بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

International Islamic University Chittagong

Department of Computer Science and Engineering

Course Title- Numerical Methods Sessional
Course Code: CSE-4746
Section: 7AM Semester, **Session:** Spring 2023

A **Project Report** on

System of Linear Equations

Author:

Team Name	Member's (Student's) Details
Team Crammer's Fan	**Team Leader:** C201042 - Jobair Uddin Jesan, 7AM **Deputy Leader:** C201041 - Emdadul Islam, 7AM **Member- 01:** C201032 – Sorowar Mahabub, 7AM **Member- 02:** C191039 – Sakif Azwad, 7AM *Dept. of* ***Computer Science and Engineering***, IIUC

Course Instructor || Submitted To

Prof. Mohammed Shamsul Alam
Professor, Computer Science and Engineering, IIUC
Mobile: 01711941680 **Mail:** alam_cse@yahoo.com , msa@iiuc.ac.bd

Completion Date: Thursday, 14 December, 2023

Abstract

This project explores the theoretical and practical aspects of solving linear systems using a select set of numerical methods: Matrix Inversion, Cramer's Rule, Gauss Elimination, and Jacobi Iterations. We delve into the implementation, analysis, and comparison of these techniques to gain a comprehensive understanding of their strengths and weaknesses, while applying them to real-world problems. Through this exploration, we aim to master the art of solving linear systems and contribute to the advancement of numerical methods.

Author: Team Crammer's Fan

1. Introduction

Linear equations form the backbone of numerous scientific and engineering disciplines. Solving systems of these equations often requires efficient numerical techniques, particularly for large and complex systems. This project focuses on exploring four key numerical methods for solving such systems, honing our problem-solving skills and gaining valuable insights into their practical applications.

2. Objectives

Comprehensive Understanding: Develop a thorough understanding of Matrix Inversion, Cramer's Rule, Gauss Elimination, and Jacobi Iterations for solving linear systems.

Algorithm Implementation: Implement and analyze these chosen methods, ensuring their accuracy and efficiency.

Performance Evaluation: Evaluate each method in terms of accuracy, computational cost, and convergence properties.

Real-World Applications: Apply these methods to solve linear systems arising from diverse fields like engineering, economics, and physics.

Comparison and Analysis: Compare and contrast the advantages and disadvantages of each method, identifying their suitability for different scenarios.

Numerical Stability: Investigate potential stability issues associated with large and ill-conditioned systems.

Documentation and Reporting: Document the project's findings, methodologies, and results in a clear and organized manner.

3. Description

The project is a web application that helps users solve systems of linear equations using various methods. The user can input the coefficients of the equations and the values of the right-hand side, and the application will then calculate the solutions using Cramer's rule, Gauss elimination, inverse matrix method, and Jacobi method. The application also displays the number of iterations and the time taken for each method.

A breakdown of its functionalities:

1. Analyzing the Matrix:

- The analyzeMatrix function takes the matrix values and the right-hand side vector as input.
- It checks if the matrix size is valid and calculates the determinant.
- It then calls different solution methods like Cramer's Rule, Gauss Elimination, and Jacobi Method.
- The results are stored in global arrays X, Y, and Z for each method.
- The function also displays the solution and performance metrics (iterations and time) in the log and charts.

2. Solution Methods:

- cramer function uses Cramer's Rule to solve the system and stores the solution in X, Y, and Z.
- gaussElimination function performs Gauss Elimination and stores the solution in X, Y, and Z.
- inverseMatrixs function calculates the inverse matrix and uses it to solve the system. The solution is stored in X, Y, and Z.
- jacobiMethod implements the Jacobi iterative method and stores the solution (if convergence is reached) in X, Y, and Z.

3. User Interface and Charts:

- The code defines functions for submitting the form, resetting the form, randomizing input values, and showing/hiding the team information.
- It also defines functions for displaying the solution, iterations, and time taken by each method in various formats like text, tables, and charts.
- **Two charts are generated:** one for comparing the solutions (X, Y, and Z) and another for comparing the number of iterations.
- A **bar chart** is also generated to compare the execution time of each method.

Overall, this code appears to be a comprehensive tool for analyzing and comparing different solutions for linear systems of equations. It provides detailed information about the solutions, performance metrics, and visualizations for better understanding.

4. Project Scope

This project focuses on the following four numerical methods:

- **Matrix Inversion:** Solve systems by directly inverting the coefficient matrix.
- **Cramer's Rule:** Solve systems by calculating determinants and ratios of determinants.
- **Gauss Elimination:** Solve systems by systematically eliminating unknowns through row operations.
- **Jacobi Iterations:** Solve systems iteratively by updating each unknown based on the latest values of other unknowns.

5. Key Features of The Project

- ✓ **Solves systems of linear equations:** The application can solve systems of equations with up to three variables.
- ✓ **Multiple methods:** The application provides four different methods for solving the equations: Cramer's rule, Gauss elimination, inverse matrix method, and Jacobi method.
- ✓ **Iteration and time analysis:** The application displays the number of iterations and the time taken for each method.
- ✓ **Visualization:** The application displays the solutions in a table and a chart.

Overall, this project is a useful tool for anyone who needs to solve systems of linear equations. It is easy to use and provides a variety of features that can be helpful for understanding and solving these equations.

6. Methodology

1. **Conceptual Understanding:** Gain a thorough understanding of the chosen methods through theoretical study and analysis.
2. **Implementation:** Develop and implement code for each method, ensuring accuracy and efficiency.
3. **Method Comparison:** Apply each method to a set of diverse linear systems and compare their performance in terms of accuracy, computational cost, and convergence properties.
4. **Testing and Validation:** Rigorously test and validate the implemented methods using various test cases.

7. Source Code (*partial, only JS file*)

```
let X = [];
let Y = [];
let Z = [];
```

```
let crammer_itr = 1;
let jacobi_itr  = 0;
let gauss_itr = 0;
```

```
let inverse_itr = 0;

let crammer_time = 0;
let jacobi_time  = 0;
let gauss_time = 0;
let inverse_time = 0;



function analyzeMatrix(matrixValues, b) {
 if (matrixValues.length !== 9) {
   document.getElementById("errorMsg").innerHTML    =
"Invalid matrix size.";
   return;
 }
 const det = [];

 const extra_det = [];
 for (let i = 0; i < 3; i++) {
   det.push(matrixValues.slice(i * 3, (i + 1) * 3));
   extra_det.push(matrixValues.slice(i * 3, (i + 1) * 3));
 }

 const extra = det;

 for (let i = 0; i < 3; i++) {
  extra[i].push(b[i]);
 }
 // console.table(extra)
 const matrixB = [[b[0]], [b[1]], [b[2]]];

 let extras = extra;

 showLogMsg("Inverse Matrix sol: ", inverseMatrixs(extras,
matrixB));

 crammer(extra_det, b);

 gaussElimination(extra);

 // jacobi
 const x = [0, 0, 0];

 const tolerance = 0.0001;
 const maxIterations = 100;

 const jacobi = jacobiMethod(extra_det, b, x, tolerance,
maxIterations);

 if (jacobi !== null) {
  X.push(jacobi[0]);
  Y.push(jacobi[1]);
  Z.push(jacobi[2]);
  showLogMsg("Jacobi Method", jacobi);
 } else {
  X.push(0);
  Y.push(0);
  Z.push(0);
 }

 showChart(X, Y, Z);
 showIteration();
 showTimeline();

 console.log(jacobi_itr + " < > " + gauss_itr + " < > " +
crammer_itr + " < > " + inverse_itr );
 console.log(jacobi_time + " < > " + gauss_time + " < >
" + crammer_time + " < > " + inverse_time );
}

function crammer(det, B) {
 /// perfectly choltese
 const startTime = performance.now();
 Ans_D0 =
  det[0][0] * (det[1][1] * det[2][2] - det[2][1] * det[1][2])
-
  det[0][1] * (det[1][0] * det[2][2] - det[2][0] * det[1][2])
+
  det[0][2] * (det[1][0] * det[2][1] - det[2][0] * det[1][1]);
 //For X:
 Ans_D1 =
  B[0] * (det[1][1] * det[2][2] - det[2][1] * det[1][2]) -
  det[0][1] * (B[1] * det[2][2] - B[2] * det[1][2]) +
  det[0][2] * (B[1] * det[2][1] - B[2] * det[1][1]);
 //For Y:
 Ans_D2 =
  det[0][0] * (B[1] * det[2][2] - B[2] * det[1][2]) -
  B[0] * (det[1][0] * det[2][2] - det[2][0] * det[1][2]) +
  det[0][2] * (det[1][0] * B[2] - det[2][0] * B[1]);
 //For Z:
 Ans_D3 =
  det[0][0] * (det[1][1] * B[2] - det[2][1] * B[1]) -
  det[0][1] * (det[1][0] * B[2] - det[2][0] * B[1]) +
  B[0] * (det[1][0] * det[2][1] - det[2][0] * det[1][1]);

 if (Ans_D0 != 0) {
  x = Ans_D1 / Ans_D0;
  y = Ans_D2 / Ans_D0;
  z = Ans_D3 / Ans_D0;
  //cout << Ans_D0 <<  " "  << Ans_D1 <<  " " <<
Ans_D2 <<  " "  <<  Ans_D3 << endl; // to check
  const solution = [x, y, z];
  const EndTime = performance.now();
  crammer_time = EndTime - startTime;
  X.push(x);
  Y.push(y);
  Z.push(z);

  showLogMsg("Cramer's Rule", solution);

 } else {
  const                    errorMsg                    =
document.getElementById("errorMsg");
  errorMsg.innerHTML = "There are Infinite solutions or
NO solution.";
 }
}
```

```
function gaussElimination(matrix) {
  const startTime = performance.now();
  const n = matrix.length;
  for (let i = 0; i < n; i++) {
    let divisor = matrix[i][i];
    for (let j = i; j < n + 1; j++) {
      matrix[i][j] /= divisor;
      gauss_itr++; // itrattion
    }
    for (let k = 0; k < n; k++) {
      if (k !== i) {
        let factor = matrix[k][i];
        for (let j = i; j < n + 1; j++) {
          matrix[k][j] -= factor * matrix[i][j];
          gauss_itr++; // iteration
        }
      }
    }
  }
  let solution = new Array(n);
  for (let i = 0; i < n; i++) {
    solution[i] = matrix[i][n];
  }

  X.push(solution[0]);
  Y.push(solution[1]);
  Z.push(solution[2]);

  showLogMsg("Gauss Elimination : ", solution);
  const EndTime = performance.now();
  gauss_time = EndTime - startTime;
  return solution;
}

// function matrixMultiplication(matrixA, matrixB) {
//   let result = [];
//   for (let i = 0; i < 3; i++) {
//     result[i] = [];
//     for (let j = 0; j < 3; j++) {
//       result[i][j] = 0;
//       for (let k = 0; k < 3; k++) {
//         result[i][j] += matrixA[i][k] * matrixB[k][j];
//       }
//     }
//   }
//   return result;
// }

function multiply(matrix1, matrix2) {
  if (
    matrix1.length !== 3 ||
    matrix1[0].length !== 3 ||
    matrix2.length !== 3 ||
    matrix2[0].length !== 1
  ) {
    document.getElementById("logMsg").innerHTML =
      "Invalid matrix dimensions for multiplication.";
    return null;
  }

  let result = [[0], [0], [0]];

  for (let i = 0; i < 3; i++) {
    for (let j = 0; j < 3; j++) {
      result[i][0] += matrix1[i][j] * matrix2[j][0];
      inverse_itr++;
    }
  }

  X.push(result[0][0]);
  Y.push(result[1][0]);
  Z.push(result[2][0]);

  return result;
}

function inverseMatrixs(det, b) {
  const startTime = performance.now();
  const n = det.length;

  let inverseMatrix = [];
  for (let i = 0; i < n; i++) {
    inverseMatrix[i] = [];
    for (let j = 0; j < n; j++) {
      inverseMatrix[i][j] = 0;
    }
  }
  let Ans = 0.0;
  for (let k = 0; k < n; k++) {
    Ans +=
      det[0][k] *
      (det[1][(k + 1) % n] * det[2][(k + 2) % n] -
        det[1][(k + 2) % n] * det[2][(k + 1) % n]);
        inverse_itr++
  }

  let d = Ans;

  //document.getElementById("logMsg").innerHTML    =
"Determinant: " + d;
  const logMsg = document.getElementById("logMsg")
  const h1 = document.createElement("h1")
  h1.innerHTML= "Determinant: " + d;
  h1.style = "font-weight:bold";
  h1.style.color = "blue"
  logMsg.appendChild(h1);
  if (d !== 0) {
    for (let i = 0; i < n; i++) {
      for (let j = 0; j < n; j++) {
        inverseMatrix[i][j] =
          (det[(j + 1) % n][(i + 1) % n] * det[(j + 2) % n][(i
+ 2) % n] -
            det[(j + 1) % n][(i + 2) % n] * det[(j + 2) % n][(i
+ 1) % n]) /
          d;
```

```
        inverse_itr++
      }
    }
  } else {
    console.table("Inverse doesn't exist for this matrix");
  }
  // X = A-I * B;
  // console.table(b);
  let XX = multiply(inverseMatrix, b);
  const EndTime = performance.now();
  inverse_time = EndTime - startTime;
  return XX;
}

function jacobiMethod(matrix, b, initial, tolerance, maxIterations) {
  const startTime = performance.now();
  const n = matrix.length;
  let x = initial.slice();

  for (let itr = 0; itr < maxIterations; itr++) {
    const xNext = [];

    for (let i = 0; i < n; i++) {
      let sum = b[i];
      for (let j = 0; j < n; j++) {
        if (i !== j) {
          sum -= matrix[i][j] * x[j];
        }

      }
      xNext[i] = sum / matrix[i][i];
    }

    // Check for convergence
    let maxDiff = 0;
    for (let i = 0; i < n; i++) {
      const diff = Math.abs(xNext[i] - x[i]);
      if (diff > maxDiff) {
        maxDiff = diff;



      }
    }


    if (maxDiff < tolerance) {
      console.log(`Converged in ${itr + 1}itr `);
      jacobi_itr = itr + 1;
      const EndTime = performance.now();
      jacobi_time = EndTime - startTime;
      return xNext;
    }

    x = xNext.slice();
  }
  const logMsg = document.getElementById("logMsg");
  const p = document.createElement("p");
  p.style = "font-weight: bold";
  p.style.color = "red";
  p.innerHTML = "Jacobi: Did not converge within " + maxIterations + " iter";

  logMsg.appendChild(p);

  return null;
}

// Main program

function submitForm(event) {
  event.preventDefault();

  X = [];
  Y = [];
  Z = [];

  const matrixValues = [];
  for (let i = 1; i <= 12; i++) {
    if (i === 4 || i === 8 || i === 12) continue;
    const inputValue = document.getElementById(`input${i}`).value;
    matrixValues.push(inputValue);
  }
  const b = [];
  for (let i = 1; i <= 12; i++) {
    if (i !== 4 && i !== 8 && i !== 12) continue;
    const inputValue = document.getElementById(`input${i}`).value;
    b.push(inputValue);
  }

  analyzeMatrix(matrixValues, b);
}

function resetForm() {
  for (let i = 1; i <= 12; i++) {
    document.getElementById(`input${i}`).value = "";
  }
  X = [];
  Y = [];
  Z = [];
  showChart(X, Y, Z);
  resetLog();
}

function randomize() {
  for (let i = 1; i <= 12; i++) {
    if (i === 4 || i === 8 || i === 12) continue;
    document.getElementById(`input${i}`).value = Math.floor(
      Math.random() * 100
    );
  }
  for (let i = 1; i <= 12; i++) {
    if (i !== 4 && i !== 8 && i !== 12) continue;
```

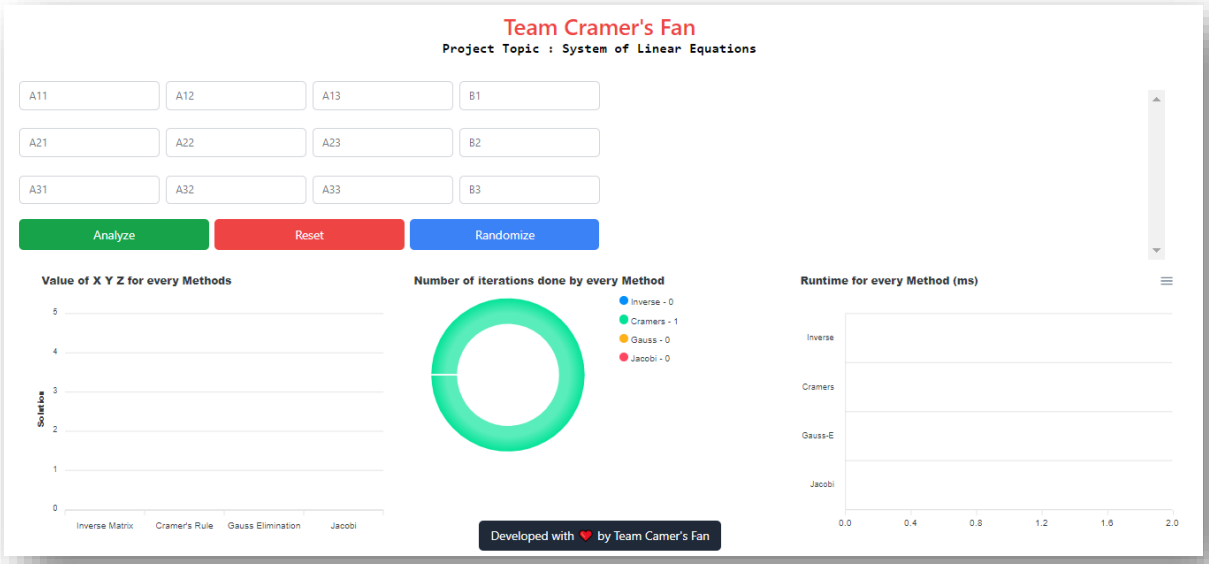
```
    document.getElementById(`input${i}`).value =
Math.floor(
      Math.random() * 100
    );
  }
}

function showLogMsg(title, matrix) {
  const logMsg = document.getElementById("logMsg");
  const p = document.createElement("p");
  p.style = "font-weight: bold";
  p.innerHTML = title;
  logMsg.appendChild(p);

  const table = document.createElement("table");
  const row = document.createElement("tr");

  for (let i = 0; i < matrix.length; i++) {
    const col = document.createElement("td");
    col.style = "padding: 6px";
    col.innerHTML = matrix[i];
    row.appendChild(col);
    table.appendChild(row);
  }

  logMsg.appendChild(table);

  const hr = document.createElement("hr");
  logMsg.appendChild(hr);

  logMsg.scrollTop = logMsg.scrollHeight;
}

function resetLog() {
  const logMsg = document.getElementById("logMsg");
  logMsg.innerHTML = "";
}

function closeModal() {
  document.getElementById("team").style.display = "none";
}

function showTeam() {
  document.getElementById("team").style.display = "block";
}

function showChart(X, Y, Z) {
  X.map((x, i) => {
    X[i] = x.toFixed(6);
  });

  Y.map((y, i) => {
    Y[i] = y.toFixed(6);
  });

  Z.map((z, i) => {
    Z[i] = z.toFixed(6);
  });
```

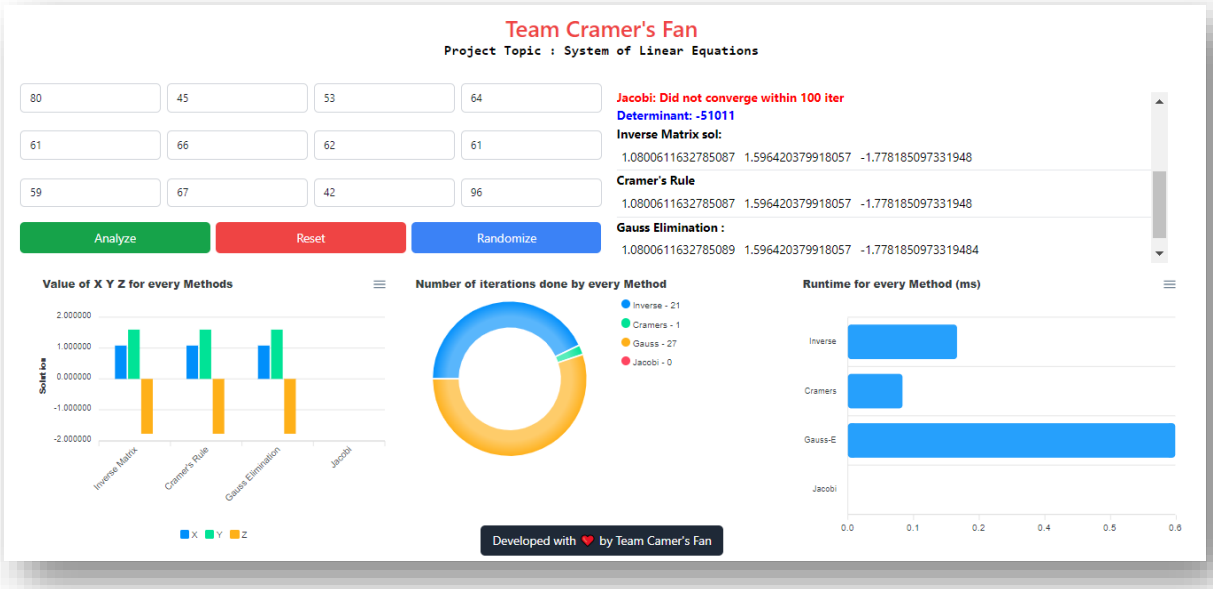
```
var options = {
  series: [
    {
      name: "X",
      data: X,
    },
    {
      name: "Y",
      data: Y,
    },
    {
      name: "Z",
      data: Z,
    },
  ],
  chart: {
    type: "bar",
    height: 350,
  },
  plotOptions: {
    bar: {
      horizontal: false,
      columnWidth: "55%",
      endingShape: "rounded",
    },
  },
  dataLabels: {
    enabled: false,
  },
  stroke: {
    show: true,
    width: 2,
    colors: ["transparent"],
  },
  xaxis: {
    categories: [
      "Inverse Matrix",
      "Cramer's Rule",
      "Gauss Elimination",
      "Jacobi",
    ],
  },
  title: {
    text: 'Value of X Y Z for every Methods'
  },
  yaxis: {
    title: {
      text: "Solution",
    },
  },
  fill: {
    opacity: 1,
  },
  tooltip: {
    y: {
      formatter: function (val) {
        return val;
```

```
      },
     },
    },
   };

  var chart = new ApexCharts(document.getElementById("myChart"), options);
  chart.render();
}


function showIteration(){

  var options = {
    series: [inverse_itr, crammer_itr, gauss_itr, jacobi_itr],
    chart: {
    width: 380,
    type: 'donut',
  },
  plotOptions: {
    pie: {
      startAngle: -90,
      endAngle: 270
    }
  },
  dataLabels: {
    enabled: false
  },
  fill: {
    type: 'gradient',
  },
  legend: {
    formatter: function(val, opts) {
      var legendNames = ['Inverse', 'Cramers', 'Gauss', 'Jacobi'];
      return legendNames[opts.seriesIndex] + " - " + opts.w.globals.series[opts.seriesIndex];
    }
  },
  title: {
    text: 'Number of iterations done by every Method'
  },
  responsive: [{
    breakpoint: 480,
    options: {
      chart: {
        width: 200
      },
      legend: {
        position: 'bottom'
      }
    }
  }]
  };

  var chart = new ApexCharts(document.querySelector("#pieChart"), options);
  chart.render();

}


function showTimeline(){
  var options = {
    series: [{
    data: [inverse_time, crammer_time, gauss_time, jacobi_time]
  }],
    chart: {
    type: 'bar',
    height: 350
  },
  plotOptions: {
    bar: {
      borderRadius: 4,
      horizontal: true,
    }
  },
  title: {
    text: 'Runtime for every Method (ms)'
  },
  dataLabels: {
    enabled: false
  },
  xaxis: {
    categories: ['Inverse', 'Cramers', 'Gauss-E', 'Jacobi'],
  }
  };

  var chart = new ApexCharts(document.querySelector("#barChart"), options);
  chart.render();



}

showChart(X, Y, Z);
  showIteration();
  showTimeline();
```

Full Source is available in GitHub Repository (https://github.com/mdadul/TeamCramersFan).

8. Output





9. Conclusions:

This project promises to equip us with a mastery of solving linear systems using a focused set of numerical methods. We will gain valuable insights into their performance, limitations, and real-world applications. By pursuing this project, we aim to contribute to the advancement of numerical methods and enhance our problem-solving skills in this crucial field.

10. Future Work (*Pushing the Boundaries of Linear Systems Analysis*)

While this application provides a robust platform for analyzing and comparing solutions to linear systems, the journey doesn't end here. We envision a future where this tool expands its capabilities and delves deeper into the fascinating world of linear algebra. Here's a glimpse into what's to come:

1. Conquering Larger Dimensions: We aim to break free from the limitations of 3x3 matrices and empower you to tackle systems of any size. Imagine effortlessly analyzing complex real-world problems with dozens of equations and variables.

2. Embracing the Realm of Complex Numbers: The world of mathematics isn't confined to real numbers. We're on a mission to equip this application with the ability to handle complex coefficients and solutions, allowing you to explore the full spectrum of linear systems.

3. Unveiling the Nuances of Errors: Precision is paramount. We'll refine the application to provide detailed error estimates for each solution method, ensuring you have the confidence to trust the results.

4. Efficiency Through Parallelization: Time is precious. We'll leverage the power of multi-core processors by implementing parallelization techniques, significantly reducing execution times for complex calculations.

5. Interactive Learning and Exploration: Knowledge is power. We'll transform the application into an interactive learning platform, offering step-by-step visualizations, customizable options, and educational resources to deepen your understanding of linear systems.

6. A World of Possibilities: We envision connecting this application to external solvers and libraries, opening doors to advanced features and pushing the boundaries of what's possible.

7. Beyond the Desktop: The world is mobile. We'll develop a mobile-friendly version of the application, making your linear algebra toolbox accessible anytime, anywhere.

This is just the beginning of our exciting journey. We invite you to join us as we continue to evolve this application into a powerful and versatile tool for anyone who wants to conquer the challenges of linear systems. Stay tuned for future updates, and let's unlock the secrets of linear algebra together!

11. References:

[1] Book: "Numerical Analysis" by G. Shanker Rao (5th Edition).

[2] Course Notes and Lecture Slides provided by our honorable course teacher.
(https://t.me/c/1653334055/30 and https://t.me/c/1653334055/88)

12. Project Live Link (Desktop View): https://team-crammers-fan.vercel.app/

The End! Assalamulaikum Waa Rahmatullh.

Author: Team Crammer's Fan