASS) |
| 6.4 | Clicked on Solutions using vertex method before Graphing a LP problem was done | | | | | Verify user is made to do graphing LP problems first |  | “Complete Graphing a LP problem first” shown | | Messagebox with”complete Graphing a LP problem first” pops up (PASS) |
| 6.5 | Clicked on Solutions using vertex method after Graphing a LP problem done | | | | |  |  | Correct screen shown and correct line of questioning | | Correct screen shown and correct line of questioning (PASS) |
| 6.6 | Clicked on General Revision | | | | |  |  | Correct screen shown and questioning from all prior categories | | Nothing |
| Overall objective | | | | | 95% PASS: I wasn’t able to implement the general revision feature due to time constraints, but it would just have been random questions from any of the three other sections and since those are completed already this isn’t really a problem. | | | | | |
| **Objective 7: Tool should be able to test students with various formats of questions.**   * Multiple choice, fill in the blank and questions that require the user to click on a graph to answer should be generated by the tool. | | | | | | | | | | |
| 7.1 | Multiple choice questions generated by tool | | | | | Verify MCQ questions can be generated and asked by tool |  | MCQ questions generated and asked by tool | | MCQ questions generated and asked by tool (PASS) |
| 7.2 | Fill in the blank questions generated by tool | | | | | Verify fill in the blank question can be generated and asked by tool |  | Fill in the blank questions generated and asked by tool | | Fill in the blank questions generated and asked by tool  (PASS) |
| 7.3 | Click type questions generated by tool | | | | |  |  | Click type questions generated and asked by tool | | Click type questions generated and asked by tool  (PASS) |
| Overall objective | | | | | PASS: Tool is able to generate various formats of questions which are used to test students. | | | | | |
| **Objective 8: User should be taken step by step in answering the questions.**   * User should be able to submit answers when done by pressing a submit button when done answering. * User should be let known if answer is correct or wrong when done answering any question. | | | | | | | | | | |
| 8.1 | Clicked submit button | | | | | Verify user can submit answers |  | One of “Correct”, “Wrong” or “Invalid input” displayed | | One of “Correct”, “Wrong” or “Invalid input” displayed |
| 8.2 | Inputted correct answer to question | | | | | Verify user is let known if answer is correct or wrong. Verify appropriate message is displayed |  | “Correct” displayed | | “Correct” displayed |
| 8.3 | Inputted wrong answer to question | | | | | Verify user is let known if answer is correct or wrong. Verify appropriate message is displayed. |  | “Wrong” displayed | | “Wrong” displayed |
| 8.4 | Inputted invalid input | | | | | Verify appropriate message is displayed when invalid inputs are entered |  | “Invalid input!” displayed | | “Invalid input!” displayed |
| Overall objective | | | | | PASS: User is let known if submission is correct, wrong or invalid for that particular question | | | | | |
| **Objective 9 : My program should have a GUI that allows for graphs to be displayed without affecting other elements onscreen.**   * This GUI should seamlessly incorporate graph plotting functionality. * Lines and objective function drawn should be labelled on the graph. * There should be tools to aid in viewing graphs.   + A tool allowing to magnify sections, and a shifting tool allowing plots on axis to be shifted. | | | | | | | | | | |
| 9.1 | Empty plot placed on GUI | | | | | Verify graph functionality works in GUI |  | Figure of graph embedded in GUI | | Figure of graph embedded in GUI  (PASS) |
| 9.2 | Inequalities plotted on graph | | | | | Verify graph functionality works in GUI |  | Inequalities plotted on figure embedded | | Inequalities plotted on figure embedded  (PASS) |
| 9.3 | Tried zooming in and out with zoom tool | | | | | Verify zoom tool works |  | Parts selected magnified | | Parts selected magnified  (PASS) |
| 9.4 | Tried shifting plots with shift tool | | | | | Verify shifting tool works |  | Graph shifted to users will | | Graph shifted to users will  (PASS) |
| Overall objective | | | | | PASS: Graphs are able to be embedded in the GUI without any bad effects. Tools to aid viewing of graph all work properly. | | | | | |
| **Objective 10: Questions should be displayed on the GUI allowing for users to input answers in appropriate manner relevant to the question (e.g radiobuttons for Multiple choice questions)** | | | | | | | | | | |
| 10.1 | Entry boxes present for fill in the blank questions | | | | |  |  | Entry boxes present for fill in the blank questions | Entry boxes present for fill in the blank questions  PASS | |
| 10.2 | Radio buttons present for MCQ | | | | |  |  | Radio buttons present for MCQ | Radio buttons present for MCQ  PASS | |
| 10.3 | Appropriate events connected to for click type questions | | | | | Verify user clicks are registered |  | Click is registered and user input is processed | Click is registered and user input is processed  (PASS) | |
| Overall objective | | | | | PASS: User is able to submit answers to questions using appriopriate interface relevant to question format. | | | | | |

## Evidence of testing

Test 1.1-1.9

A screenshot of a computer

Description automatically generated with medium confidence

Test 2.1-2.2

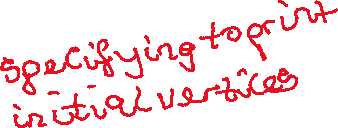
A screenshot of a computer

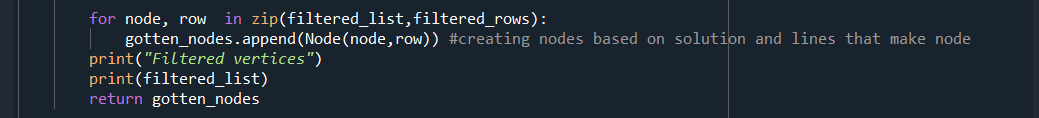
Description automatically generated with medium confidence

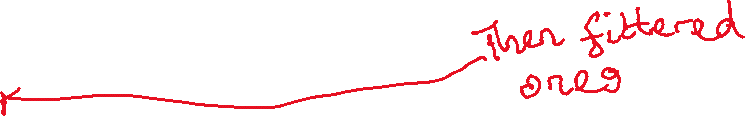
Test 2.3-2.4

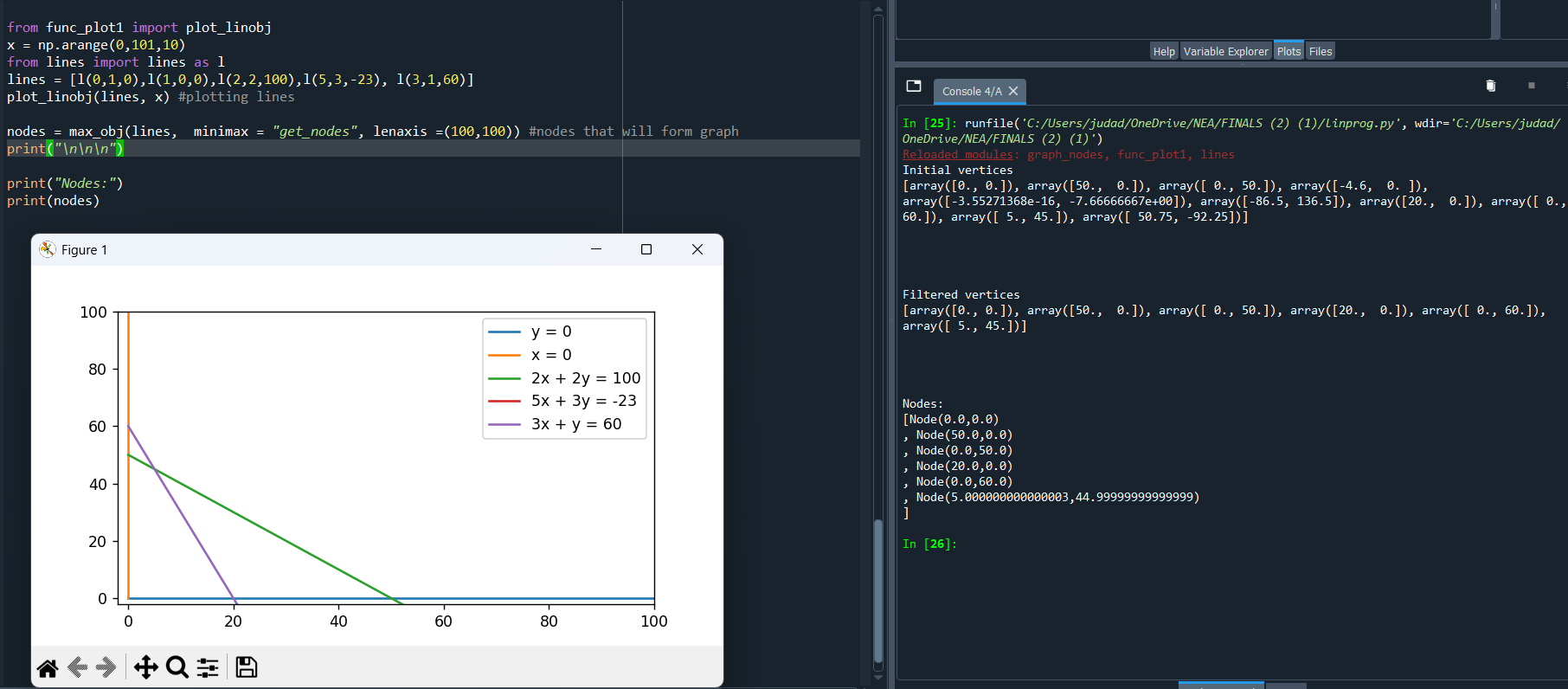
Text

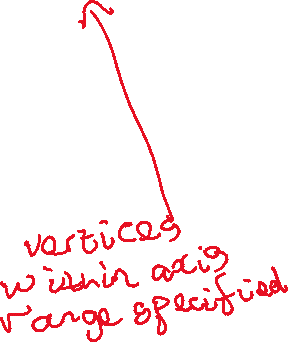
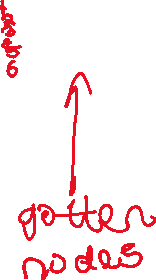
Description automatically generated







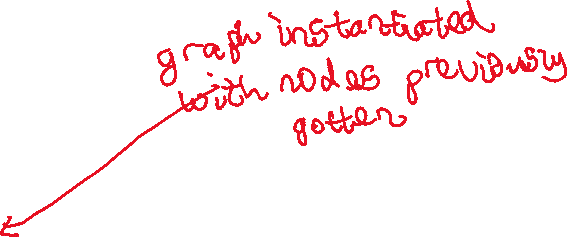
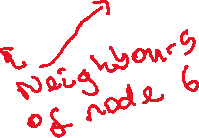




Test 2.5

A screenshot of a computer

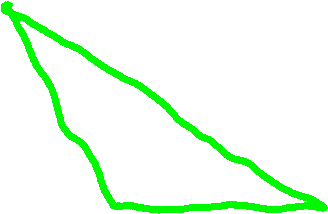
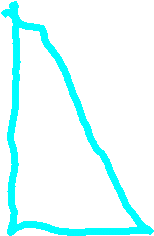
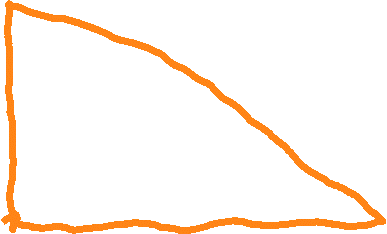
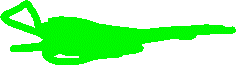
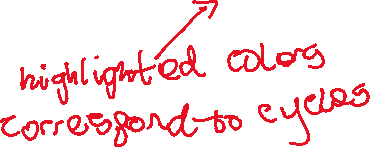
Description automatically generated with medium confidence



Test 2.6Graphical user interface

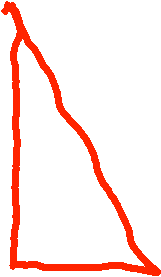
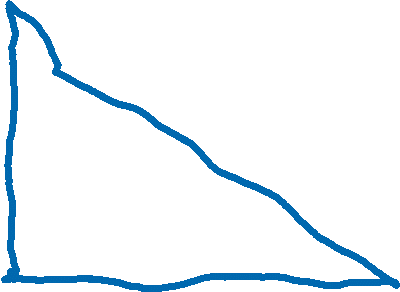
Description automatically generated Chart, line chart

Description automatically generated



Chart, line chart

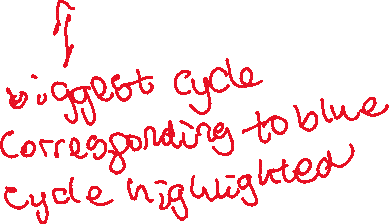
Description automatically generated



Test 2.7

A screenshot of a computer

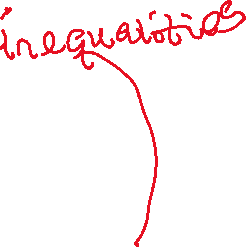
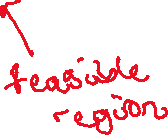
Description automatically generated with medium confidence



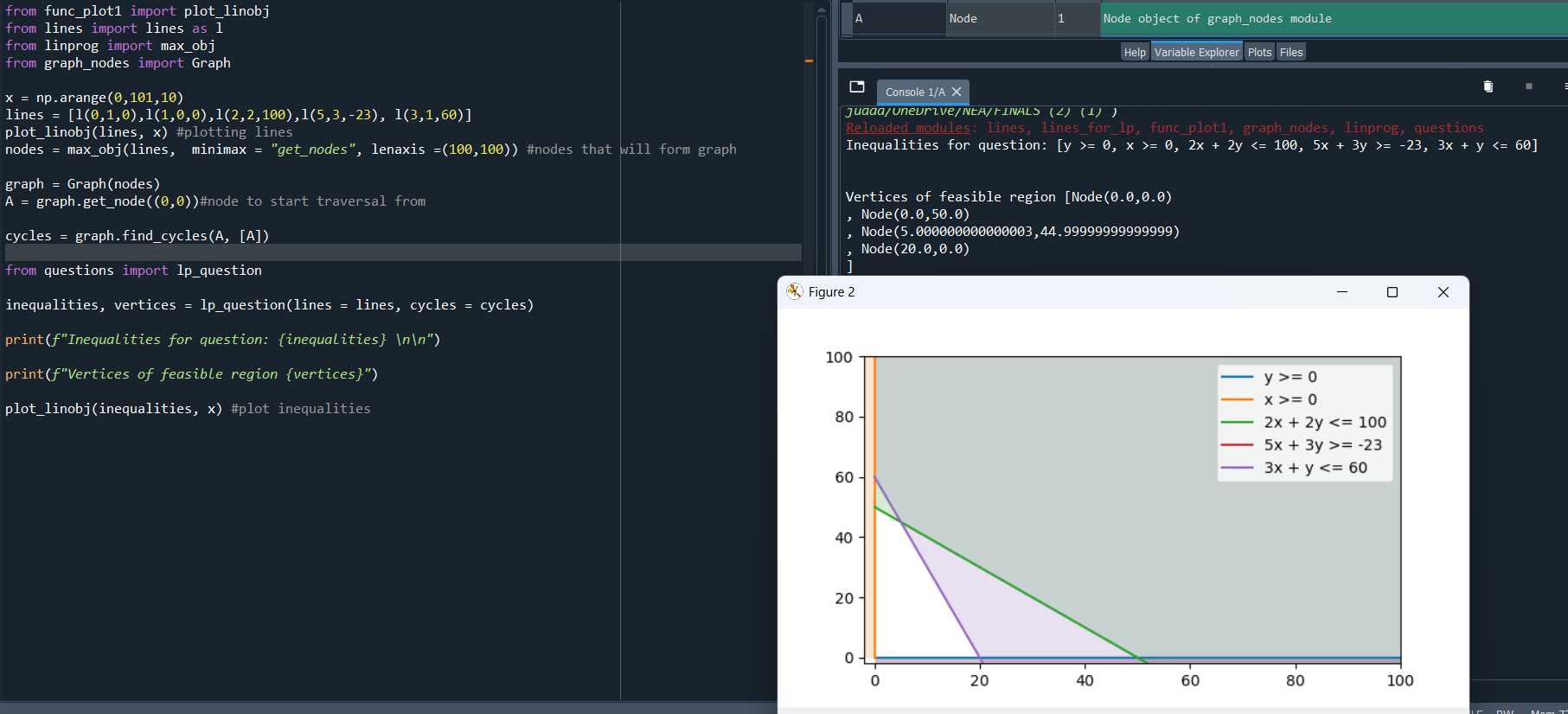
Test 2.8 -2.9

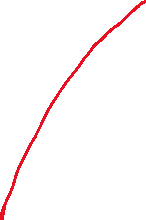
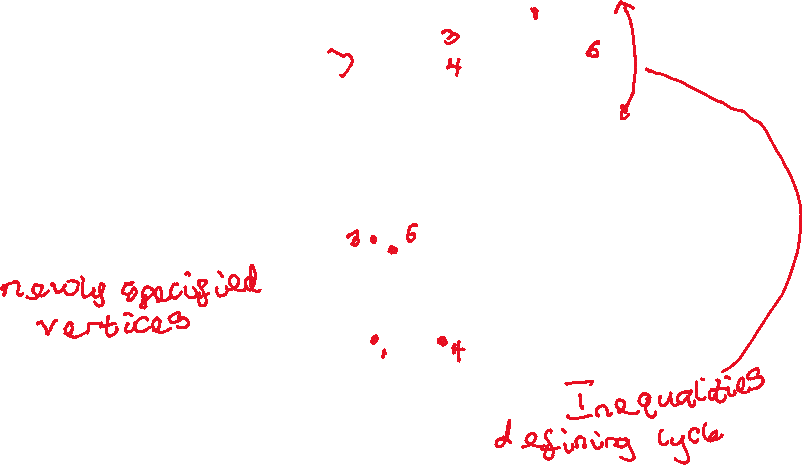
Graphical user interface

Description automatically generated



In this scenario, inequalities generated don’t define biggest cycle found. In this scenario this is actually the best that can be done as the biggest cycle can actually never be the feasible region. However my solution uses the average of the nodes in the cycle to make inequalities allowing for a cycle to still be define regardless. If this were not the scenario ie the cycles can actually form a feasible region, inequalities generated will define cycle as I show below with specifying the vertices of another cycle in figure above.





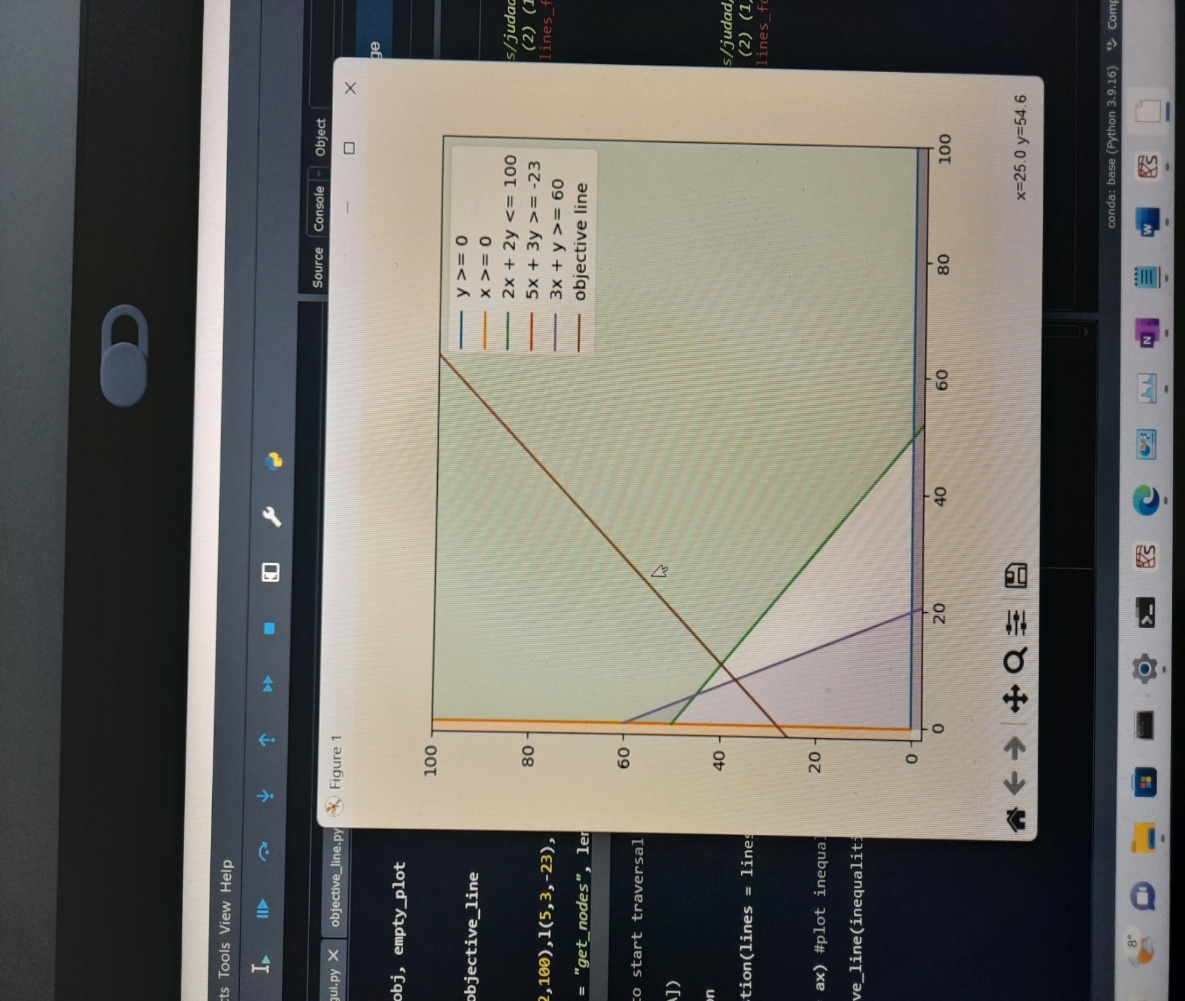
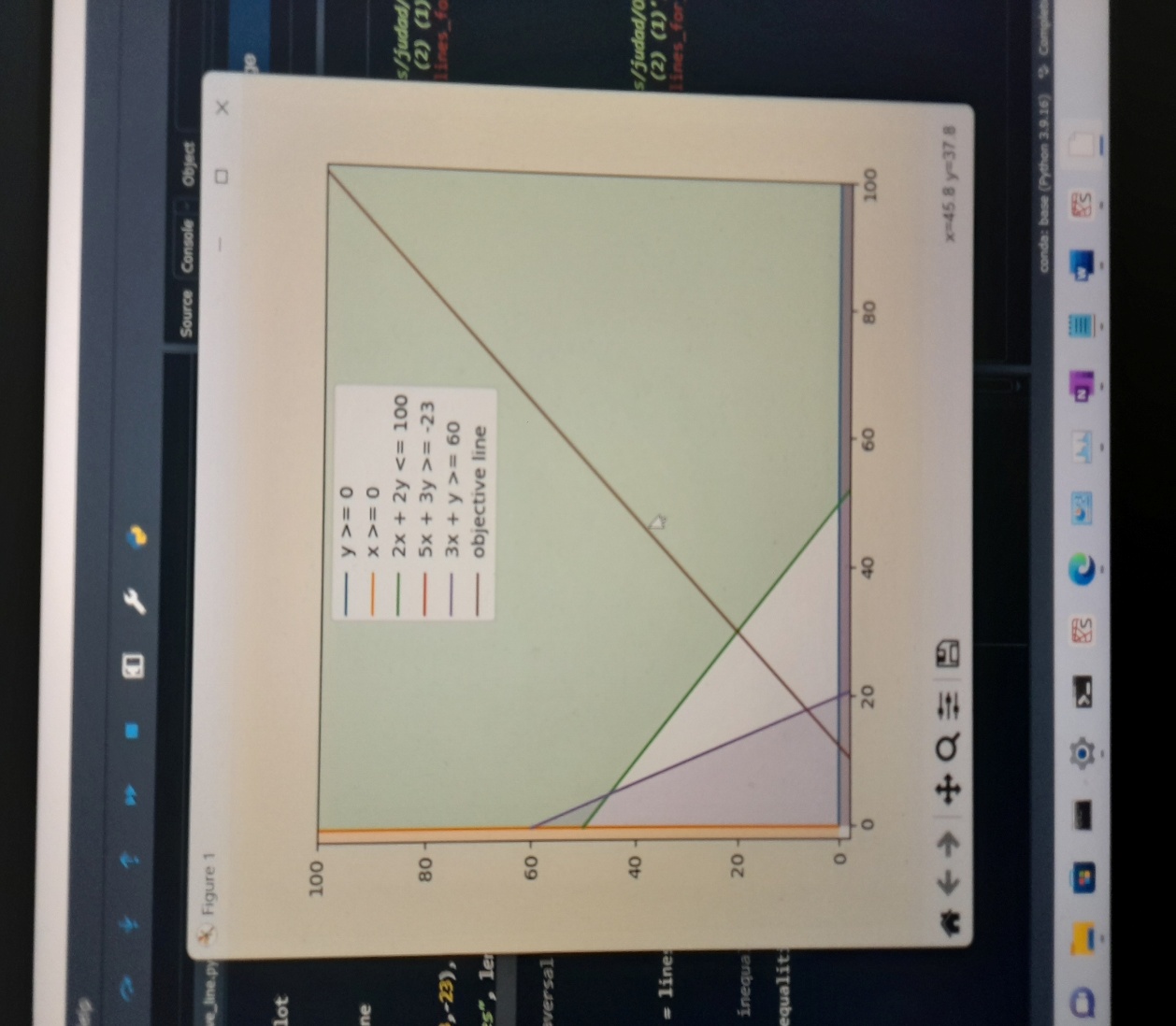
As such only in instances when feasible region isn’t possible do the inequalities not define the biggest cycle. However, they will still form a cycle and will be able to be used in questions. So this is still a pass.

Test 3.1-3.2

Graphical user interface

Description automatically generated

Test 3.3





Test 3.4

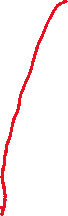
Graphical user interface

Description automatically generated with low confidence



Graphical user interface

Description automatically generated



Test 4.1

Example 1

Text, letter

Description automatically generated



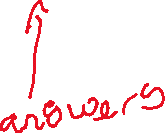
Example from integral maths





A screenshot of a computer

Description automatically generated with medium confidence



<https://my.integralmaths.org/pluginfile.php/121331/mod_resource/content/1/aqafmasdl1n.pdf> (link to question)



Text

Description automatically generated

Example from integral. https://my.integralmaths.org/pluginfile.php/121327/mod\_resource/content/2/aqafmasdl1ax\_level1.pdf



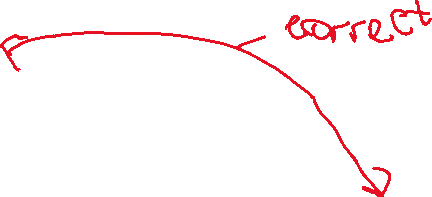
Text, letter

Description automatically generated



Graphical user interface, application

Description automatically generated



Test 4.2

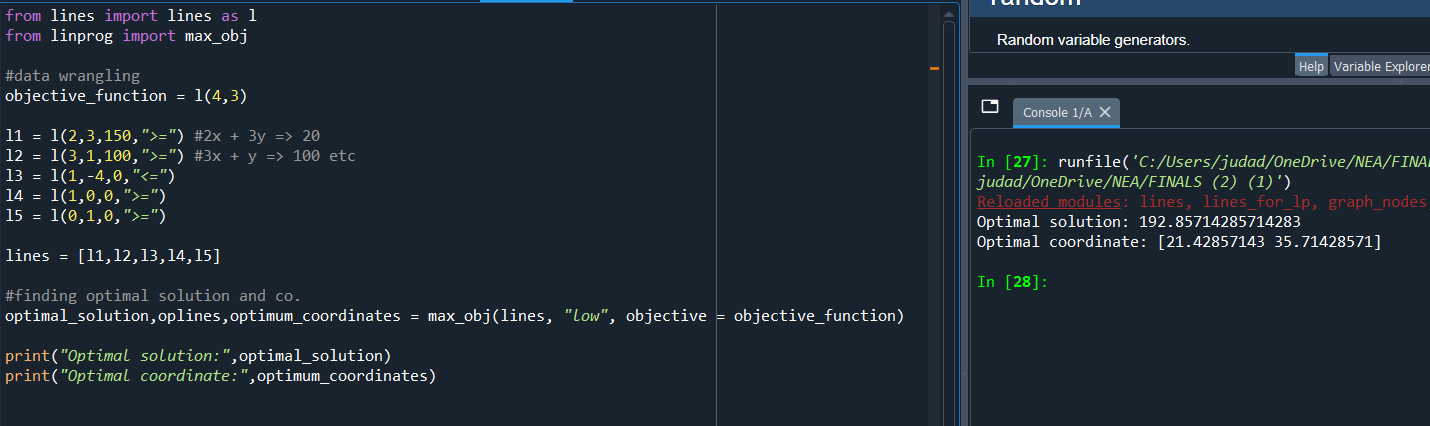
Example 1

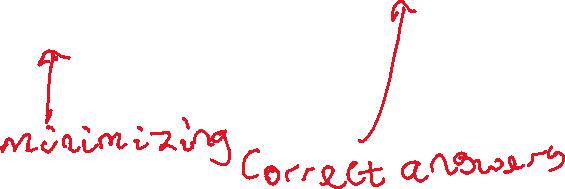
Text, letter

Description automatically generated

Example from integral. <https://my.integralmaths.org/pluginfile.php/121331/mod_resource/content/1/aqafmasdl1n.pdf> (link to question)







Example 2

Text, letter

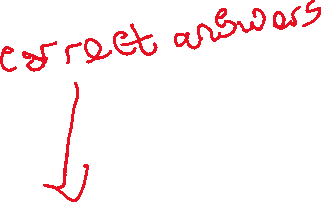
Description automatically generated

https://my.integralmaths.org/pluginfile.php/121327/mod\_resource/content/2/aqafmasdl1ax\_level1.pdf



Text, letter

Description automatically generated



Graphical user interface, text, application, chat or text message

Description automatically generated

Test 5.1

Text

Description automatically generated

Chart, line chart

Description automatically generated

Example drawn inequalities from integral

Chart, line chart

Description automatically generated



Text

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated



Test 6.1

Before Graphing a LP problem clicked

Graphical user interface, text, application, chat or text message

Description automatically generated

After clicked

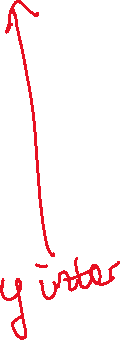
Text

Description automatically generated

Correct line of questioning

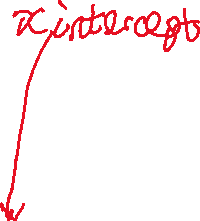
Chart

Description automatically generated



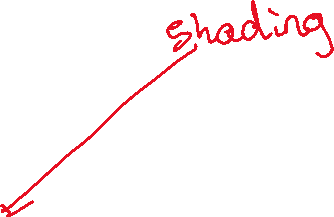
Chart

Description automatically generated



Chart, line chart

Description automatically generated



Test 6.2

Before clicking

Graphical user interface, text, application

Description automatically generated

After clicking

Graphical user interface

Description automatically generated with medium confidence

Test 6.3

Before clicking

Graphical user interface, text, application

Description automatically generated

After clicking

Chart

Description automatically generated

Correct line of questioning begins

Chart

Description automatically generated



Chart, line chart

Description automatically generated



Chart

Description automatically generated



Chart, line chart

Description automatically generated



Test 6.4

Before clicking

Graphical user interface, text, application

Description automatically generated

After clicking

Graphical user interface

Description automatically generated with medium confidence

Test 6.5

Before clicking

Graphical user interface, text, application

Description automatically generated

After clicking

Chart

Description automatically generated

Correct line of questioning begins

Chart

Description automatically generated



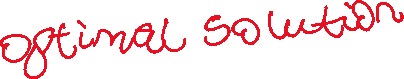
Chart

Description automatically generated



Chart

Description automatically generated



Test 7.1-7.3 & 10.1-10.2

MCQ, fill in the blank and click type questions all generated and asked showing appropriate entry methods for user input

Chart, box and whisker chart

Description automatically generated

Chart, box and whisker chart

Description automatically generated

Chart, line chart

Description automatically generated

Test 8.1

Before clicking Submit

Chart, box and whisker chart

Description automatically generated

After selecting option and clicking submit

Chart, box and whisker chart

Description automatically generated

Test 8.2

Before inputting answer

Chart, box and whisker chart

Description automatically generated

After inputting correct answer and submitting

Chart

Description automatically generated

Ex 2

Before inputting answer

Chart, line chart

Description automatically generated

After submitting correct answer

Chart, line chart

Description automatically generated

Test 8.3

Before inputting answer

Chart, box and whisker chart

Description automatically generated

After inputting wrong answer and submit

Chart

Description automatically generated

Ex2

Before inputting answer

Chart, line chart

Description automatically generated

After inputting wrong answer and submit

Chart, line chart

Description automatically generated

Test 8.4

Before submit Chart, box and whisker chart

Description automatically generated

After submitting invalid input

Chart, box and whisker chart

Description automatically generated

Chart, box and whisker chart

Description automatically generated

Test 9.1

Empty plot embedded in Gui

Chart

Description automatically generated

Test 9.2

Inequalities plotted on graph in gui

Chart

Description automatically generated

Test 9.3

Before magnification of this part

Chart

Description automatically generated

After magnification

Chart

Description automatically generated

Test 9.4

Before shifting

Chart

Description automatically generated

After shifting to the side

Chart

Description automatically generated

Test 10.3

Before clicking on graph

Chart, line chart

Description automatically generated

After clicking on graph – wrong click

Chart, line chart

Description automatically generated

Correct click

Chart

Description automatically generated

# 

# Evaluation

## Objective Results - Self Assessment

Though briefly highlighted in the testing section, I give an overview of the extent to which objectives have been met .

**Objective 1:Tool should be able to Generate lines and inequalities.**

* Lines generated should contain values in a specific range eg lines could have a maximum x being 100 and as such beyond that the line should no have no other value.
* Lines should be displayed properly in the form ay + bx = c
* Lines that won’t be adequately represented with conventional format should be catered to.
* Inequalities should be represented the same as lines but with appropriate inequality symbol ie one out of; >=, <= , > , <.

**Actual outcome Description:**

My tool can generate lines and these lines are adequately represented throughout the entire system. Inequalities though aren’t generated as was initially thought in order to satisfy objective 2, however they are also adequately represented throughout the system.

**Overall:** Objective fully met.

**Objective 2: Inequalities used in questions must form a closed cycle.**

* + When inequalities are drawn on graph there should be at least one cycle.
  + If more than one cycle, cycle with highest number of edges should be chosen for the question.

**Actual outcome Description:**

I implement graphs and run a cycle detection algorithm to get the cycles on it and then choose the one with the highest number of nodes, essentially working backwards to get the question. However, this fulfils my objectives.

**Overall: Objective fully met.**

**Objective 3: Tool should be able to make an Objective function (function that would be optimised subject to constraints)**

* Objective function should be draggable when drawn on graph.
* Program should be able to “decipher” when objective function is at the optimal vertex to linear programming problem ie there should be some way to know when objective function is at optimal vertex.

**Actual Outcome Description:**

I am able to use event handling to get when the user clicks on the objective function and shift it according to mouse movements. With maths and help from the Numpy library I am able to extrapolate when the objective function comes in contact with the optimal vertex.

**Overall: Objective fully met.**

**Objective 4: Program should be able to minimize or maximize an objective function subject to constraints (inequalities), returning optimal solution and coordinate at which solution occurs.**

**Actual Outcome Description:**

Algorithm to minimize and maximize an objective function subject to inequalities works properly returning the optimal solution

**Overall: Objective fully met.**

**Objective 5: Program should be able to plot inequalities on a graph accurately.**

* Inequalities and lines drawn must be labelled.

**Actual Outcome Description**

Lines and inequalities are drawn on matplotlib figures and are labelled.

**Overall: Objective fully met.**

**Objective 6: User should be able to select a topic of choice to study (under Linear Programming) or general revision to practice.**

* One out of:
  + Graphing a LP problem
  + Solutions using objective method
  + Solutions using the vertex method.
* Graphing a LP problem should be done first by user as the other sections build on what is done on the graph that is drawn.
* When any of them Is chosen the corresponding screen should be displayed and questioning on the topic should begin.
  + Questions on Graphing a LP problem should involve; Drawing the lines (identifying the x and y intercepts of lines) and identifying the area to shade of lines.
  + Questions on Solutions using the vertex method should involve; Identifying the feasible region, finding the coordinates of these vertices and obtaining the optimal solution.
  + Questions involving the objective solution Identifying the feasible region, finding the coordinates of these vertices, drawing the objective function, Shifting the objective function to the optimal vertex and obtaining the optimal vertex.

**Actual Outcome Description:**

User is able to choose one of the 3 subtopics to practice and the corresponding line of questioning begins. However there wasn’t enough time to implement the “General Revision” Section.

**Overall : Objective partially met**

**Objective 7: Tool should be able to test students with various formats of questions.**

* Multiple choice, fill in the blank and questions that require the user to click on a graph to answer should be generated by the tool.

**Actual Outcome Description:**

Various formats of questions for each question are generated for each line of questioning where possible.

**Overall: Objective fully met.**

**Objective 8: User should be taken step by step in answering the questions.**

* Questions should be broken into smaller parts that will be solved to allow students work through questions logically and to easily trace sources of student error ie final answer to LP problem shouldn’t be inputted immediately, all other aspects should be done first.
* User should be able to submit answers when done by pressing a submit button when done answering.
* User should be let known if answer is correct or wrong when done answering any question.

**Actual Outcome Description:**

When done answering questions, user is able to submit questions and a corresponding message based on input is displayed. This message is either “Correct”, “Wrong”, or “Invalid input”.

**Overall: Objective fully met.**

**Objective 9: My program should have a GUI that allows for graphs to be displayed without affecting other elements onscreen.**

* This GUI should seamlessly incorporate graph plotting functionality.
* Lines and objective function drawn should be labelled on the graph.
* There should be tools to aid in viewing graphs.
  + A zoom tool allowing to magnify sections, and a shifting tool allowing plots on axis to be shifted.

**Actual Outcome Description**

A figure containing plots is “placed” on the GUI. The zoom and shift tool are directly below to aid viewing as some times feasible region might be small.

**Overall: Objective fully met**

**Objective 10: Questions should be displayed on the GUI allowing for users to input answers in appropriate manner relevant to the question (e.g radiobuttons for Multiple choice questions)**

**Actual Outcome Description**

There are blank entry boxes when fill in the blank questions are asked, radio buttons for MCQ.

**Overall: Objective fully met.**

## End user feedback

Once the end user had experimented with the program, I asked him the following questions to get his feedback on it.

* How easy did you find the GUI to use, and were there any problems with how it was displayed?

*I’m pleasantly surprised by everything about the GUI. From the colour scheme to the font size and the buttons, it all seemed quite modern. The fact that buttons have hover colors is also a nice touch. The buttons did as they were supposed to and the instant feedback when questions were answered is great. My only itch will be to have a quit or home button. Some of the question answering processes are a bit long (not that this is a problem) but a student might feel like they have tested themselves enough already and won’t want to finish the whole question before being able to do another section.*

* What do you think of the graph plotting functionality?

*The accuracy and utility of the graphs is incredible. In terms of inequality plotting your tool easily rivals many of the graphing software currently used, though obviously they have a lot more utility. The proper labelling of the lines allowed for way easier cross checking of work.*

* After using the program for a while would you say it adequately tested your knowledge of the basics of linear programming and would you suggest it is suitable for students’ use?

*Absolutely. You thoroughly did your research and included all that students need to know regarding those subtopics. I believe students will prefer to use this over any others currently available to revise linear programming. Would be lovely if you could extend the number of topics in it.*

* Were there any other issues with the software that could be improved on?

*Issues performance wise- none. It didn’t crash or run into any sort of errors.*

* Is there anything you feel is missing from the tool?

*To be honest it is really well done. Regarding the subtopics you covered, there is nothing off the top of my head comes to mind that needs adding. As was initially planned to aid teachers, id say teachers being able to monitor students in some way would be ideal, however if I remember correctly, you were considering this as an extension objective.*

## Conclusion and Future Improvements

After testing my solution extensively, only one subobjective wasn’t met (General Practice). However, this subobjective not being completed doesn’t have any detrimental effects to my solution. Given more time, it would have been easily implemented as it would just necessitate selecting questions from each of the other sections.

The main things I would do with more time is include a database, allowing for teachers to have classes and to monitor the progress of students. Unfortunately, the time wasn’t sufficient to design or implement such.

The end user commented about adding in more topics. Adding in a framework for teachers to set their own questions will be a good improvement and with more time would be doable. However, from his comment im thinking he means to add in question generation for each topic. To do so will be complex and time consuming though will probably be doable.

Also, with more time I would have added more in-depth error checking to give users more insight as to the kind of errors they made while working.

In conclusion, I believe this project was successful because it met almost all objectives, the one main objective that wasn’t met could easily be implemented with more time. The end user is happy with the software and has stated that it would be useful to him and his students when revising linear programming.