# Linear-Equations-Solver

This Python project aims to build a linear equation solver program. Users are requested to input 4 variables that define 2 linear equations in slope-intercept form (y = ax + b) and configures the solver's step-size and range (span) before having the solver to determine if the delta-y value converges within the given span (range) defined by the user.

This Python program is written in Python 3.

Problem statement:

Given 2 linear equations, in slope-intercept form, which are:

y = ax + b and

y = cx + d

respectively,

Find the solution that satisfies both equation, i.e. find the point of intersection of both lines.

How does the solver work? (Algorithm):

(In the example that follows, we will use the 2 linear equations y = 2x + 3 and y = x respectively.)



Imagine you have 2 straight lines in a Cartesian coordinate system. You start off with an arbitrary x-value, which we call x0. This gives you two corresponding y-values for the 2 equations (lines). Let's call them y1 and y2 respectively. Hence, y1 = a\*x0 + b and y2 = c\*x0 + d.

Note for Python Programming:

Before beginning to solve the optimization problem in the code,

Define 6 variables called PE, PL, PIS, PDS, PI and PD for later usage. Assign each of them a value equivalent to a blank string, ''.

PE is a variable that determines if the solution is x0, the initiated x-value itself (= True) or otherwise (= False).

PL is a variable that determines if the lines are parallel to each other (= True) or otherwise (= False).

PI is a variable that determines if the dy(see below for info on dy) is decreasing/converging for increasing x-values starting from x0 (= True) or otherwise (= False) BUT in both cases, a solution is NOT found.

PIS functions the same as above (PI) but in the case where PIS == True, a solution is FOUND while x0 <= x <= (x0 + sp). (It also converges.)

PD is a variable that determines if the dy(see below for info on dy) is decreasing/converging for decreasing x-values starting from x0 (= True) or otherwise (= False) BUT in both cases, a solution is NOT found.

PDS functions the same as above (PD) but in the case where PDS == True, a solution is FOUND while (x0 - sp) <= x <= x0. (It also converges.)



If x0 starts here, we get y1 = y2.

Now, if y1 = y2, we know that the initial x-value is indeed the solution itself, therefore the solution is x = x0. PE = True.



However, if x0 starts here, we get a y1 which is also different from y2.

If x0 starts here, we get a y1 which is different from y2.

If say, y1 is different from y2, we know the solution must be either at an x-value greater than x0, i.e. x > x0 or an x-value less than x0, i.e. x < x0. Or there could be no solution at all (if the lines are parallel).

So, to check whether the solution is at x > x0 or x < x0, we split our program into 2 sections, by first checking if the solution (if exists) is at x > x0 and only then, look for the solution (if exists) at x < x0.

Let's start with x > xo,

We need another x value to compare the corresponding y-values with our previous y-values and to check their difference.

So, let's increase the x-value by a certain amount or delta-x, dx, where dx is defined by the user. dx is called the step size.

The new x is therefore x = x0 + dx. Then, we calculate the new corresponding y-values, y1 and y2.



x0

Legend:

Small span, dy converges but no solution found

Large span, dy converges and solution exists

…delta y, dy, reduces, until it becomes 0.

As x increases by dx each loop/iteration…

Now, if the difference in the new ys, delta-y, i.e. abs(y2 -y1) or the absolute value of (y2 - y1), is LESS than the previous value of delta-y, i.e. abs(y02 - y01), that means the delta-y is decreasing and the delta-y values are converging, i.e. the 2 lines are getting closer and closer together as x increases from x0. We then check if the subsequent iterations of x, where x = x + dx, will result in further convergence, until delta-y is zero. If delta-y is zero, the corresponding x-value would be the solution, i.e. PIS = True. Otherwise, conclude that though dy is decreasing/converging for increasing values of x from x0, there is no solution found, i.e. PI = True. (We shall come back to this case later. Hint: too large a step-size!)

Take note that there needs to be a limit in the number of iterations in which the program will run (otherwise it will run forever!), and it is defined by the user through a value called 'span'. Span or the variable, sp as, defined in the code, is the maximum increase or decrease in x, or delta-x, such that the iterations will only be conducted if (x0 - sp) <= x <= (x0 + sp). That is, span defines the boundary of the x-values which we intend to look for convergence/divergence and solution.



Legend:

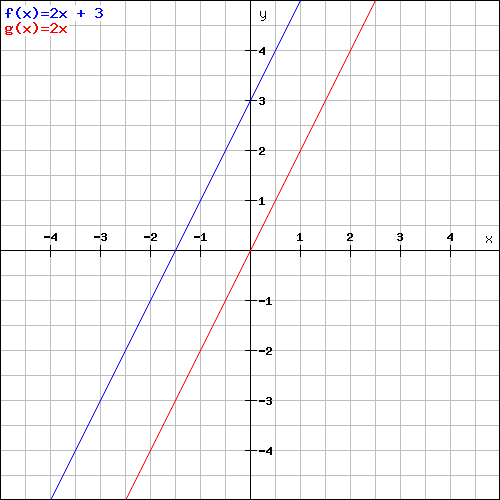
No matter how large or small dx is, dy diverges as x increases from x0. (Arrow does not represent magnitude of dx, it represents magnitude of span!)

x0

As x increases by dx each loop/iteration…

dy diverges with x.

However, if the new delta-y, i.e. abs(y2 - y1), is GREATER than the previous delta y, i.e. abs(y02 - y01), that means the delta-y values are diverging. The 2 lines are getting further and further away as x increases from x0. Preliminary conclusion is therefore: dy is increasing/diverging as x increases from x0. PI = False.



No matter how large or small dx is, no matter which x0 you begin with and no matter what value x is assigned per loop/iteration, dy is a constant.

What happens if the new delta y is equal to the old delta-y, i.e. abs(y2 - y1) == abs(y02 - y01)? We check the new delta-y values for subsequent iterations of x. If delta-y value is the SAME regardless of x-value in the range of x defined by the user (defined by span), we make one further check before concluding that the 2 lines are parallel. We check that a, the gradient of the first line is equal to c, the gradient of the second line. If a == c, that means the 2 lines are parallel, i.e. PL = True. Otherwise, we conclude that there is a ‘bug’ in the program.

After looking for delta-y convergence in the region x0 <= x <= (x0 + sp), we look for the other region, i.e. (x0 - sp) <= x <= x0.



Legend:

Small span, dy converges but no solution found

Large span, dy converges and solution exists

x0

The same rule applies, except in the opposite sense. If subsequent delta-y values decrease, look for the case delta-y = 0. If exists, PDS = True. Otherwise, conclude that dy is decreasing/converging for decreasing x-values smaller than x0 (BUT bigger than x0 - sp) or PD = True.

If subsequent delta-y values increase, conclude that dy is increasing/diverging for decreasing x-values smaller than x0 (BUT bigger than x0 - sp) or PD = False. (This is not shown in the above graph by arrows. But one can see this by referring to the case where x < -3 in the graph above, notice that the delta-y value diverges for decreasing x-values for x0 beginning at x0 < -3.)

Now that we are done there is one last possibility assuming no programming errors. What happens if the step size chosen is too large, so large that the solver misses the solution altogether?



3rd iteration/loop

4th iteration/loop

Small step-size (dx), dy converges to 0 within 3 loops/iterations (assume scale of dx similar to graph).

Large step-size (dx), dy converges while x < -3 and diverges while x > -3. Solution missed on 2nd loop/iteration.

Very large step-size (dx), dy diverges for 1st and subsequent loops/iterations.

x0

Recall that step-size is a change in delta-x values. In each iteration, the x-value is either incremented or decremented by a step-size. If the solution happens to occur at a value between an x-value and an (x + dx) value for x0 <= x <= (x0 + sp) or between an x-value at a value between an x-value and an (x - dx) value for (xo - sp) <= x <= x0, that means the solver will see that the dy value reduces and then suddenly increases or vice versa, without producing a solution.

This hints at too large a step-size. And this will only occur when PI = False and PD = False.

Notice that for the case where the solution is an integer, a small step-size like 1 is usually the most suitable value. Referring to the graph above, a small step-size of 0.5 ensures the dy reduces down to 0 within 4 loops/iterations for x0 = -5:

\*1st iteration: x0 loops to itself

\*2nd iteration: x0 loops to x0 + dx, i.e. (-4.5 loops to -4)

\*3rd iteration: x0 loops to x0 + dx + dx, i.e. (-4 loops to -3.5)

\*4th iteration: x0 loops to x0 + dx + dx + dx, i.e. (-3.5 loops to -3)

whereas if the step size is larger than 0.5 but between 1 and 2, the solution will be missed altogether and one will obtain a result in which the dy seems to reduce at first before slowly diverging after several iterations.

The mathematical requirement for a solution not to be missed due to too large or incorrect (non-whole number ratio of solution) is

It is not possible to predict the solution in first place, but trial and errors should get you the right dx value in a short time. As a rule of thumb,

Set dx = 1 for solutions with integers

Set dx = a fraction for others beginning ½, moving on with 1/3, ¼, 1/5, etc.

End result:

PE == True: The solution is x = xo.

PL == True: The 2 lines are parallel and they never intersect each other in Euclidean space.

(PIS == True) and (PD == False): Solution converges for increasing x-values. The solution is x = (the value of x found).

(PDS == True) and (PI == False): Solution converges for decreasing x-values. The solution is x = (the value of x found).

(PI == True) and (PD == False): No solution is found, delta y converges with increasing values of x.

(PI == False) and (PD == True): No solution is found, delta y converges with decreasing values of x.

(PI == False) and (PD == False): There is a solution in between (x0 - sp) and (xo + sp), step size is incorrect, and the solver miss the solution altogether.

Have a look at the codes to see how the program is executed.

Some further notes:

This program has some built-in restrictions for the users. For example, the 2 gradient values can never be 0 at the same time or there will be no linear equation(s) to be solved! The user will be punished by an infinite loop if she fails to comply to this simple rule for the 2nd time. You have NO 3rd chance!

When you are asked to enter yes or no, please follow the instructions. Or risk another infinite loop! You make the choice.

Also, please take note that the program does not check for inputs of values other than numerical values. Inputs of non-numerical values will result in type-conversion errors and the program will exit immediately.

Lastly, this program has a built-in 'restarter'. At the end of the very last execution, the user wil be given a choice to continue trying the program with other values or to terminate and exit the program.

Acknowledgement:

Special thanks to tl8roy (Tristian) and Face (Friedrich) from Orbiter Forum for suggesting me to take up Python. It is a very nice language!

Also thank you Ironrain (Derk) for helping me to check the 500 lines of codes, not to mention suggesting me to start a GitHub account and share my codes with the rest! Thank you, Derk.