

Universidade Federal do Ceará Campus Crateús

Ciência da Computação Álgebra Linear

Relatório da Implementação do Trabalho de Álgebra

Rafael Vieira Barbosa – 418812

Natanael Vieira Barbosa – 391028

**Crateús - CE 2018**

Sumário

[Introdução 3](#_Toc26000153)

[Contexto Histórico 3](#_Toc26000154)

[Aplicações 4](#_Toc26000155)

[Dupla 4](#_Toc26000156)

[Estrutura do Sistema 5](#_Toc26000157)

[Algoritmos 5](#_Toc26000158)

[Matrizes 5](#_Toc26000159)

[Sistemas 5](#_Toc26000160)

[Gram-Schmidt 7](#_Toc26000161)

[Conclusão 9](#_Toc26000162)

[Objetivos do trabalho 9](#_Toc26000163)

[Dificuldades encontradas 9](#_Toc26000164)

[Aprendizado 9](#_Toc26000165)

[Limitações encontradas pela tecnologia usada 9](#_Toc26000166)

[Decisões tomadas 9](#_Toc26000167)

[Entre outros que se julgue importante. 9](#_Toc26000168)

[Referências 9](#_Toc26000169)

# Introdução

## Contexto Histórico

O estudo e aplicações com as matrizes e seus determinantes tiveram início no século II a.C. Segundo estudos, os babilônios ao analisar certos problemas, chegaram a resoluções obtidas pelos sistemas com duas variáveis. Já os primeiros exemplos do uso de matrizes, foi apresentado pelos chineses, em no texto “Nove capítulos da arte Matemática”, escrito durante a dinastia Han, aproximadamente entre 200 e 100 a.C.

Muitos resultados da teoria de matrizes elementares apareceram, antes mesmo das matrizes serem realmente estudadas. Por exemplo, de Witt em sua publicação Elementos de Curvas, mostrou uma operação muito comum nas matrizes: a diagonalização de uma matriz. Nessa publicação, ele usava uma cônica para transformar na forma canônica, utilizando transformações de eixos. Outro exemplo, é ideia de determinante que apareceu no Japão e na Europa.

Porém, foi a partir do século XVIII que começou a estudar-se determinantes, como por exemplo: em 1764, quando Bézout apresentou os determinantes de Vandermonde; e em 1801, onde Gauss discutiu sobre formas quadráticas.

Perto da metade do século XIX, o matemático Arthur Cayley foi o pioneiro a estudar matrizes, definindo matriz nula e matriz identidade a partir de operações sobre as mesmas, e apresentou uma ideia sobre matriz inversa. Ele usou as matrizes, em seus artigos, como uma simples forma de representar valores e no estudo de transformações em equações lineares. Por exemplo, ele introduziu o conceito de multiplicação de matrizes, e percebeu que não era comutativa como as outras operações. Anos depois, o mesmo introduziu as operações de adição de matrizes e multiplicação de matrizes por escalares, além colocar suas propriedades.

Já a ideia de sistemas lineares surgiu no Oriente, onde se tem relatos d que os chineses utilizavam desse meio 111 a.C. Mas foi só no século XVII, no trabalho do japonês Seki Kowa, que criou a ideia de determinante (como polinômio que se associa a um quadrado de números), no qual utilizou do estudo de sistemas lineares, para conseguir isso.

## Aplicações

Podemos utilizar Sistemas Lineares na Física, para calcular a corrente elétrica, juntamente com as leis que se aplicam a circuitos elétricos, como as Leis de Kirchhooff e a Lei de Ohm.

Outra área em que esses conceitos são usados, é na da Economia, no qual podemos utilizar de interpretação gráfica dos dados, ou até mesmo para controlar a lucratividade da empresa, por exemplo, o economista Russo Wassily, em seu trabalho de modelagem econômica, utiliza métodos matriciais para analisar as relações em diferentes setores da economia. Em um de seus modelos, ele tenta determinar os preços dos produtos de uma empresa, para equilibrar sua economia. Nesse modelo, ele utiliza de *n* indústrias, e cada indústria produz uma quantidade fixa de um produto ou serviço que é usado pelas *n* indústrias.

Ainda mais, pode-se utilizar os conhecimentos sobre matrizes na Criptografia. Um exemplo prático seria criptografar um texto. Para isso, é necessário criar uma matriz codificadora e outra decodificadora, com números inteiros, onde a matriz decodificadora é a inversa da codificadora. Além disso, precisaríamos criar representantes numéricos para o nosso alfabeto, e depois criar uma matriz com os representantes do texto a ser criptografado. Assim para criptografar uma mensagem, multiplicaríamos a matriz do texto pela matriz codificadora, gerando uma matriz codificada. Para descriptografar, multiplicaríamos a matriz codificada pela matriz decodificadora.

# Dupla

Divisão do trabalho:

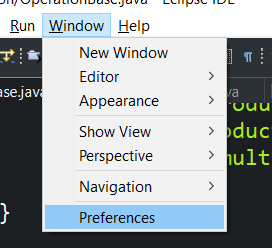
|  |  |
| --- | --- |
| Tabela 1 - Divisão das Atividades do Trabalho | |
| Tópico | **Integrante** |
| Álgebra Matricial | Rafael |
| Sistemas Lineares | Rafael |
| Processo de Ortogonalização | Rafael |
| Relatório | Rafael |
| Interface Gráfica | Natanael e Rafael |

Fonte: Elaborada pelo próprio autor

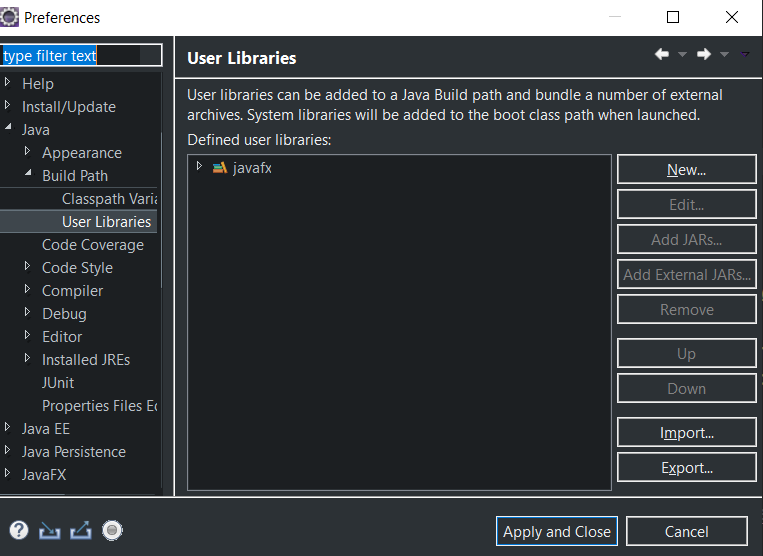
# Configurações do Eclipse para utilizar o Programa

Junto ao sistema, foi adicionado o JDK (Kit de Desenvolvimento Java) que é necessário para executar esse programa, caso o computador do usuário não tenha instalado. Caso sua máquina já tenha instalado, você não precisa ler esses passos.

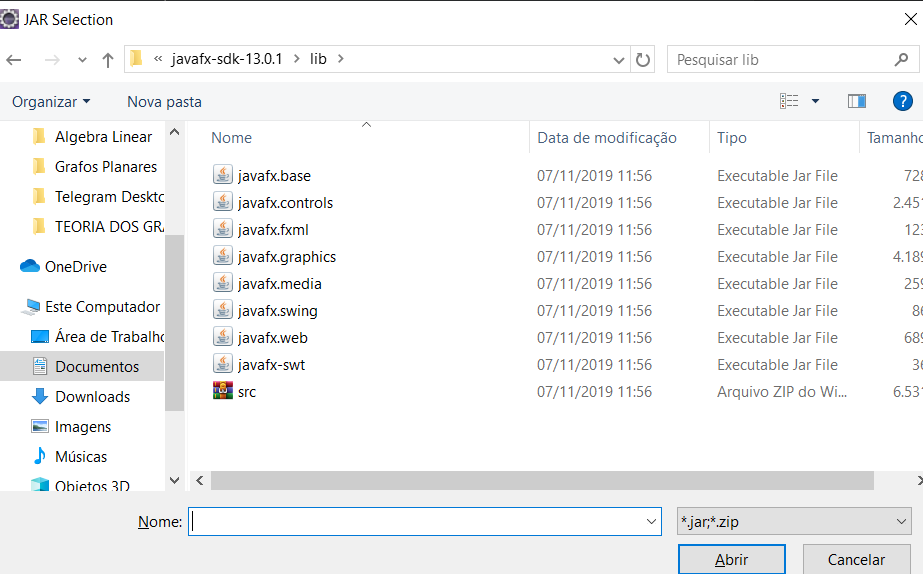
1. Após abrir o Eclipse, clique no menu Windows, e escolha Preferences.



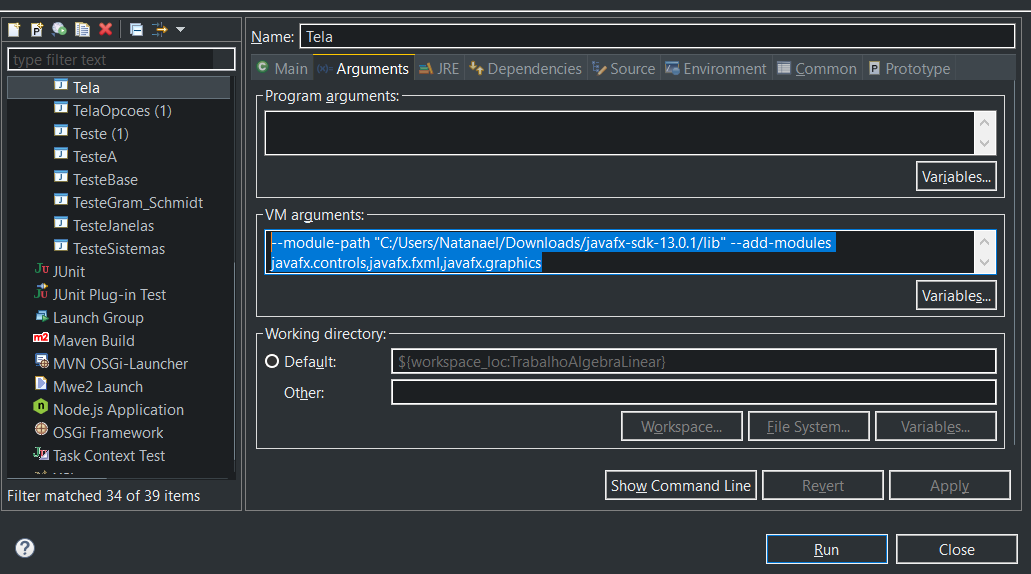
1. Depois, clique nas opções Java -> Build Path -> User Libraries :



1. Clique no botão New, escreva o nome “javafx”, e confirme.
2. Clique no botão Add External JARs, e vá até a pasta onde se encontra o JDK desse sistema, e entre na pasta “lib”. Depois, selecione todos os arquivos, exceto o arquivo zipado “src”. Após isso, clique em “Abrir”.



1. Clique em “Apply and Close”;
2. Agora, ainda no Eclipse, abra o pacote view, e clique com o botão direito sobre a classe “Tela.java”, e vá em Run As -> Run Configurations;
3. Clique na aba Arguments. Depois, escreva ‘--module-path "caminho/Trabalho-de-Algebra-Linear/javafx-sdk-13.0.1/lib" --add-modules javafx.controls,javafx.fxml,javafx.graphics’ no campo VM arguments. Obs.: retire as aspas simpes, troque a palavra “caminho”, pelo caminho onde se encontra a pasta “Trabalho-de-Algebra-Linear”:



1. Clique em Apply, e depois em Close.

Na primeira vez que for executar o programa, abra a classe Tela.java no Eclipse, clique com o botão direito sobre ela, e vá em Run As -> Java Application.

# Estrutura do Sistema

No trabalho foi decidido que usaríamos o JavaFx para desenvolver as telas. O código-fonte do programa foi dividido em 4 partes(pacotes): view, model, operation e controller.

O pacote view contém as telas do programa, além de seus elementos gráficos, como as abas e os botões.

O pacote model possui a representação dos elementos básicos para o programa. Essas estruturas são: Matriz, Sistema e Vetor. Basicamente, ela seguem o mesmo padrão: possui uma matriz de valores, o número de colunas e de linhas. Porém, cada uma foi implementada com características diferentes.

No pacote operation, são encontradas as implementações próprias de cada estrutura, como os métodos que escalonam sistemas, e que ortogonaliza uma base de vetores. Para se usar as instâncias das classes desse pacote usa se o método: getInstance(). Logo, quando as outras classes querem utilizar dos métodos implementados nas classes do Operation, usa-se a expressão “*OperationEstrutura.getInstance().operacao()*”, no qual, *OperationEstrutura* é classe correspondente (*Matriz, System ou Base*), e *operacao()* é a operação desejada. O mesmo vale para as classes do pacote controller.

No pacote controller, estão as operações para permitir as conversões de cada tipo das estruturas citadas anteriormente, para a o tipo de dado gráfico, além da construção de respostas. Por exemplo, quando deseja-se que os dados da Matriz resultante de alguma operação, sejam exibidas na tela, ou quando precisa-se resgatar os valores dos vetores digitados pelo usuário.

**Observação:** Quando a quantidade de campos da matriz ou do sistema ou da base é maior que a tela, cria-se uma barra de rolagem, tanto vertical quanto horizontal. O mesmo acontece na exibição do resultado, principalmente no Gauss e no Gauss-Jordan. A barra é quase da cor da tela, por isso pode ser um pouco difícil de vê-la.

# Algoritmos

## Matrizes

### Fatoração LU

FUNÇÃO fatoracaoLU (system : Sistema) : Matriz[]

DECLARE det : REAL

det = OperationMatriz.*getInstance*().determinante( new Matriz( system.getMatrizCoef() ) )

SE det == 0 FAÇA

IMPRIMA " Erro na Fatoração LU do sistema => A Matriz dos Coeficientes tem determinante igual a 0, logo não admite inversa e nem FatoraçãoLU!"

DECLARE matrixLower : REAL[][]

matrixLower = new double[system.getNumEq()][system.getNumIncog()];

PARA i = 0, ATÉ i < system.getNumEq(), FAÇA

matrixLower[i][i] = 1

PARA j = i, ATÉ j < system.getNumIncog() FAÇA

matrixLower[i][j] = 0

DECLARE matrixUpper: REAL[][]

matrixUpper = system.getMatrizCoef()

PARA i = 0, ATÉ i < system.getNumEq()-1 FAÇA

SE matrixUpper[i][i] == 0 FAÇA

SE rowsSwap(matrixUpper,i) == false FAÇA

continue;

PARA linha = i+1, ATÉ linha < system.getNumEq() FAÇA

SE matrixUpper[linha][i] == 0 FAÇA

continue;

DECLARE firstTermo : REAL

firstTermo = matrixUpper[linha][i] / matrixUpper[i][i]

matrixLower[linha][i] = firstTermo

PARA coluna = i, ATÉ coluna < system.getNumIncog()FAÇA

matrixUpper[linha][coluna] = matrixUpper[linha][coluna] - (firstTermo \* matrixUpper[i][coluna])

DECLARE lu : Matriz [2]

lu[0] = new Matriz(matrixU)

lu[1] = new Matriz(matrixL)

RETORNE lu;

### Determinante

FUNÇÃO determinante (matrix: Matriz) : double

SE !matrix.isSquare() FAÇA

IMPRIMA "Cálculo do Determinante: A matriz não é quadrada!"

RETORNE 0

SE matrix.getLinha() == 0 FAÇA

RETORNE 0

SEmatrix.getLinha() == 1) FAÇA

RETORNE matrix.value(0,0)

SE matrix.getLinha() == 2 FAÇA

RETORNE matrix.value(0,0) \* matrix.value(1,1) - matrix.value(0,1) \* matrix.value(1,0)

DECLARE newMatriz: Matriz [][]

newMatriz = new double[matrix.getLinha()][matrix.getColuna()+1];

PARA i = 0, ATÉ i < matrix.getLinha(), FAÇA

PARA j = 0, ATÉ j < matrix.getColuna(), FAÇA

newMatriz[i][j] = matrix.value(i, j);

newMatriz[i][matrix.getColuna()] = 0;

DECLARE diagonal: REAL[][]

diagonal = OperationSystem.getInstance().gauss( new Sistema( newMatriz, matrix.getLinha(), matrix.getColuna() ) ).getMatrizAmpliada();

DECLARE det : REAL

det = 1;

PARA i = 0 ATÉ i < diagonal.length FAÇA

det = det \* diagonal[i][i]

RETORNE det

## Sistemas

### Gauss

FUNÇÃO gauss (system : Sistema) : Sistema

SE system.getNumEq() <= 1 OU system.getNumIncog() <= 1, FAÇA

RETORNE system

DECLARE matrixAmp : REAL[][]

matrixAmp = system.getMatrizAmpliada()

PARA i = 0, ATÉ i < system.getNumEq()-1 AND i < system.getNumIncog() FAÇA

SE matrixAmp[i][i] == 0, FAÇA

SE !rowsSwap(matrixAmp,i)

CONTINUE

SE matrixAmp[i][i] != 1 FAÇA

DECLARE divisor: REAL

divisor = matrixAmp[i][i]

PARA j = 0, ATÉ j <= system.getNumIncog(), FAÇA

matrixAmp[i][j] = matrixAmp[i][j] / divisor

PARA linha = i+1, ATÉ linha < system.getNumEq(), FAÇA

DECLARE firstTermo: REAL

firstTermo = matrixAmp[linha][i]

SE firstTermo == 0 FAÇA

CONTINUE

FAÇA coluna = i, ATÉ coluna <= system.getNumIncog() ATÉ

matrixAmp[linha][coluna] = matrixAmp[linha][coluna] - (firstTermo \* matrixAmp[i][coluna])

RETORNE new Sistema( matrixAmp, system.getNumEq(), system.getNumIncog() )

### Troca de Linhas (rowsSwap)

FUNÇÃO rowsSwap(matrix : REAL[][], I : INTEIRO) : boolean

DECLARE j : INTEIRO

j = i+1;

ENQUANTO j < matrix.length E matrix[j][i] == 0 FAÇA

j = j+1;

SE j == matrix.length FAÇA

RETORNE false;

DECLARE aux : REAL

PARA coluna = 0, ATÉ coluna < matrix[0].length FAÇA

aux = matrix[i][coluna];

matrix[i][coluna] = matrix[j][coluna];

matrix[j][coluna] = aux;

RETORNE true;

### Gauss-Jordan

FUNÇÃO gaussJordan(system : Sistema) : Sistema

SE system.getNumEq() <= 1 OU system.getNumIncog() <= 1, FAÇA

RETORNE system

DECLARE matrixAmp : REAL[][]

matrixAmp = gauss(system).getMatrizAmpliada();

DECLARE i : INTEIRO

SE system.getNumEq() < system.getNumIncog(),FAÇA

i = system.getNumEq() - 1

SENÃO, FAÇA

i = system.getNumIncog() - 1

PARA i, ATÉ i >=0, FAÇA

SE matrixAmp[i][i] == 0 FAÇA

CONTINUE;

SE matrixAmp[i][i] != 1 FAÇA

DECLARE divisor : REAL

divisor = matrixAmp[i][i]

PARA j = 0, ATÉ j <= system.getNumIncog() FAÇA

matrixAmp[i][j] = divisor / matrixAmp[i][j]

PARA linha = i-1, ATÉ linha >= 0, FAÇA

DECLARE firstTermo: REAL

firstTermo = matrixAmp[linha][i];

SE firstTermo == 0 FAÇA

CONTINUE;

PARA coluna = system.getNumIncog(), ATÉ coluna >= 0, FAÇA

matrixAmp[linha][coluna] = matrixAmp[linha][coluna] - (firstTermo\*matrixAmp[i][coluna])

RETORNE new Sistema( matrixAmp, system.getNumEq(), system.getNumIncog() )

### Soluções

FUNÇÃO solucions(system : Sistema) : TEXTO

system = gaussJordan(system)

DECLARE analyzePost : INTEIRO

analyzePost = analyzePost(system);

DECLARE solucao: TEXTO

solucao = "";

SE analyzePost == 2 FAÇA

solucao = "Não existe solução!"

SE analyzePost == 0 FAÇA

PARA pos = 0, ATÉ pos < system.getNumEq() FAÇA

solucao += "x"+pos+" = "+system.getTermo(pos)+"\n"

SENÃO FAÇA

DECLARE matrix: REAL[][]

matrix = system.getMatrizAmpliada();

PARA pos = 0, ATÉ pos < system.getNumEq() E pos < system.getNumIncog() FAÇA

SE matrix[pos][pos] == 0 FAÇA

solucao = solucao + "x"+ pos + " = x" + pos + "\n"

SENÃO FAÇA

solucao = solucao + "x"+ pos+" = "+system.getTermo(pos) + returnCoef(matrix,pos) + "\n"

RETORNE solucao

## Gram-Schmidt

### Projeção

FUNÇÃO projection (v: Vetor, w:Vetor) : Vetor

DECLARE vw, ww : REAL

vw = productInternal(v, w)

ww= w.modulo()

DECLARE cw : Vetor

cw = w.multiplicaEscalar(vw/ww);

RETORNE cw;

### Ortogonalização

FUNÇÃO orthogonalization (base : Vetor[])

DECLARA lengthBase : INTEIRO

lengthBase = base[0].getNumCoordenadas()

DECLARA newBase: VETOR[]

newBase = new Vetor[lengthBase]

PARA posAtual = 0, ATÉ posAtual < lengthBase FAÇA

newBase[posAtual] = base[posAtual]

PARA anterior = 0, ATÉ anterior < posAtual FAÇA

newBase[posAtual] = subtraction( newBase[posAtual], projection( base[posAtual], base[anterior] ) )

RETORNE newBase

### Ortogonortomalização

FUNÇÃO orthonormalization (base : Vetor[]) : Vetor[]

DECLARA newBase : Vetor[]

Vetor[] newBase = orthogonalization(base);

PARA posAtual=0, ATÉ posAtual < newBase.length FAÇA

newBase[posAtual] = newBase[posAtual].normaliza();

RETORNE newBase;

# Conclusão

## Objetivos do trabalho

Esse trabalho teve como objetivo fixar os conteúdos estudados durante a disciplina de Álgebra Linear, além de desenvolver o trabalho em equipe. Para isso, foram implementados as principais operações sobre Matrizes, e Sistemas. Além disso foi desenvolvido um algoritmo para ortogonalizar e ortonormalizar um conjunto de vetores.

## Dificuldades encontradas

A principal dificuldade foi a desenvolver as telas, pois nenhum dos envolvidos não tinham experiência com JavaFX ou Swing. Além disso, tínhamos limitações físicas, por causa do notebook, que muitas vezes não suportava o Eclipse aberto.

# Referências

[https://repositorio.unesp.br/bitstream/hEle/11449/151612/levorato\_gbp\_me\_rcla.pdf?sequence=3](https://repositorio.unesp.br/bitstream/handle/11449/151612/levorato_gbp_me_rcla.pdf?sequence=3)

<http://matrizesss.blogspot.com/2015/05/comosurgiram-as-matrizes-as_12.html>

<http://matematicarapidaja.blogspot.com/2014/11/uma-breve-historia-das-matrizes-e.html>

[https://repositorio.ufsc.br/bitstream/hEle/123456789/96804/Cristini\_Kuerten.PDF?sequence=1](https://repositorio.ufsc.br/bitstream/handle/123456789/96804/Cristini_Kuerten.PDF?sequence=1)