**ANEXO A**

Abaixo está o código, comentado, sobre como é feito a simulação via MATLAB.

% Parâmetros

m\_s = 250; % Massa suspensa (kg) - 250 a 500 kg

m\_u = 50; % Massa não suspensa (kg) - 25 a 75 kg

k\_s = 15000; % Rigidez da suspensão (N/m) - 10 000 a 50 000 N/m

k\_t = 200000; % Rigidez do pneu (N/m) - 150 000 a 250 000 N/m

c\_s = 1000; % Amortecimento da suspensão (Ns/m) - 1 000 a 5 000 Ns/m

% Matrizes do Espaço de Estados

A = [0, 1, 0, 0;

-k\_s/m\_s, -c\_s/m\_s, k\_s/m\_s, c\_s/m\_s;

0, 0, 0, 1;

k\_s/m\_u, c\_s/m\_u, -(k\_s+k\_t)/m\_u, -c\_s/m\_u];

B = [0; 0; 0; k\_t/m\_u];

C = [1, 0, 0, 0; 0, 0, 1, 0];

D = [0; 0];

% Função do MATLAB (sys) para definir o Sistema das matrizes do Espaço de Estados

sys = ss(A, B, C, D);

% Parâmetros para Simulação

A = 0.1; % Amplitude do solavanco, valor arbitrário

t = 0:0.01:5; % Período/Tempo de 0 a 5 segundo com degrau (step) de 0.01 s

% Simulação

u = A \* sin(2 \* pi \* t); % Excitação da rua de 0.1 m (altura do solavanco)

[y, t, x] = lsim(sys, u, t); % lsim é uma função linear do MATLAB para simular sistemas no domínio tempo

% Resultados - basicamente aqui são funções e operações do MATLAB, deve ter semelhante no JAVA

figure;

subplot(2,1,1);

plot(t, y(:,1), 'LineWidth', 1.5);

title('Deslocamento da Massa Suspensa (x\_s)');

xlabel('Tempo (s)');

ylabel('Deslocamento (m)');

grid on;

subplot(2,1,2);

plot(t, y(:,2), 'LineWidth', 1.5);

title('Deslocamento da Massa Não Suspensa (x\_u)');

xlabel('Tempo (s)');

ylabel('Deslocamento (m)');

grid on;

Os resultados extraídos são:

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

**ANEXO B**

Abaixo está o código inicial, comentado, implementado em Java.

package application;

import org.apache.commons.math3.linear.\*;

import javafx.application.Application;

import javafx.stage.Stage;

public class SistemaDeSuspensao extends Application {

Amortecedor Amortecedor;

Mola MolaSuspensao;

Mola MolaPneu;

Massa MassaSuspensa;

Massa MassaNaoSuspensa;

Estrada Estrada;

double DeslocamentoMAX\_SUS;

double DeslocamentoMAX\_N\_SUS;

//Deixei as siglas para auxiliar nos cálculos

double m\_s = MassaSuspensa.getMassa(); // Massa suspensa (kg)

double m\_u = MassaNaoSuspensa.getMassa(); // Massa não suspensa (kg)

double k\_s = MolaSuspensao.getConstanteK(); // Rigidez da suspensão (N/m)

double k\_t = MolaPneu.getConstanteK(); // Rigidez do pneu (N/m)

double c\_s = Amortecedor.getConstanteC(); // Amortecimento da suspensão (Ns/m)

double A\_sin = Estrada.getAmplitude(); //Amplitude da estrada

double dt = 0.01; // Passo de tempo (s)

int steps = 500; // Número de passos (5 s com passo de 0,01 s)

public static void main(String[] args) {

// Lançando a aplicação JavaFX

launch(args);

}

@Override

public void start(Stage stage) {

// Matrizes do espaço de estados

double[][] A = {

{0, 1, 0, 0},

{-k\_s / m\_s,-c\_s / m\_s, k\_s / m\_s, c\_s / m\_s},

{0, 0, 0, 1},

{k\_s / m\_u, c\_s / m\_u,-(k\_s + k\_t) / m\_u,-c\_s / m\_u}

};

double[][] B = {

{0},

{0},

{0},

{k\_t / m\_u}

};

double[][] C = {

{1, 0, 0, 0},

{0, 0, 1, 0}

};

// Vetores de tempo e entrada

double[] Tempo = new double[steps];

double[] Oscilação = new double[steps];

for (int i = 0; i < steps; i++) {

Tempo[i] = i \* dt;

Oscilação[i] = Estrada.OscilacaoEstrada(Tempo[i]);

}

// Estado inicial

double[] x = {0, 0, 0, 0};

double[][] Deslocamento = new double[steps][2];

RealMatrix AMatrix = new Array2DRowRealMatrix(A);

RealMatrix BMatrix = new Array2DRowRealMatrix(B);

RealMatrix CMatrix = new Array2DRowRealMatrix(C);

// Função para calcular dx/dt

private static RealMatrix calculateDx(RealMatrix A, RealMatrix B, RealMatrix x,

RealMatrix u) {

return A.multiply(x).add(B.multiply(u));

}

**ANEXO C**

Código contemplando o cálculo do “*x*”.

// Método de Runge-Kutta de 4ª ordem

RealMatrix k1 = CalcularDerivada(AMatrix, BMatrix, xMatrix, OscilaçãoMatrix);

RealMatrix k2 =CalcularDerivada(AMatrix, BMatrix, xMatrix.add(k1.scalarMultiply(dt /

2)), OscilaçãoMatrix);

RealMatrix k3 = CalcularDerivada(AMatrix, BMatrix, xMatrix.add(k2.scalarMultiply(dt /

2)), OscilaçãoMatrix);

RealMatrix k4 = CalcularDerivada(AMatrix, BMatrix,

xMatrix.add(k3.scalarMultiply(dt)), OscilaçãoMatrix);

RealMatrix dx =

k1.add(k2.scalarMultiply(2)).add(k3.scalarMultiply(2)).add(k4).scalarMultiply(dt / 6);

// Atualiza o x

x = xMatrix.add(dx).getColumn(0);

**ANEXO D**

Código contemplando o cálculo do deslocamento.

// Saída

RealMatrix DeslocamentoMatrix = CMatrix.multiply(new Array2DRowRealMatrix(x));