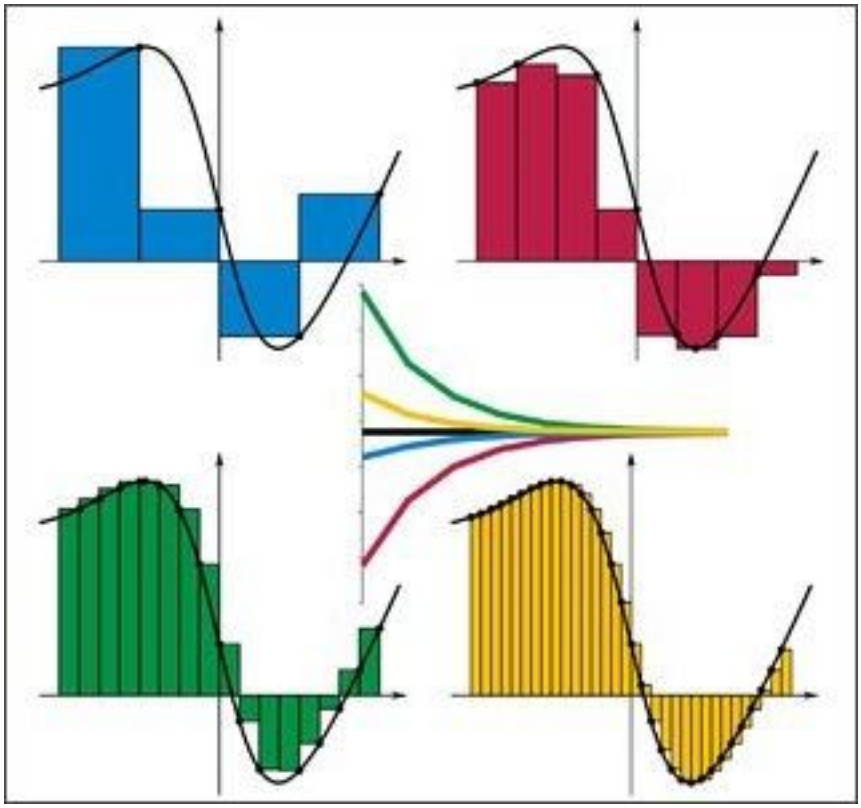
CS-374 Project
Mid-Review



By:

Shantanu Tyagi - 201801015

Shivani Nandani - 201801076

Pratvi Shah - 201801407

Arkaprabha Banerjee - 201801408

Project Guide:

Prof. Madhukant Sharma

Ms. Jhanvi Chauhan (Teaching Assistant)

Section 1: Root Finding Method

For this section we considered 2 physically significant equations and used iterative root finding techniques to find the solution of our equations.

Part 1 : Planck's Radiation Law Equation

Problem Statement: Find the wavelength for maximum energy density within an isothermal blackbody with absolute temperature=300K.

Solution:

The equation which gives us the energy density of an isolated blackbody is:

$$\phi(\lambda) = \frac{8\pi ch\lambda^{-5}}{e^{(ch/kT\lambda)} - 1}$$

λ = Wavelength of radiation
c= Speed of light
h= Planck's Constant
k= Boltzmann Constant
T= Absolute temperature of blackbody
In order to find maximum energy density we differentiate $\Phi(\lambda)$ and equate it to 0.

After simplifying we obtain the equation to be solved as:

$$e^{-x} = 1 - \frac{x}{5} \text{ where } x=ch\lambda/kT$$

The final iterative equation comes out to be:

$$f(x) = e^{-x} - 1 + \frac{x}{5} = 0$$

On plotting the graph of $f(x)$ we found that it has two roots at $x = 0, 5.$ We used five root finding methods with appropriate starting points to get the root $x = 4.9651$ as $x = 0$ root does not give us finite value of the wavelength. On getting x we obtain λ as $\lambda = ch/kTx$.

Following is the table we get using our code:

Method	x	Wavelength	Iterations	Accuracy
'Bisection'	4.96511077880859	9.66583119234984e-06	16	1e-06
'Newton'	4.96511423291115	9.66582446807912e-06	3	1e-06
'Secant'	4.96511423174423	9.66582447035082e-06	5	1e-06
'Regula-Falsi'	4.9651142314364	9.66582447095009e-06	6	1e-06
'Steffensen'	4.96511423775616	9.66582445864712e-06	3	1e-06

Observations:

1. Bisection method converged for all values of a, b given x $\in (a, b)$, here we took $(a, b) = (1, 6)$.
2. Newton method converged to $x = 0$ when x_0 was near to 0 instead of 4.9651, here we have taken x_0 =3.5.
3. Secant method converges to $x = 4.9651$ whenever $1 \leq x_0$ and $1.65 \leq x_1$,for all other cases it does not converge to $x = 4.9651$. Here, we have taken $(x_0, x_1) = (1, 6)$
4. Regula-Falsi method converges to $x = 4.9651$ whenever the initial condition is satisfied i.e., for $x_0,\ x_1$ such that $f(x_o) * f(x_1) < 0$ is satisfied. Here, we have taken $(x_0, x_1) = (1, 6)$.
5. For Steffensen Method we have taken $(x_0, x_1) = (1, 6)$.

Conclusions:

We can see from the table that for the same accuracy Newton Raphson Method and Steffensen Method give the root in the same number of iterations, suggesting to have quadratic convergence as opposed to linear convergence of the Bisection Method.

We can also verify that the Secant Method converges faster than the Regula-Falsi Method.

From all the iterative methods we can conclude that the **wavelength for maximum energy density** of an isolated blackbody at temperature 300K is **9.6658 e-06 m.**

Part 2 : Projectile Motion Equation

Problem Statement: Consider the projectile problem in which a projectile is launched from a tower of height h > 0, with initial speed v and at an angle θ with respect to the horizontal onto a hill. We wish to find the optimal launch angle θ_m which maximizes the horizontal distance. In our calculations, we neglect air resistances.

Solution:

The equation which gives us the maximum x for a given h, v and $g = 9.8\ m/s^2$ is:

$$y = h + \frac{v^2}{2 \cdot g} - g \cdot \frac{x^2}{2 \cdot v^2}$$

We then find θ_m using the following formula:

$$\theta_m = \arctan\left(\frac{v^2}{x_m \cdot g} \right)$$

Following is the table we get using our code:

Method	Xm	Theta	Iterations	Accuracy
"Newton"	49.823	0.68636	4	1e-06
"Regula Falsi"	49.823	0.68636	14	1e-06
"Secant"	49.823	0.68636	7	1e-06
"Steffensen"	49.823	0.68636	3	1e-06
"Bisection"	49.823	0.68636	21	1e-06

Observations:

1. Newton method converged to $x = 49.832$ when $x_0 = 30$.
2. Regula-Falsi method converges to $x = 49.832$ whenever the initial condition is satisfied i.e., for $x_0,\ x_1$ such that $f(x_o) * f(x_1) < 0$ is satisfied. Here, we have taken $(x_0, x_1) = (0, 30)$.
3. Secant method converges to $x = 49.832$ for $(x_0, x_1) = (0, 30)$.
4. For Steffensen Method we have taken $x_0 = 30$.
5. Since the equation is parabolic, Bisection method converged for all values of a, b given x ∈ (a, b), here we took $(a, b) = (0, 50)$.

Conclusions:

For a given error, Steffensen method, Newton's method and Secant method give result in less than 10 iterations, with Steffensen method being the fastest. The linear convergence of bisection can be concluded based on the number of iterations for Bisections as compared to the other methods.

Section 2: Gaussian Elimination

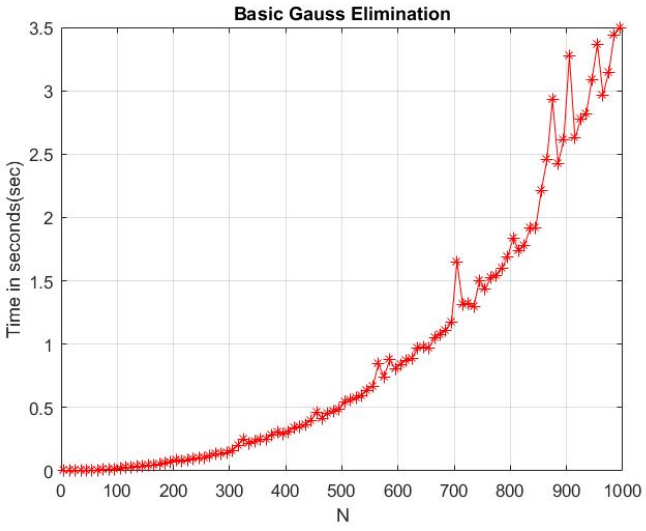
In this section we look into the time complexity, effects of perturbation and analyze the ill-conditioned matrices.

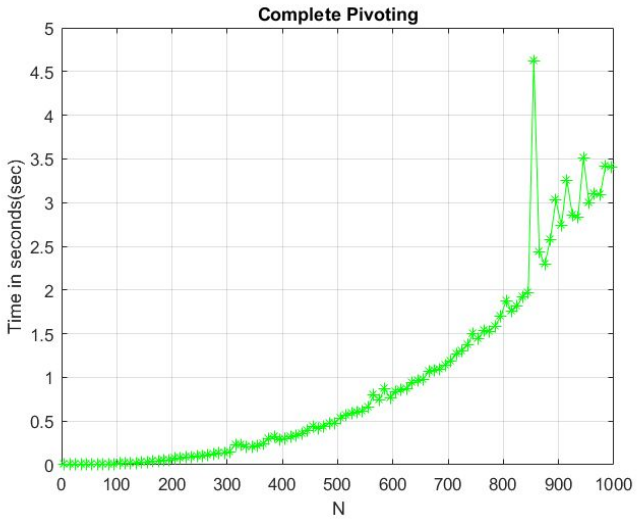
Part 1: Time Complexity

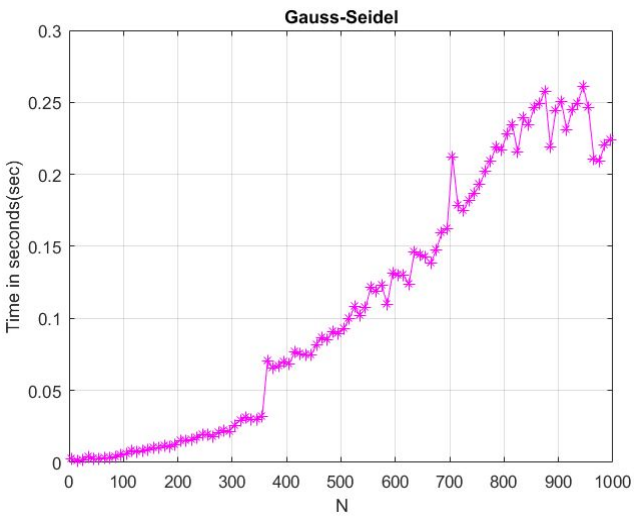
- Here, we have analysed how the time required to solve a given NxN matrix varies with the size of the matrix N.
- In this we generated T=5 matrices for a given size N and calculated the time required to solve the system of linear equations given by the matrix A and b. We added the time taken to solve the system of equations T times and then took its average to get a better approximation of the time taken to solve a matrix of dimension NxN.
- We did this experiment where N=5 to 1000 in steps of 10, A and b were randomly generated matrices of dimension NxN and Nx1 respectively.
- Time taken to find the solution was measured using the MATLAB functions *tic* and *toc*.
  tic : Marks the starting of the clock to measure time
  toc : Stops the clock and gives out the time elapsed after the corresponding *tic* function

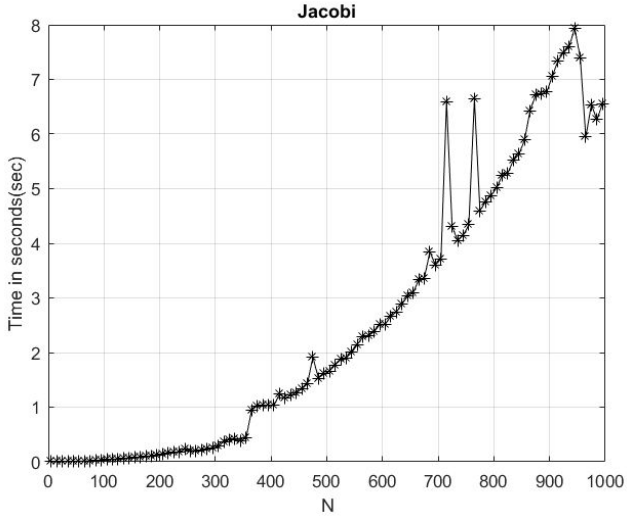
  Observations
  The above procedure was performed using six methods (Richardson, Jacobi, Gauss-Seidel and three types of gauss elimination technique). The general trend as observed from the plots is in accordance with the one we studied in class i.e., the trend given us the complexity of solving NxN matrix as O(N^3).

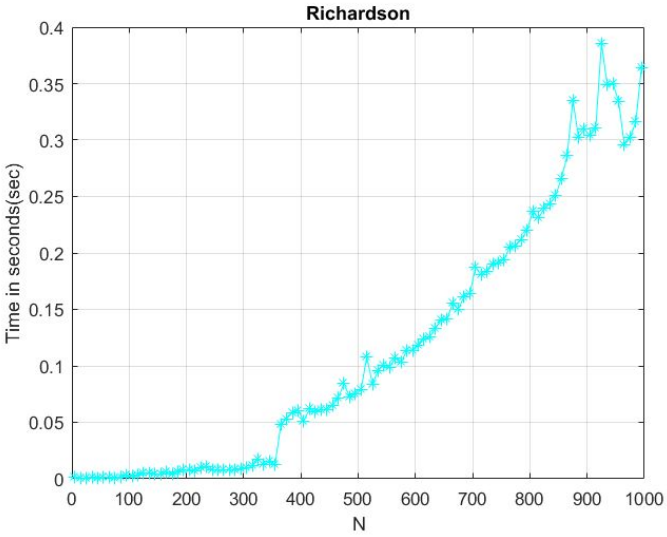
  The plots for the same are given below.

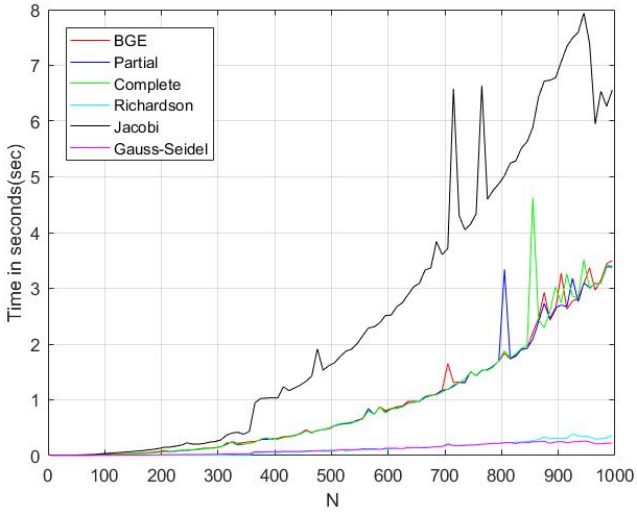












As we can see from the comparison plot, all the three methods (Basic Gauss Elimination, Partial Pivoting, Complete Pivoting) take almost the same time to solve a given system of linear equations.

Verifying the observation:

N_1 =500 T_1 ≃ 0.5s

N_2 =2* N_1 =1000 T_2 ≃ $(2 \times N_1)^3 = 8 * T_1 =$ 4s

Conclusion:

The above values and the plots clearly signify that the solution of system of N linear equations has O (N^3) time complexity for the gauss elimination methods.

We could also conclude that as the matrices were randomly generated and we have no surety for them to be diagonally dominant, so the Jacobi method came out to be the slowest of all for every value of N.

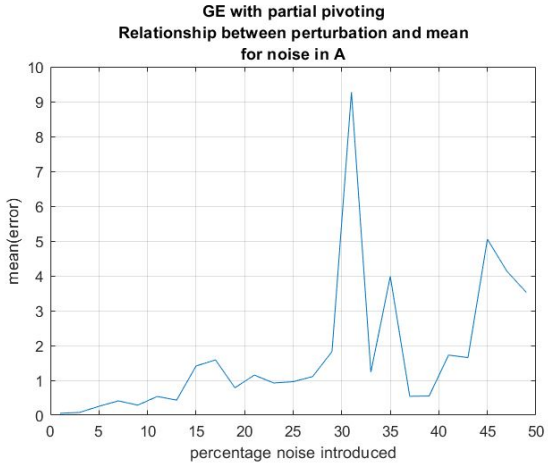
We could also conclude clearly from the plots that Richardson and Gauss Seidel methods are better iterative methods than Jacobi and the various ways of gauss elimination.

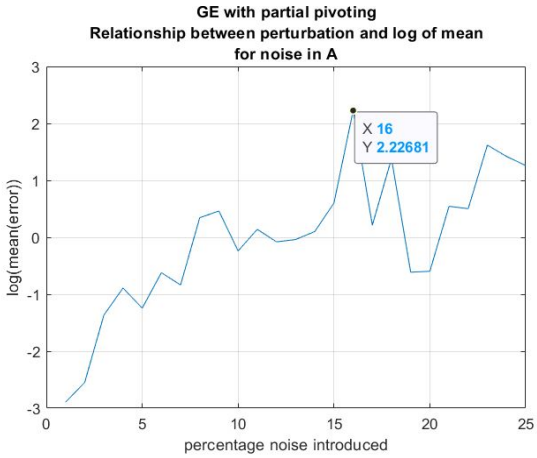
Part 2: Effect of Perturbation

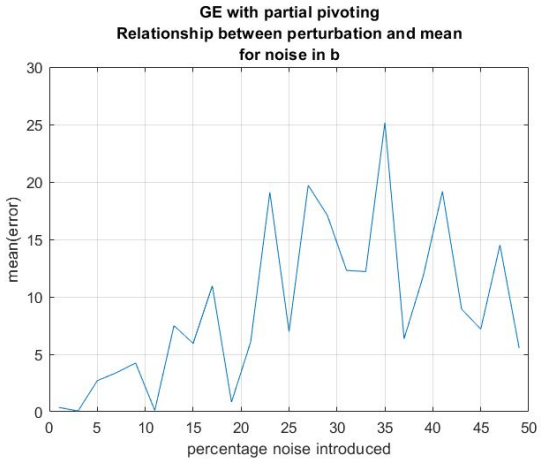
- The Perturbation Experiment seeks to find how changing the values of a parameter matrix and its corresponding coefficients can affect the solution of the system. This is done by adding a 1% noise to the original system.
- The noise added is gaussian
- A and b were generated randomly (size - 5x5).
- This noise was added to both A and b, keeping the other constant. Then the system was solved for x using the Basic Gauss Elimination and Gauss Elimination with Partial Pivoting.
- The infinity norm of the *error matrix* (= *abs* (*x_noise - x*)) for each iteration was calculated.

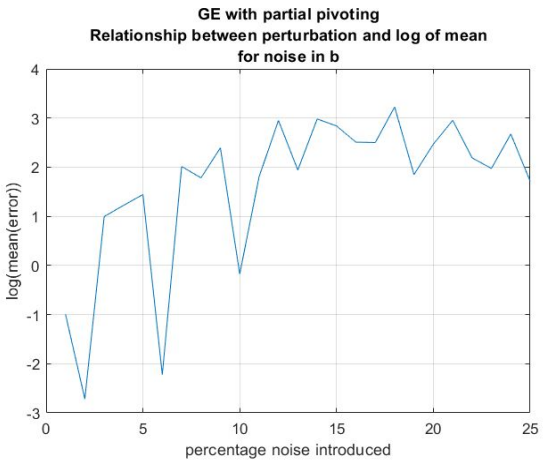
Observations

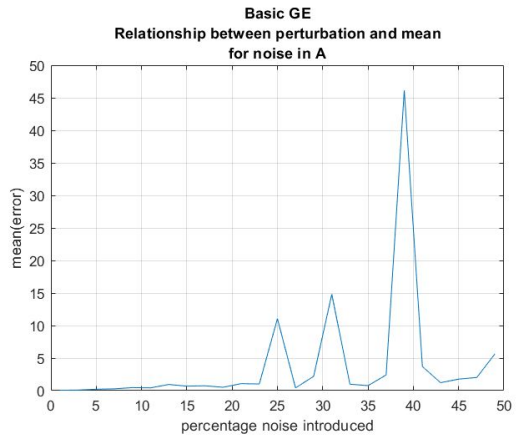
The process of adding noise was repeated for noise varying from 1 percent to 50 percent in steps of 2.

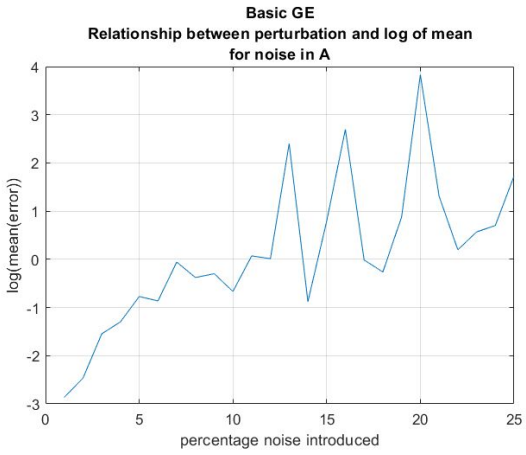


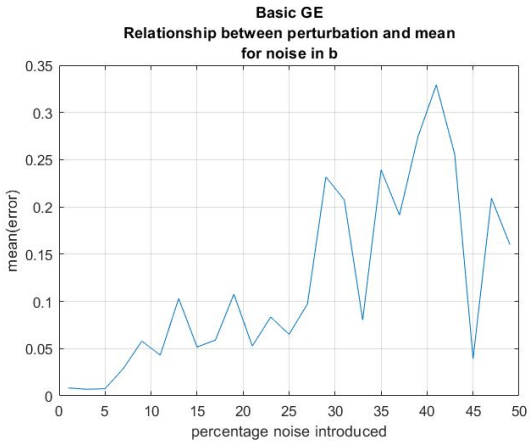


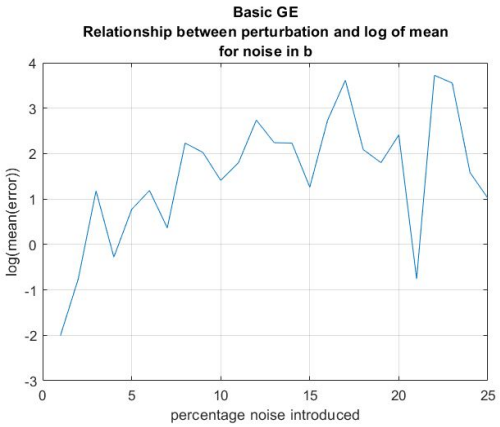












It is clear from these graphs that as the percentage noise added increases, the mean of error also increases.

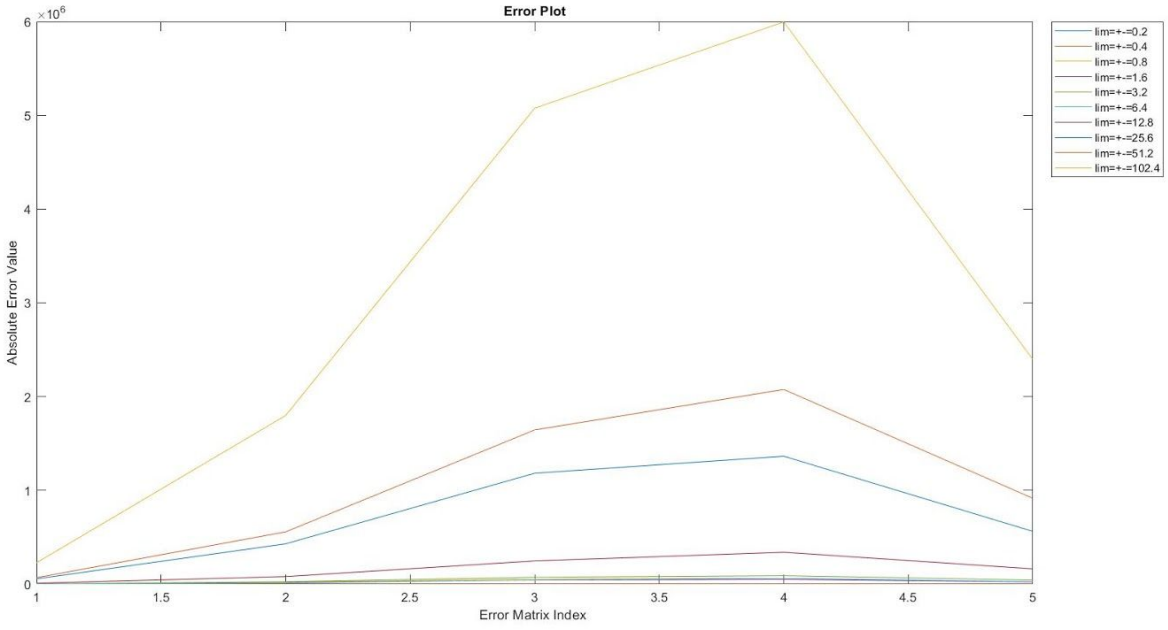
Conclusion

Error increases as the perturbation (noise added to b) increases or even when noise added to A increases.

Part 3: Analysis of Vandermonde Matrices

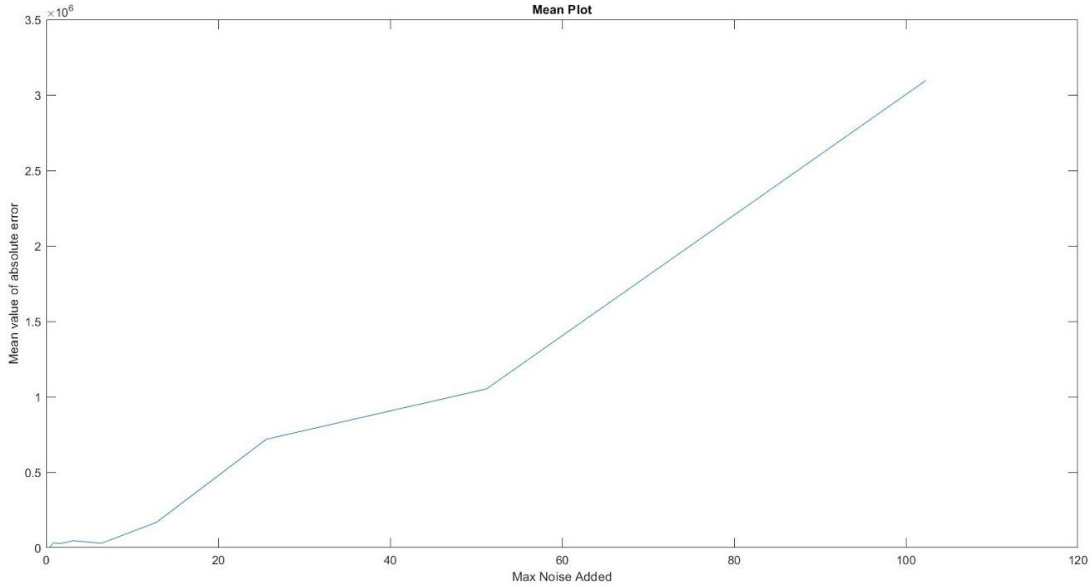
- Vandermonde Matrices are a set of matrices which are inherently ill-conditioned in nature. The primary motive is to observe the variations in the solution vector when the requirement vector is perturbed, whilst tweaking certain other parameters as well. The following procedure was followed :
    1. Variation of the solution vector on perturbing/adding increasing amounts of noise to the constant/requirement vector.
    2. Variation of the solution vector with increasing dimensions of the vandermonde matrix while maintaining a constant threshold of noise in the requirement vector.
    3. Mean and standard deviation analysis have been done to further analyze and report the findings.
    4. The above procedure has been done for both Basic and Partial Gauss Elimination Methods.
- The primary motive is to verify the fact that except for a very exclusive class of vandermonde matrices, most of them tend to be severely ill-conditioned. This ill-conditioned behaviour further follows an increasing trend with increasing dimensions of the aforementioned matrix.

- **Experiment 1 - Varying Noise in the constant/requirement vector with a fixed Vandermonde Matrix (Basic Gauss Elimination)**

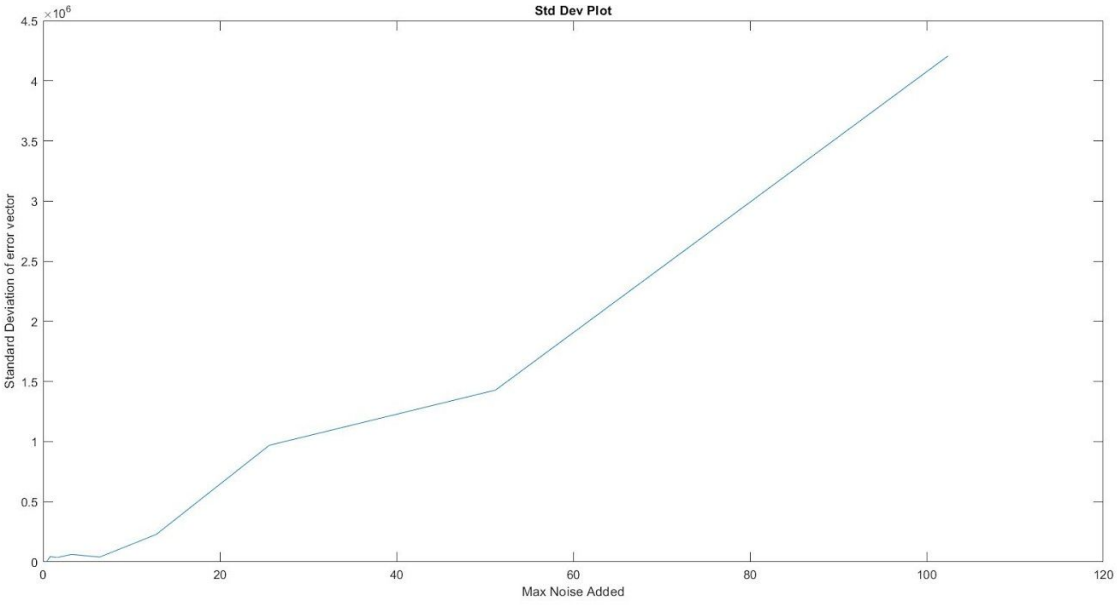


We observe a general trend that for increasing amounts of noise the solution vector is perturbed rather significantly.The noise is added from a uniform distribution with max and min limits mentioned in the legends. This experiment highlights the ill-conditionedness of the vandermonde matrix. To analyze the graph we need to look at the integer points on x-axis. Each of those points represent the absolute error corresponding to the i^{th} cell in the error vector.

For this particular case we have considered a 5*5 matrix matrix, hence we have 5 data points on the x-axis.



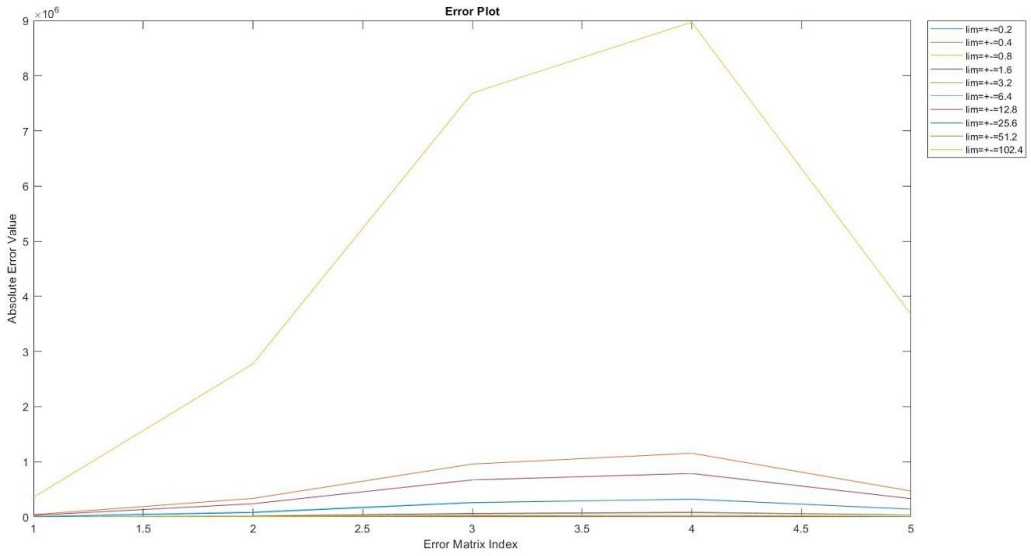
A similar trend can be seen for the mean plot. With increasing amounts of noise the mean of the absolute values for all the entries in the error vector tends to increase. Although the general trend is upwards, we may face anomalies at times on account of the fact that random matrices are considered.

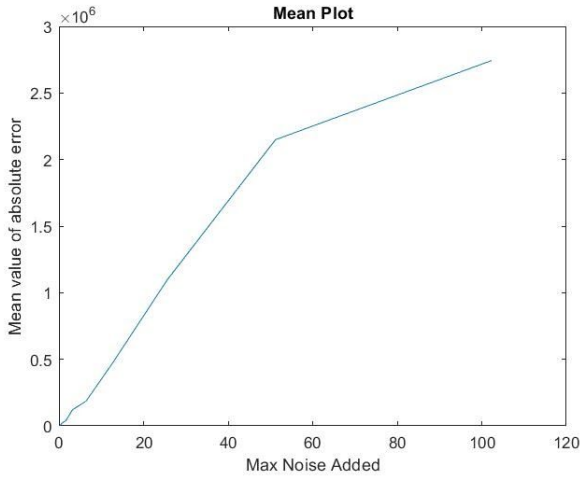


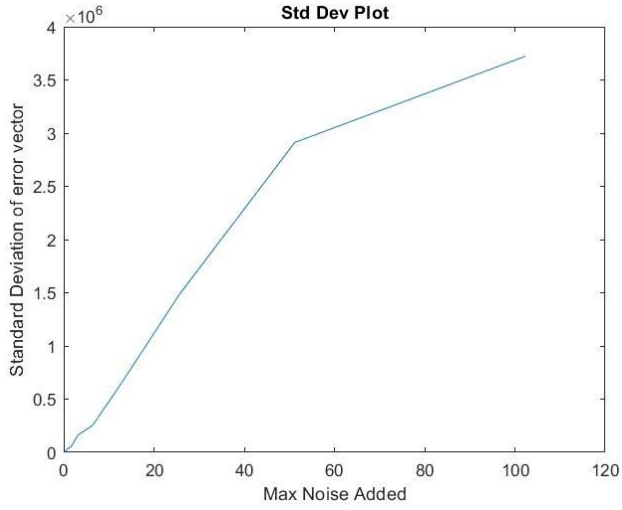
The same observation is also extended for this case as well. For the purpose of our calculations the actual value of the error is considered rather than the modulus so as to highlight the overall deviation.

- **Experiment 2 - Varying Noise in the constant/requirement vector with a fixed Vandermonde Matrix (Gaussian Elimination with Partial Pivoting)**

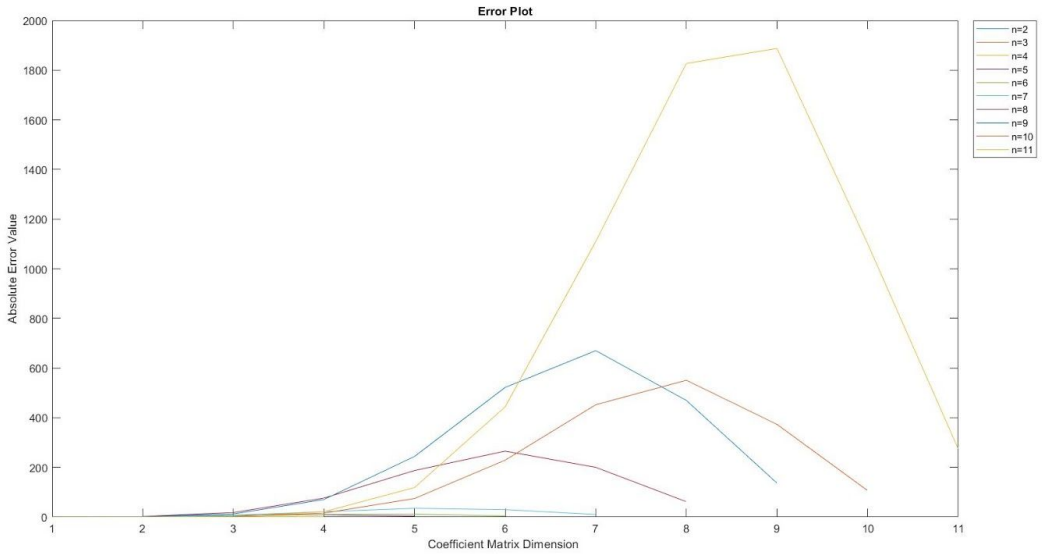
  The above observations were verified by performing the same experiment by using Gaussian Elimination with Partial Pivoting.We can see the consistency of the plots with respect to the general trend.



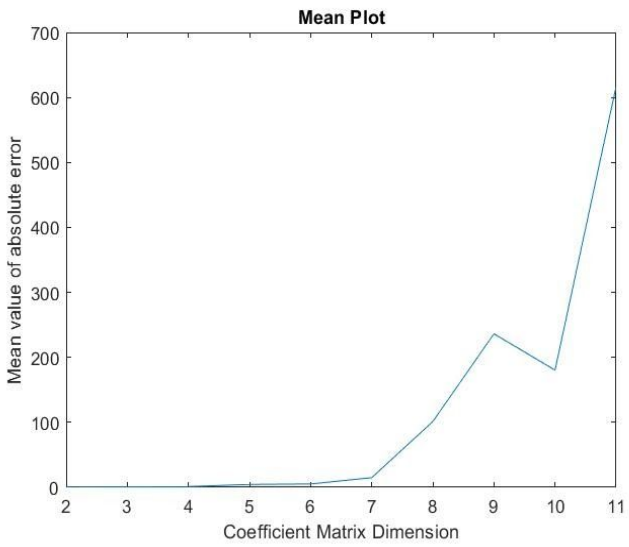


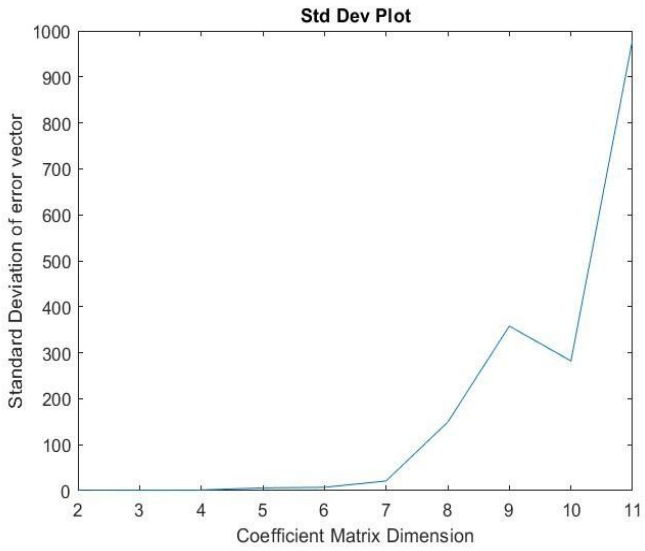


- **Experiment 3 - Varying size of the coefficient matrix with fixed threshold of noise in constant/requirement vector (Basic Gauss Elimination)**



We observe a general trend that for higher dimensional matrices the solution vector is perturbed rather significantly. This experiment also highlights the ill-conditionedness of the vandermonde matrix and it's staggering effect on increasing the dimensions. To analyze the graph we need to look at the integer points on x-axis. Each of those points represent the absolute error corresponding to the i^{th} cell in the error vector. Since we have variable vector sizes, hence the following trend.



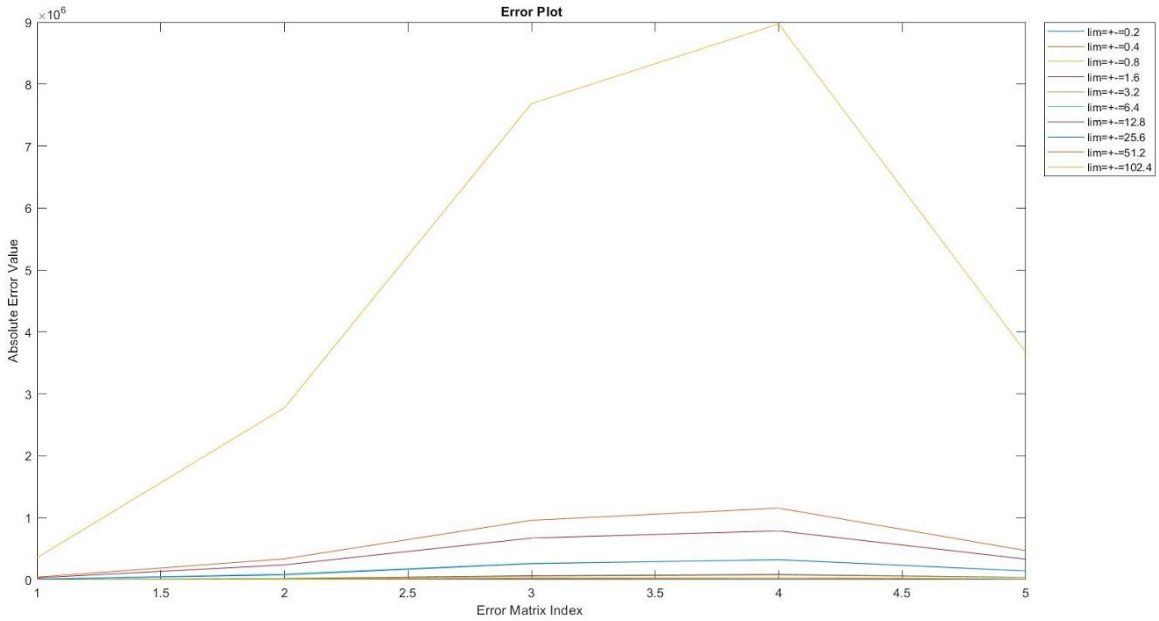


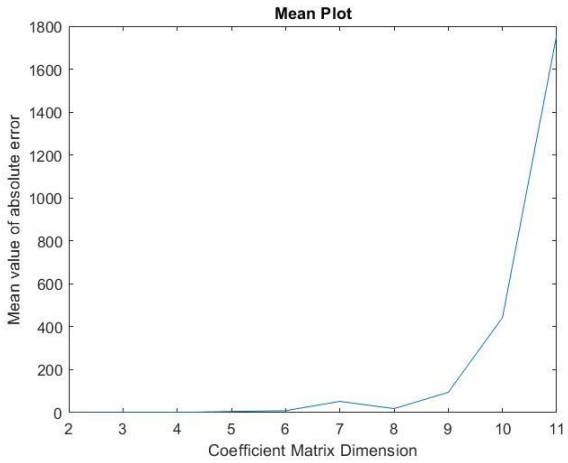
A similar trend can be seen for the mean plot. With increase in the size of the coefficient matrix, the mean of the absolute values for all the entries in the error vector tends to increase. Although the general trend is upwards, we may face anomalies at times on account of the fact that random matrices are considered.

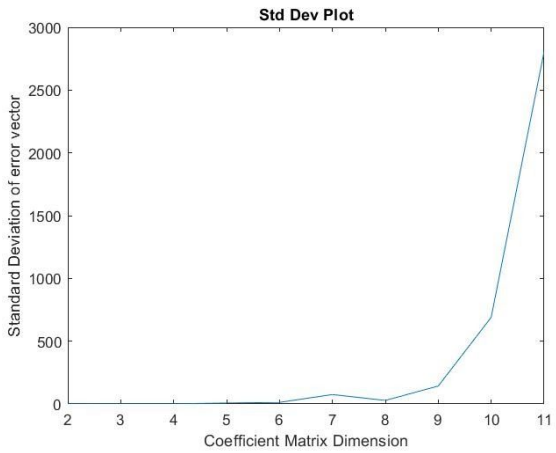
The same observation is also extended for the case of standard deviation as well. For the purpose of this calculation the actual value of the error is considered rather than the modulus so as to highlight the overall deviation.

- **Experiment 4 - Varying size of the coefficient matrix with fixed threshold of noise in constant/requirement vector (Gaussian Elimination with Partial Pivoting)**

  Experiment 3 is again performed using Gaussian Elimination with Partial Pivoting. We see the same trends as evident from the plots below.

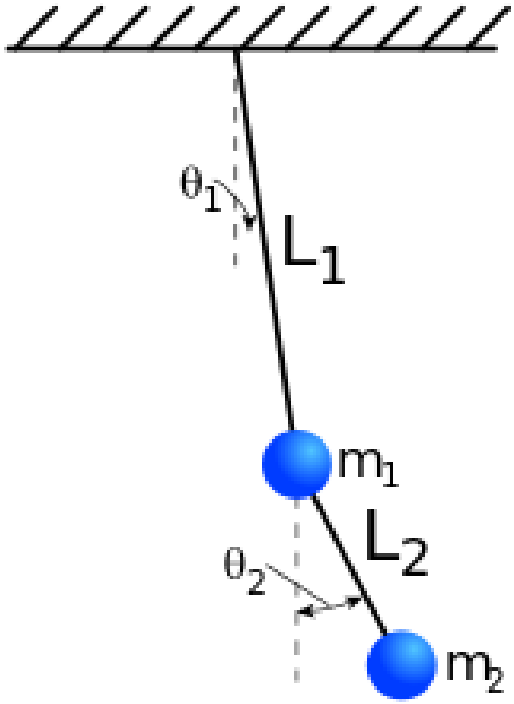






We also have special exceptions for the above experiments as seen in [arXiv:1504.02118](). For the purpose of our report, we shall not consider those cases.

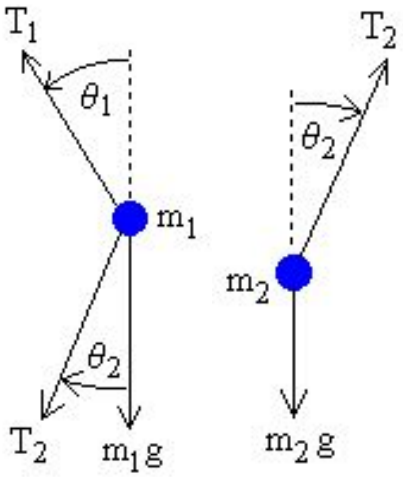
Section 3: Analysis of Double Pendulum



- In dynamical systems, a double pendulum is a pendulum with another pendulum attached to its end. It is a simple deterministic physical system that exhibits rich dynamic behavior with a strong sensitivity to initial conditions.
- The motion of a double pendulum is governed by two coupled ordinary differential equations and it is chaotic. For smaller angles however, it acts like a simple pendulum. The jump in complexity, which is observed at the transition from a simple pendulum to a double pendulum is amazing.
- We study the variations in the solution vector when certain parameters are tweaked and the procedure followed is as follows:
    1. Variation of the solution vector on perturbing the initial angle for both the pendulums while keeping other parameters fixed.
    2. Variation of the solution vector on changing the value of time step interval used in the approximation while keeping other parameters same.
    3. The final position and corresponding error have been plotted in each case to analyze the instability in the system.
- The primary motive is to verify the fact that the system can be modelled numerically using linear approximations and also deterministically and double pendulums with near identical initial conditions diverge over time displaying the chaotic nature of the system.
- We place the origin at the pivot point of the upper pendulum and get the following equations:

For position,

$$x_1 = L_1 \sin\theta_1 \quad \text{and} \quad y_1 = -L_1 \cos\theta_1$$
$$x_2 = x_1 + L_2 \sin\theta_2 \quad \text{and} \quad y_2 = y_1 - L_2 \cos\theta_2$$



Double differentiation gives acceleration as,

$$x_1'' = -\theta_1'^2 L_1 \sin\theta_1 + \theta_1'' L_1 \cos\theta_1$$
$$y_1'' = \theta_1'^2 L_1 \cos\theta_1 + \theta_1'' L_1 \sin\theta_1$$
$$x_2'' = x_1'' - \theta_2'^2 L_2 \sin\theta_2 + \theta_2'' L_2 \cos\theta_2$$
$$y_2'' = y_1'' + \theta_2'^2 L_2 \cos\theta_2 + \theta_2'' L_2 \sin\theta_2$$

We have the force balance equations as,

$$m_1 x_1'' = -T_1 \sin\theta_1 + T_2 \sin\theta_2 \text{ and } m_1 y_1'' = T_1 \cos\theta_1 - T_2 \cos\theta_2 - m_1 g$$
$$m_2 x_2'' = -T_2 \sin\theta_2 \text{ and } m_2 y_2'' = T_2 \cos\theta_2 - m_2 g$$

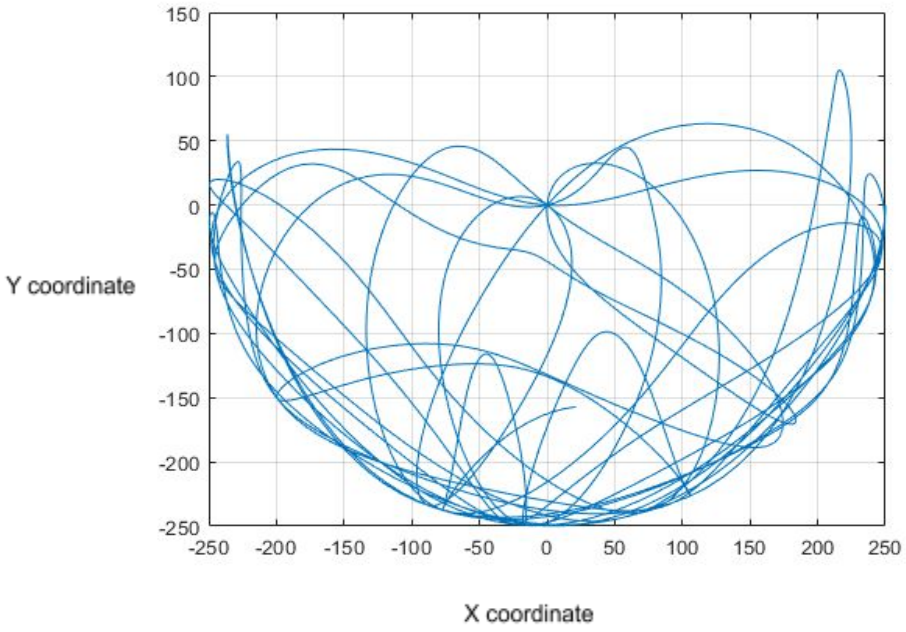
Solving all these equations, we get

$$\theta_1' = \omega_1$$
$$\theta_2' = \omega_2$$

$$\omega_1' = \frac{-g(2m_1 + m_2)\sin\theta_1 - m_2 g \sin(\theta_1 - 2\theta_2) - 2\sin(\theta_1 - \theta_2) m_2 (\omega_2^2 L_2 + \omega_1^2 L_1 \cos(\theta_1 - \theta_2))}{L_1 (2m_1 + m_2 - m_2 \cos(2\theta_1 - 2\theta_2))}$$

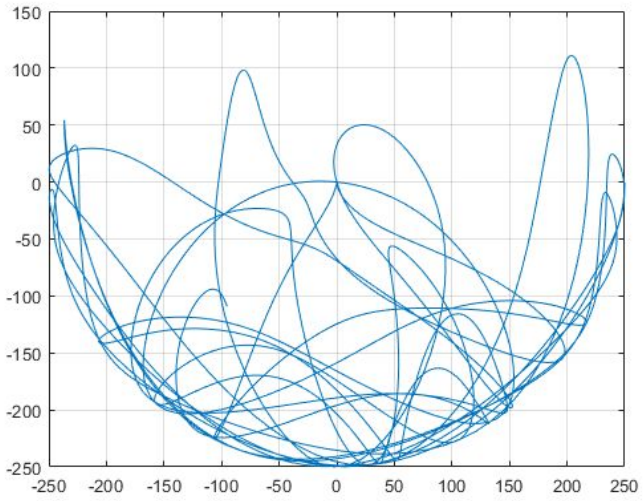
$$\omega_2' = \frac{2\sin(\theta_1 - \theta_2)(\omega_1^2 L_1 (m_1 + m_2) + g(m_1 + m_2)\cos\theta_1 + \omega_2^2 L_2 m_2 \cos(\theta_1 - \theta_2))}{L_2 (2m_1 + m_2 - m_2 \cos(2\theta_1 - 2\theta_2))}$$

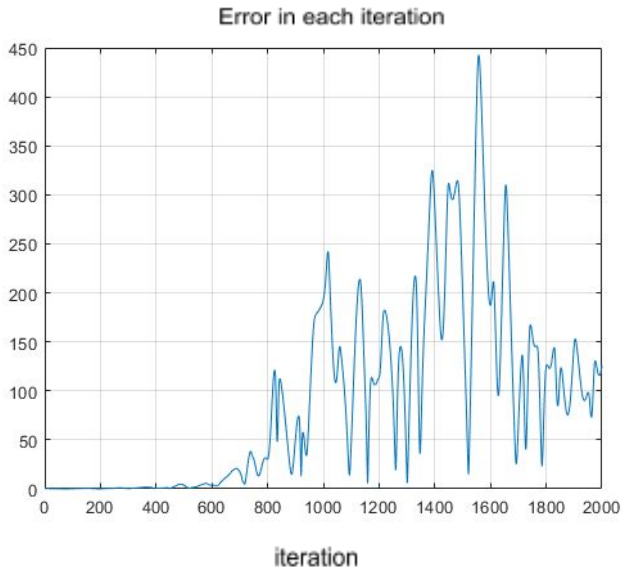
This is the numerical form that we can now plug into the algorithm to approximate the solution. Plot for final position(2D cartesian) of bob 2 after 2000 iterations with time step = 0.5 is shown below. We will use this as a base case and compare all the subsequent outputs with this simulation.



- **Experiment 1:**

  Plot for final position of bob 2 after 2000 iterations with time step = 0.5 but perturbing the initial starting angle by 0.001 radians.





As it is evident from the plot, even a minute variation in input conditions leads to greater and greater errors in the final position. This highlights the inherent instability of the system. The error plot shown below shows the 2D displacement between actual and perturbed values.

On plotting for a higher number of iterations, the error increases locally. This is because even though the error is constantly propagating and increasing, the position of bob 2 might coincide due to the fact that in the motion of a pendulum the angles repeat after intervals. These intervals of repetition are unpredictable in case of double pendulum but the result is seen in error plot as unpredictable locally increasing displacement error plot. In simple words, since the pendulum motion is restricted, the final position might coincide sometimes hence justifying the dips in the error plot.

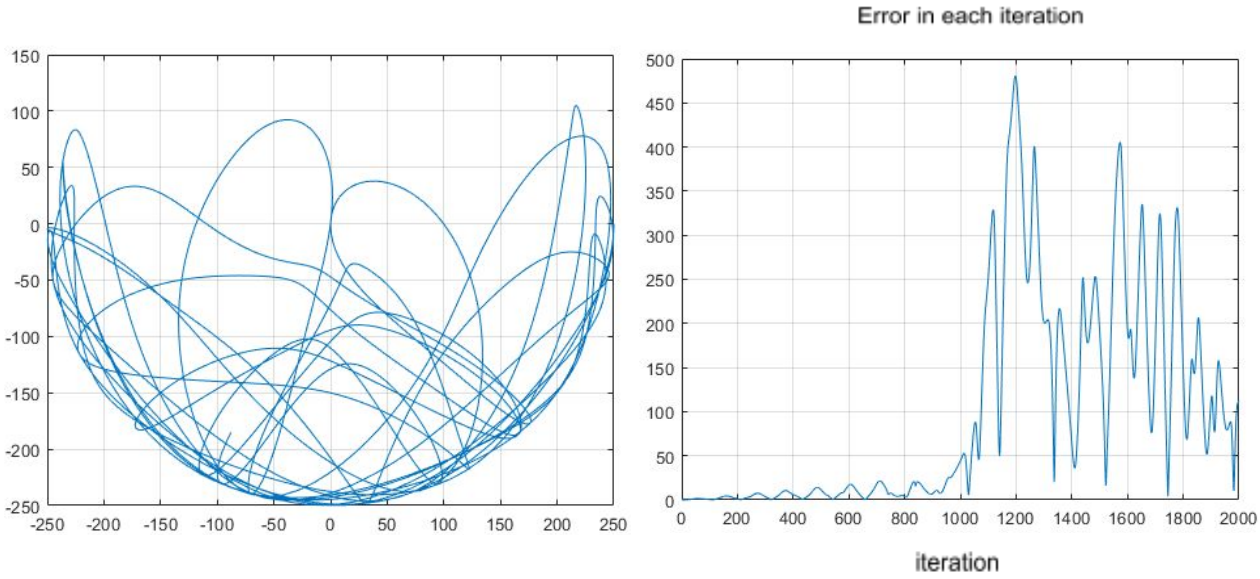
NOTE:

1. $Error(i)^2 = (X_2'(i) - X_2(i))^2 + (Y_2'(i) - Y_2(i))^2$
   Where i = iteration/ data point
2. For values of g < 4, time step size near to 1 converges to the solution however for g values > 4 and < 10, time step size around 0.5 converges. We can take much smaller time steps say 0.01 but that would increase the time complexity and calculate very close values so in order to see the proper motion a lot of iterations would be required.

- **Experiment 2:**

  In most physical stable systems this minute change does not have much effect on the output like in case for solving for projectile motion with drag. However the double pendulum system is chaotic and the solution.

  Plot for final position of bob 2 after 2000 iterations with time step = 0.501 but the same angle as in the base case is shown on the left and the error plot on the right.



  Error plot again has the same explanation as before however for early iterations a ripple like phenomenon is seen. After running the simulation with multiple values, we have come to a conclusion that the early minima are the positions where the pendulum is at rest (extreme positions). As a result the position value doesn't change much when it's almost at rest and hence the error is less. As the error keeps propagating this pattern breaks since the original pendulum and the one with perturbed time step do not come to rest at similar time/ iteration value.

  Here's the link for the simulation ANIMATION that we made in java for the same, along with source code, dependencies and x64 application.

Section 4: Lagrange Interpolation

In this section we made use of the results obtained in Section 3. We used the values of $x_1()$, $y_1()$ and θ and we applied Lagrange Interpolation technique to estimate the polynomials $x_1()$ and $y_1()$.

We performed interpolation using 25, 50 and 100 nodes and obtained three different set of polynomials :

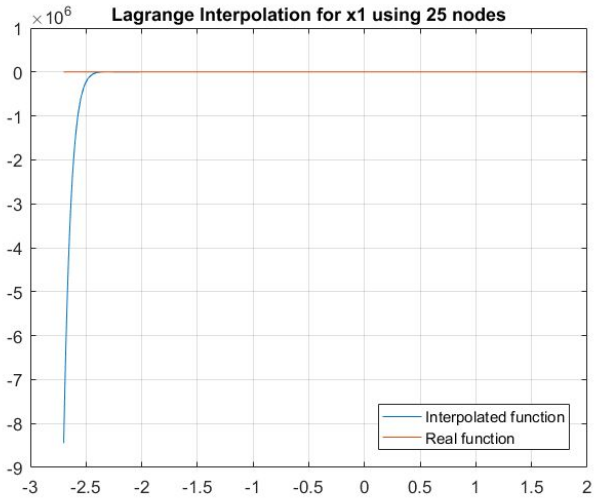
- $x_{25}(\theta)$, $y_{25}(\theta)$ for 25 nodes
- $x_{50}(\theta)$, $y_{50}(\theta)$ for 50 nodes
- $x_{100}(\theta)$, $y_{100}(\theta)$ for 100 nodes

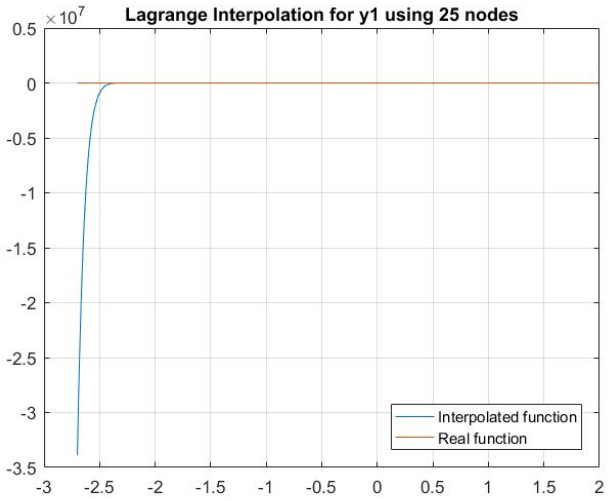
Observations:

After the polynomials were obtained we compared them with the original polynomials which led us to the following graphs:

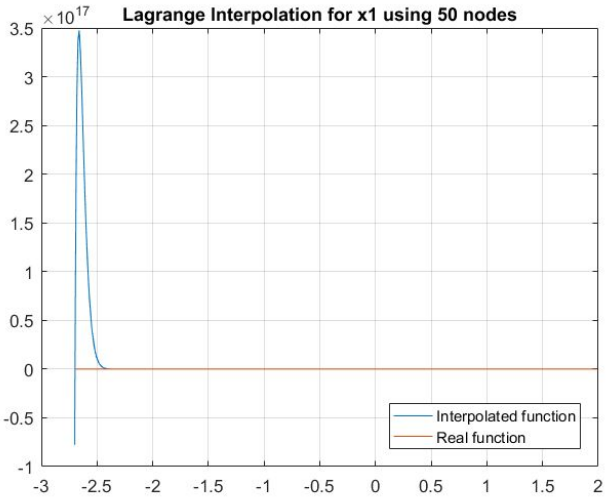
For all plots: x-axis is $\boldsymbol{\theta}$ and y-axis has $\mathbf{x_i}$ and $\mathbf{y_i}$ for i=25,50,100.

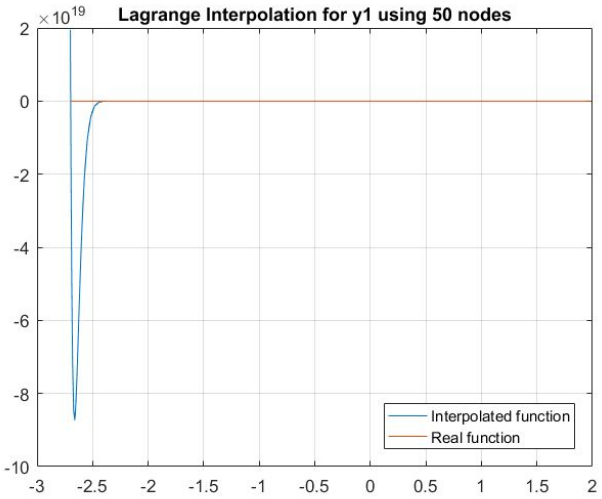
25 Nodes



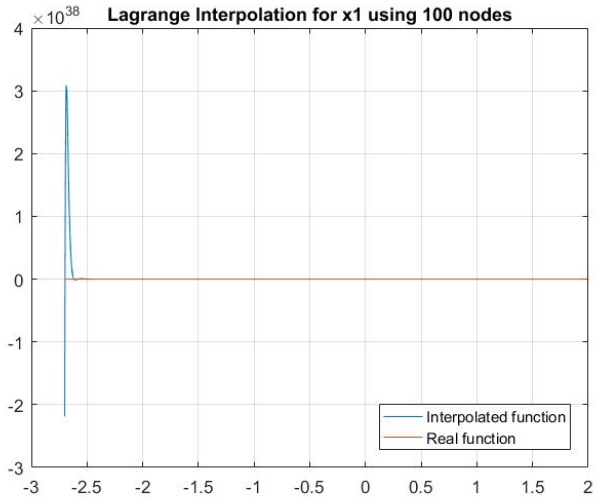


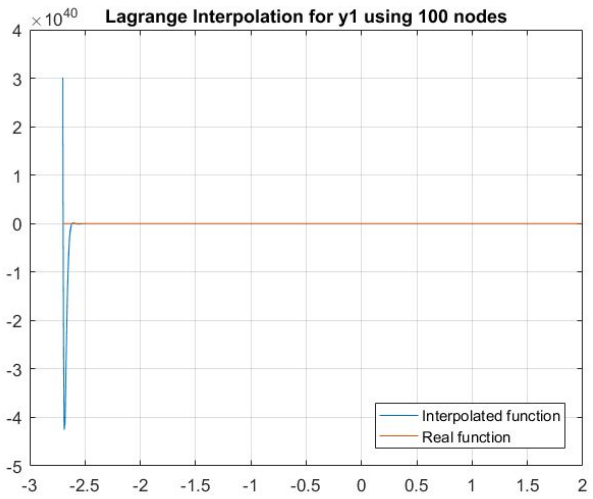
50 Nodes





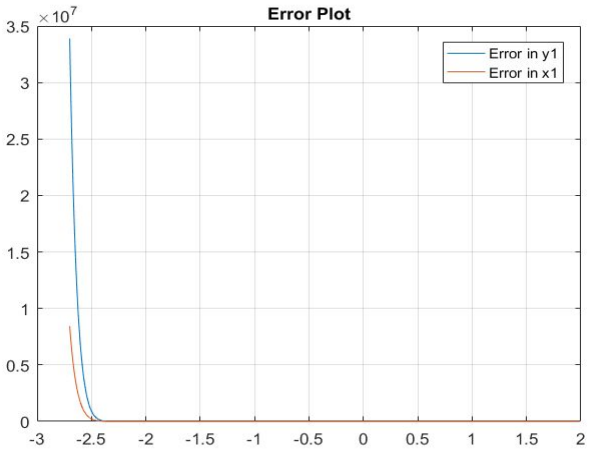
100 Nodes



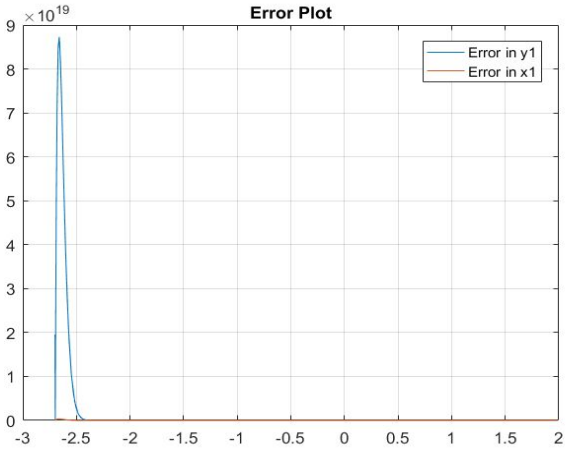


We can observe that as we increased the number of nodes the spike/anomaly in both $x_i(\theta)$ and $y_i(\theta)$ where i = 25,50,100 , reduces in width suggesting greater accuracy in interpolated polynomials.
To further verify the same we also obtained the error plots (absolute error, θ) for all the three sets of equations.

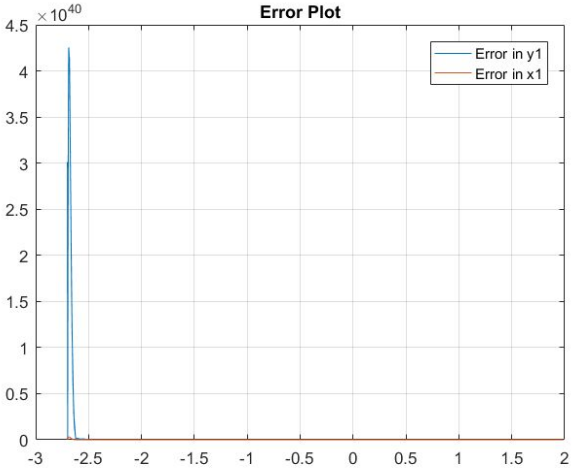
25 Nodes



50 Nodes



100 Nodes



as

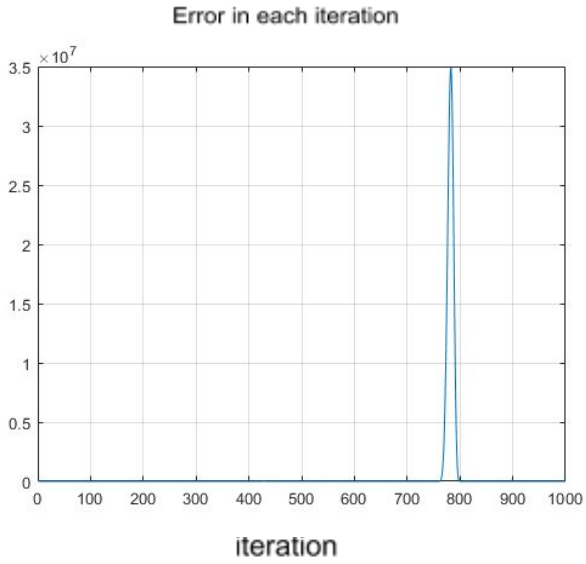
We have taken absolute error for visualisation = abs(x_i - x_1) or abs(y_i - y_1)
From the error plots we can verify that

we increase the number of nodes for interpolation we can obtain better

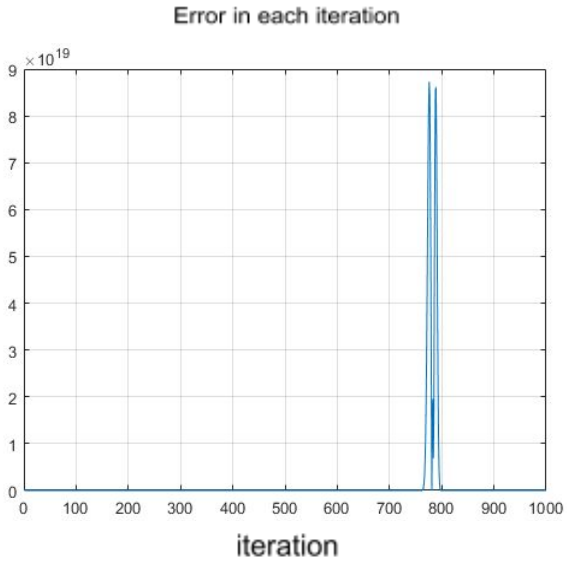
accuracy.

We obtained the values of $x_1()$ and $y_1()$ using the lagrange interpolation polynomial and now we can use these values to predict the values of $x_2()$ and $y_2()$ in the previous question. For this, we take each of the 1000 interpolated values of $x_1()$, $y_1()$ and find the values of $x_2()$, $y_2()$ using a slightly modified code and compare each of the 1000 predicted values obtained in this manner with the actual values for 1000 iterations. Then we plot the displacement error plot like we did in the previous question for 25, 50 and 100 nodes used to find the interpolating polynomial.

25 Nodes used



50 Nodes used



100 Nodes used