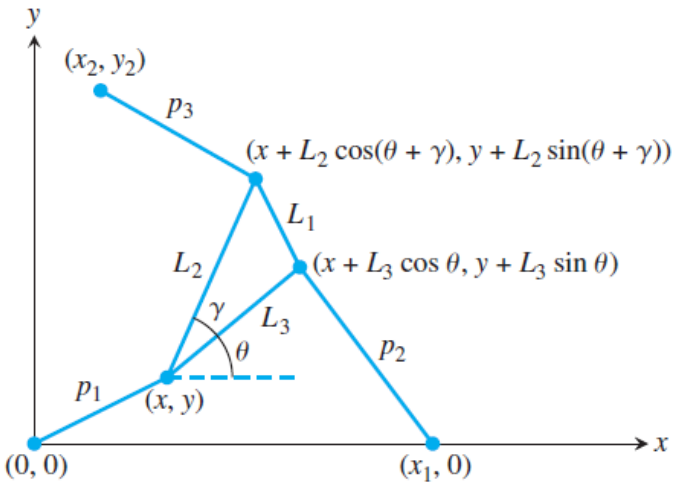
Project 1: Kinematics of the Stewart Platform.

Group1: Radek Jan Holik/ Sheng-Han Yueh/ Keesara Sharath Kumar Reddy

1. **Introduction**

   We want to find the possible rotating angles in a 2D plan with three fixed struts (as the following figure). In the problem, we have to find the values for θ, x, and y. We used the equation f(θ)=0 to find all possible roots, θ. When we know the θ, we can find x and y.



2. **Methodology**

   2.1 Define the function

   In order to change the multiple combinations of parameters, we defined a class that includes setParamenters() and f() functions. We used setParameters() to define the constant values in f(). The function f() will have only one argument, θ. By finding the roots in equation f(θ)=0, we can find possible rotating angles. Therefore, we returned f(θ) and the x-y coordinate values.

   2.2 Define the starting intervals and starting points

   From $-\pi$ to π we tested several intervals. For 6 intervals, the bisection method found only two roots, since the other four intervals all have the same sign at the endpoints. To find all roots, we evenly divided 12 or 24 intervals between $-\pi$ to π as the initial intervals in the bisection method. We assumed that the smaller the intervals we have, the greater the probability of finding all the roots. For the starting points of Newton's method, we used the midpoints of the 12 intervals.

   2.3 Bisection method

   By Bolzano's theorem, if the function is continuous between [a, b] and $f(a)$ and $f(b)$ have different signs, it has at least one root between a and b. In our case, a and b were the endpoints of the 12 intervals and $f(\theta)$ was all continuous in 12 intervals. First, we checked if $f(a)\times f(b)<0$. Only if this criterion is met, the iteration starts. We set the $TOL=0.5\times10^{-8}$, since we wanted the root to be correct with 8 decimal places. From the definition of p decimal places (as the following calculation), we required 27 iterations. Therefore, we had the maximum iteration tolerance is 30 (1).

$$\frac{1}{2^{n+1}} < 0.5 \times 10^{-8}$$

$$n > \frac{8}{log_{10}2} \approx 26.6$$

We set the following criteria to stop the iteration:
a. the root is found when $f(\theta)\approx 0$
b. the root is found when $(b-a)/2 < TOL$
c. the root is not found when the iteration exceeds the maximum tolerance.
If the root was found successfully, we returned the bisection process table, the final root, and the iteration number (2).

2.4 Newton's method

Before developing Newton's method, we defined the derivative function $numDer()$ (3,4). We used the central difference to estimate the result of the derivative. The derivative function was correct because $f'(\theta)$ was equal to 0 at three critical points of $f(\theta)$ (as 3.4.2 in .ipynb file).In Newton's method, we set the following criteria to terminate the iteration:
a. the root is found when $f(\theta)\approx 0$
b. the root is found when $|\theta i+1-\theta i|<TOL$
c. the root is not found when the iteration exceeds the maximum tolerance.
We set the $TOL=10^{-8}$ since we wanted the root to be correct with 8 decimal places and maximum tolerance of iteration was 30 as we expected it cost fewer iteration steps than the bisection method when converged. If the root was found successfully, we returned the table of the iteration process, the final root, and the iteration number (2).

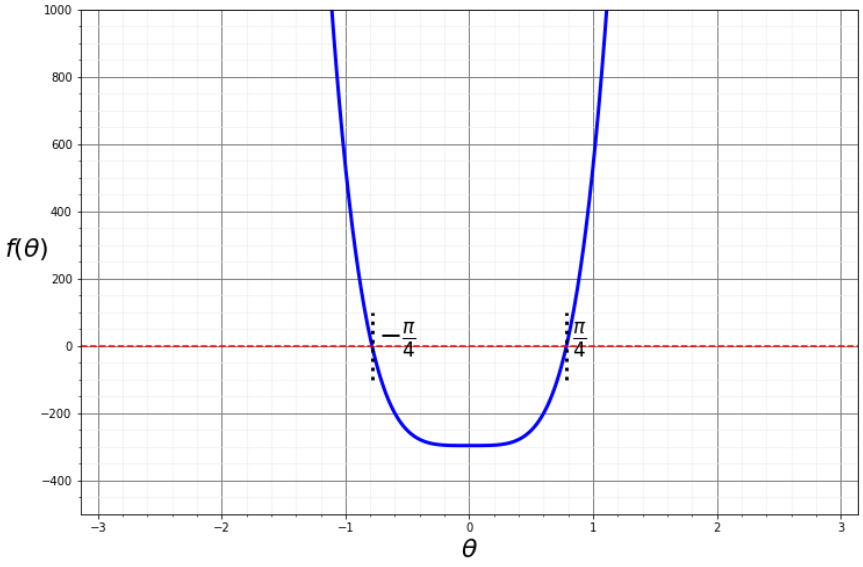
3. Computer Experiments/Simulations and Results

3.1 Define the function and test (Activity 1)

We already knew in the parameters L1=2, L2=L3=√2, γ=π/2, p1=p2=p3=√5, x1=4, and (x2, y2) =(0, 4), the roots, θ, would be -π/4 and π/4. We expected f(-π/4) and f(π/4) equal to zero if the function f(θ) is correct. The result demonstrated that both f(-π/4) and f(π/4) approximate 0. Therefore, f(θ) is correct (as 1.3 in .ipynb file).

3.2 Visualization of the function $f(\theta)$ (Activity 2)

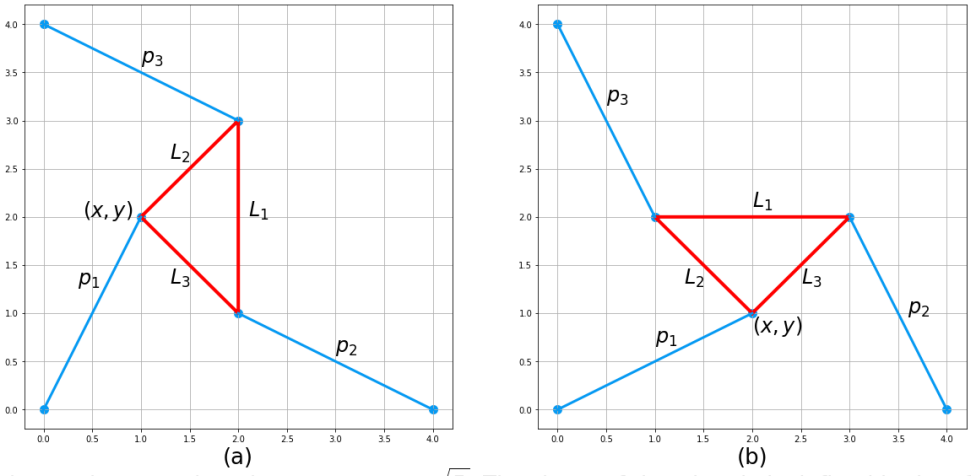
In the parameters L1=2, L2=L3=√2, $\gamma=\pi/2$, p1=p2=p3=√5, x1=4, and (x2, y2) =(0, 4), we plotted the function f(θ) as a blue line, where the f(θ) equals to 0 as a red dashed line, and where θ equals to -π/4 and π/4 as black dashed line (as the following figure) (5). The plot showed that function f(θ) intersects with the red dashed line, demonstrating there are two roots at -π/4 and π/4 (as 2.1 in .ipynb file).



3.3 Visualization of two poses (Activity 3)

With the parameters L1=2, L2=L3=√2, γ=π/2, p1=p2=p3=√5, x1=4, and (x2, y2) =(0, 4), there are two possible poses with θ equals to -π/4 (as the left figure) and π/4 (as the right figure). The (x, y) equals (1, 2) and (2, 1) individually (6).

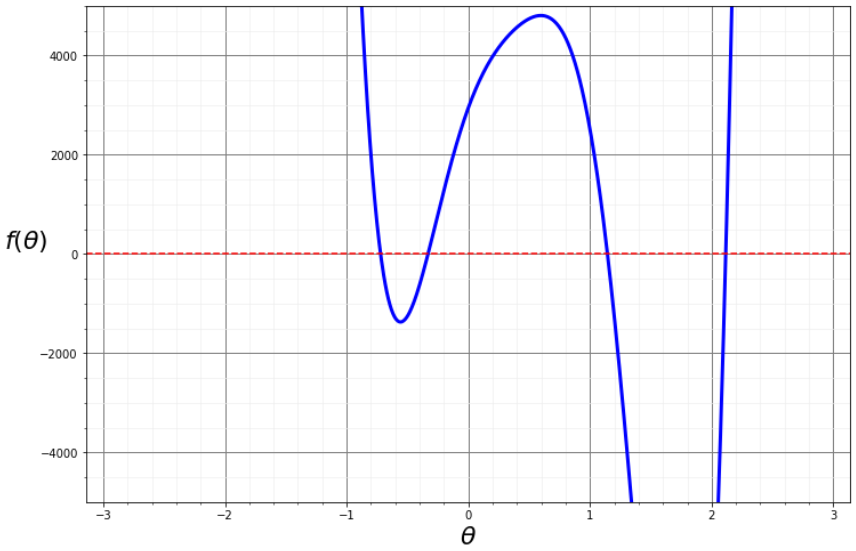
Two poses of the planar Stewart platform with identical arm lengths.



Each pose has strut lengths $p_1 = p_2 = p_3 = \sqrt{5}$. The shape of the triangle is defined by $L_1 = 2$, $L_2 = L_3 = \sqrt{2}$, $\gamma = \pi/2$.

3.4 Finding poses for given parameters (Activity 4)

From -π to π, the figure in 3.2 in .ipynb files showed there are four points where the function f(θ) equals 0 (as the following figure), demonstrating there are four roots in the new parameter where two roots locate [-1, 0] and the other two roots locate [1, 2.5].



3.4.1 Which equation solver did you use, and why?

We used the bisection method and Newton's method as the solvers. For the fixed point method, we made $g(\theta)$=$f(\theta)$+θ and solved the equation, $g(\theta)$=θ. However, in this case, the iteration diverged. Since the absolute value of the slope is too large, a mild change in θ caused dramatic differences in $g(\theta)$

between iterations, making the result easily diverge. The reason that we chose the bisection method and Newton's method is that the bisection method always converges when given an interval $[a,b]$, which $f(a)\times f(b)<0$, and if the function converges, Newton's method converges much faster than linearly convergent method.

3.4.2 How fast is your solver?

When solving with the bisection method, it is linear convergence and the error is halved in each iteration (S=12). While solving with Newton's method, it is quadratic convergence. The results demonstrated bisection method cost way more iterations, which was 27, than Newton's method, which was less than 10 iterations. The total execution time of the bisection method is 0.018981 sec, while Newton's method cost 0.033940 sec. Even though Newton's method is quadratic convergence, it iterated in all 12 initial points, taking more time than the bisection method, which only iterated for those intervals with $f(a)\times f(b)<0$.

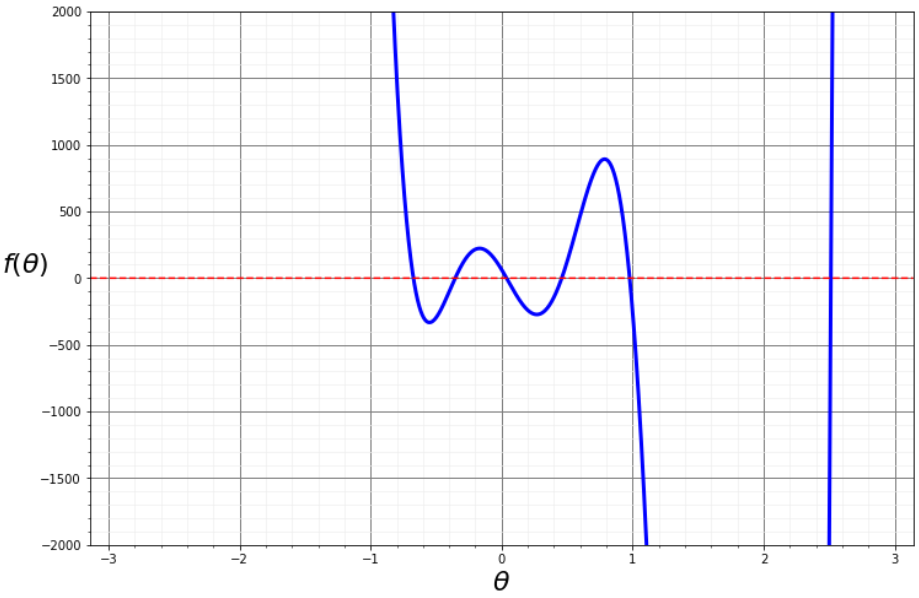
3.4.3 How accurate is the obtained solution(root)?

As we don't know the true roots, we can only estimate backward error to define how accurate the solution is. The backward error in Newton's method reached 10^{-12} and even 0. However, in the bisection method, the error was much larger compared with Newton's method. The roots from both methods had the same number before the 8 decimal digits as we expected (as the following table). Detailed data is as 3.5 in .ipynb file.

method	root	θ	f(θ)	backward error	x	y
bisection	r1	-0.720849205	6.49E-06	6.49E-06	-1.37837963	4.80625318
	r2	-0.331005185	-9.15E-06	9.15E-06	-0.91470872	4.91561878
	r3	1.143685515	6.69E-05	6.69E-05	4.48175007	2.21673553
	r4	2.115909014	-4.14E-05	4.14E-05	4.57183018	2.02444285
Newton's	r1	-0.720849204	7.28E-12	7.28E-12	-1.37837963	4.80625318
	r2	-0.331005184	6.04E-10	6.04E-10	-0.91470872	4.91561878
	r3	1.143685518	-4.58E-10	4.58E-10	4.48175007	2.21673552
	r4	2.115909014	0.00E+00	0.00E+00	4.57183018	2.02444285

3.5 Changing the parameter p2=7 (Activity 5)

We plotted the function with the new parameters and found there are 6 roots (as the following figure). Since there have more roots in this case, we evenly divided the given interval $[-\pi,\pi]$ into twenty-four intervals. We took the middle number from these intervals as the initial points. To know the differences in the time cost of both of the methods when finding more roots, we solved the equation by both methods (the result is in the following table). The execution time is 0.037898 sec in the bisection method and 0.060837 sec in Newton's method. The accuracy of Newton's method is much smaller than the bisection method and the time cost is still acceptable, so we decided to use Newton's method in the following questions.



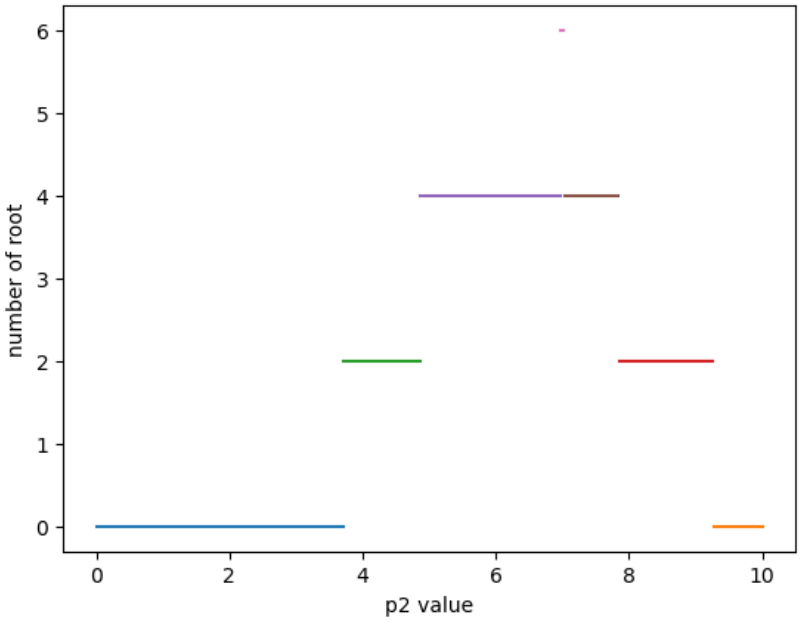
method	root	θ	f(θ)	backward error	x	y
bisection	r1	-0.67315749	2.42E-05	2.42E-05	-4.31475959	2.52643022
	r2	-0.35474027	1.19E-07	1.19E-07	-4.80489652	1.38310138
	r3	0.037766759	1.95E-06	1.95E-06	-4.94902462	0.7121484
	r4	0.458878184	7.00E-06	7.00E-06	-0.81980013	4.93233494
	r5	0.977672892	2.91E-05	2.91E-05	2.3035541	4.43775153
	r6	2.513852801	2.61E-04	2.61E-04	3.21569604	3.8287464
Newton's	r1	-0.673157486	0.00E+00	0.00E+00	-4.3147596	2.52643021
	r2	-0.35474027	-1.19E-09	1.19E-09	-4.80489652	1.38310138
	r3	0.037766761	-1.36E-12	1.36E-12	-4.94902462	0.7121484
	r4	0.458878181	1.19E-09	1.19E-09	-0.81980017	4.93233491
	r5	0.977672895	1.82E-12	1.82E-12	2.3035541	4.43775152
	r6	2.513852799	-5.82E-11	5.82E-11	3.21569604	3.8287464

3.6 Finding $p2$ with two poses (Activity 6)

By generating the interactive plot (as figure in 3.7.1 in .ipynb file) (7), we knew when p2 is greater than 10, there are no roots anymore. We substituted the p2 value with 0.01 differences in Newton's method to find the number of roots. We found the p2 value that only has two roots (as the result in 3.7.2 in .ipynb file).

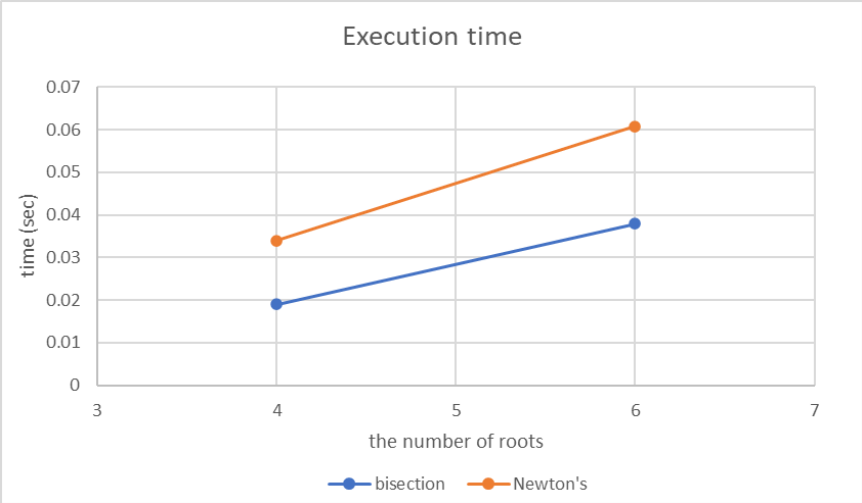
3.7 Finding $p2$ with 0, 2, 4, and 6 poses (Activity 7)

Although Newton's method takes longer execution time than the bisection method, we used Newton's method to solve the problem since it is more accurate than the bisection method and the time spent is acceptable. From 3.6, we could find all six roots when starting with 24 points. As a result, we used the 24 starting points in Newton's method to solve the problem. We searched from the range 0<p2<=10 with a 0.01 difference in p2 between each iteration as there were no roots when p2>10 (as the interactive plot in 3.7.1 in .ipynb file). The final result is as the following figure (8).



4. Conclusions

Through the results, we found Newton's method is accurate and quick in converging with fewer numbers in iteration. However, the execution time is slow in Newton's method (as the following figure) because it iterated at each starting point and might repeatedly find the same roots, and the bisection method only iterated when $f(a) \times f(b) < 0$ and it did not repeatedly find the roots, causing a shorter execution time. Even though Newton's method takes longer execution time, the small backward error made us choose Newton's method for the following problems. The number of starting intervals or the starting point played an important role in finding roots. When we use a small number of intervals, we might not find all the roots. On the other hand, when we generate a great number of intervals, it would take a long time to execute.



5. References

1. Sauer, Tim. 2018. *Numerical Analysis*. N.p.: Pearson.
2. Davies, Joseph. n.d. “Root-Finding Methods in Python. Bisection, Newton’s and Secant… | by Andrew Joseph Davies.” Finding the roots. Accessed October 20, 2022. https://towardsdatascience.com/root-finding-methods-from-scratch-in-python-84040c81a8ba.
3. Reti, Daniel. 2021. “The Best Numerical Derivative Approximation Formulas | by Daniel Reti.” Cantor's Paradise. https://www.cantorsparadise.com/the-best-numerical-derivative-approximation-formulas-998703380948 .
4. “Numerical differentiation.” n.d. Wikipedia. Accessed October 20, 2022. https://en.wikipedia.org/wiki/Numerical_differentiation#/media/File:AbsoluteErrorNumericalDifferentiationExamp.
5. “matplotlib.pyplot.plot — Matplotlib 3.6.0 documentation.” n.d. Matplotlib. Accessed October 20, 2022. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html.
6. “matplotlib.pyplot.scatter — Matplotlib 3.6.0 documentation.” n.d. Matplotlib. Accessed October 20, 2022. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html.
7. “ipywidgets interact.” n.d. IPyWidgets. Accessed October 20, 2022. https://ipywidgets.readthedocs.io/en/stable/examples/Using%20Interact.html.
8. “seaborn.lineplot — seaborn 0.12.1 documentation.” n.d. Seaborn. Accessed October 20, 2022. https://seaborn.pydata.org/generated/seaborn.lineplot.html.