Códigos

- main.js

```
const { Matrix, inverse } = require('ml-matrix');
const io = require('console-read-write');
const fs = require('fs');
const util = require('util');
const Jacobi = require('./methods/Jacobi');
const Potency = require('./methods/Potency');
const createOutputFile = require('./utils/createOutputFile');

util.inspect.defaultOptions.depth = null;

async function main() {
  console.log('Digite o nome do arquivo:(ele deve estar na mesma pasta do executavel)');
  const fileName = await io.read();
  // const fileName = 'example.txt';

  let buffer;
  try {
    buffer = fs.readFileSync(`./${fileName}`);
  } catch (error) {
    console.log('Arquivo não encontrado!');
    return;
  }

  const bufferAsString = buffer.toString();

  let [
    n,
    ICOD,
    IDET,
    ...rest
  ] = bufferAsString.split('\n');

  const shouldCalculateDeterminant = !parseInt(IDET);
  ICOD = parseInt(ICOD);
  n = parseInt(n);

  const matrix_elements = rest.slice(0, n);
  const TOLm = rest[n] || -1;

  const matrixA = matrix_elements.map((line) => line.split(' ').map((a) => parseFloat(a)));
  try {
    const matrix = new Matrix(matrixA);
    inverse(matrix);
  } catch (error) { console.log({ error });
    console.log('Essa matriz é singular!');
    return;
  }

  const params = {
    n,
    matrixA,
    shouldCalculateDeterminant,
    tol: Math.pow(10, TOLm),
  };

  console.log({ params });

  let answer;
  let method;

  switch(ICOD) {
    case 1:
      answer = Potency(params);
      method = 'Potência';
      break;
    case 2:
      answer = Jacobi(params);
      method = 'Jacobi';
      break;
    default:
      break;
  }

  createOutputFile(answer, ICOD, method);
}

main().then(response => console.log(response));
```

- utils/createEmptyMatrix.js

```
function createMatrix(n,m) {
  const matrix = [];
  for (let i = 0; i < n; i++) {
    matrix.push([]);
    for (let j = 0; j < m; j++) {
      matrix[i].push(0);
    }
  }
  return matrix;
}

module.exports = createMatrix;
```

- utils/createOutputFile.js

```
const fs = require('fs');

function createOutputFile(answer, ICOD, method) {
  let fileString = `Resolvido pelo método ${method}\n`;

  const {
    matrixX,
    autoValores,
    autoVetores,
    iterations,
    errors,
    determinant,
  } = answer || {};

  fileString += 'AutoVetores:\n';
  if (matrixX) {
    matrixX.forEach((line) => {
      fileString += `${line.join(' ')}\n`;
    });
  } else if (autoVetores) {
    console.log({ autoVetores })
    autoVetores.forEach((autoVetor) => {
      fileString += `${autoVetor.join(' ')}\n`;
    });
  }

  fileString += 'AutoValores:\n';
  autoValores.forEach((autoValor) => {
    if(!Number.isNaN(autoValor)) fileString += `${autoValor}\n`;
  });

  if (errors && errors.length) {
    fileString += 'Erros:\n';
    errors.forEach((error) => {
      fileString += `${error}\n`;
    });
  }
  if (determinant) {
    fileString += `Determinante: ${determinant}\n`;
  }
  fileString += `Iterações: ${iterations}\n`;

  console.log({ file: fileString });
  fs.writeFileSync('answer.txt', fileString);
}

module.exports = createOutputFile;
```

- utils/isSymetric.js

```
function isSymetric(matrix) {
  for (let i = 0; i < matrix.rows; i++) {
    for (let j = 0; j <= i; j++) {
      if (matrix[i][j] !== matrix[j][i]) {
        return false;
      }
    }
  }
  return true;
}

module.exports = isSymetric;
```

- utils/LUHelper.js

```
const createEmptyMatrix = require('./createEmptyMatrix');

function determinantLU(matrix, pivot) {
  const n = matrix.length;
  let det = 1;
  for (let i = 0; i < n; i++) {
    det *= matrix[i][i];
  }

  if (pivot % 2) {
    console.log('Inverteu o sinal do det');
    det = -det;
  }

  console.log({ det });
  return det;
}

function LUDecomposition(n, matrixA, shouldCalculateDeterminant) {

  let pivot = 0;
  const matrixL = createEmptyMatrix(n,n);
  for (let i = 0; i < n; i++) matrixL[i][i] = 1;

  for (let k = 0; k < n - 1; k++) {
    // agora vou percorrer todas as linhas e pegar o maior valor na posição k,k
    let biggerValue = matrixA[k][k];
    let biggerLine = k;
    for (let i = k + 1; i < n; i++) {
      if (Math.abs(biggerValue) < Math.abs(matrixA[i][k])) {
        biggerValue = matrixA[i][k];
        biggerLine = i;
      }
    }

    //se biggerLine nao for k, trocar essas linhas
    if (biggerLine !== k) {
      const temp = matrixA[k];
      matrixA[k] = matrixA[biggerLine];
      matrixA[biggerLine] = temp;
      pivot++;
    }

    for (let i = k + 1; i < n; i++) {
      const m = - matrixA[i][k]/matrixA[k][k];
      matrixL[i][k] = -m;
      for (let j = k; j < n; j++) {
        matrixA[i][j] = m * matrixA[k][j] + matrixA[i][j];
      }
    }
  }

  return {
    matrixL,
    matrixU: matrixA,
    determinant: shouldCalculateDeterminant ? determinantLU(matrixA, pivot) : undefined,
  };
}

module.exports = LUDecomposition;
```

- utils/transpose.js

```
const createEmptyMatrix = require('./createEmptyMatrix');

function transpose(matrix) {
  const n = matrix.length;
  const transposed = createEmptyMatrix(n,n);

  for (let i = 0; i < n; i++) {
    for (let j = 0; j < n; j++) {
      transposed[i][j] = matrix[j][i];
    }

  }
  return transposed;
}

module.exports = transpose;
```

- utils/mult.js

```
const { Matrix } = require('ml-matrix');
const createEmptyMatrix = require('./createEmptyMatrix');

function mult(A, B) {
  const n = A.length;
  const matrixA = new Matrix(A);
  const matrixB = new Matrix(B);

  const mul = matrixA.mmul(matrixB);

  console.log({ matrixA, matrixB, mul });

  const answer = createEmptyMatrix(n,n);

  for (let i = 0; i < n; i++) {
    for (let j = 0; j < n; j++) {
      answer[i][j] = mul.get(i,j);
    }
  }

  return answer;
}

module.exports = mult;
```