1.Código do MATLAB

Seguindo o código que foi utilizado no MATLAB, tentou-se extrair as informações necessárias para poder fazer a mesma aplicação em java.
Código em 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 segundos 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
```

Inicialmente, como essa aplicação necessita de uma manipulação de matrizes, dessa forma, foi necessário adicionar uma biblioteca para suprir esta necessidade, então a biblioteca “org.apache.commons.math3.linear.” foi adicionada ao programa.

Aqui será mais aprofundado o cálculo do sistema de estado (sys = ss(A, B, C, D);), já que esta é uma função que não está presente em nenhuma das bibliotecas adicionadas, portanto seu cálculo receberá mais atenção.

2.Matrizes A, B, C e D

As matrizes A, B, C e D são definidas de uma forma que fazendo certa operação entre elas seja possível retornar as equações iniciais de movimento, que estão na imagem anexada a seguir.

$$m_s\ddot{x}_s = k_s(x_u - x_s) + c_s(\dot{x}_u - \dot{x}_s)$$

$$m_u\ddot{x}_u = -k_s(x_u - x_s) - c_s(\dot{x}_u - \dot{x}_s) + k_t(x_r - x_u)$$

3.1Apresentação do cálculo do Espaço de estados

Segundo a documentação oficial do MATLAB, a função para o cálculo do espaço de estados pode ser descrita como a imagem anexada a seguir:

`sys = ss(A,B,C,D)` creates a continuous-time state-space model object of the following form:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

Na qual A, B, C e D são as matrizes apresentadas anteriormente, “x” é um vetor de deslocamento , sendo que “x[0]” é o deslocamento da massa suspensa, “x[1]” é a velocidade da massa suspensa, “x[2]” é o deslocamento da massa não suspensa e “x[3]” é a velocidade da massa não suspensa e todas elas começaram zeradas já que o sistema parte do repouso, logo as velocidades são zero, e como a oscilação da estrada segue uma função senoidal, o seno quando tempo é igual a zero, será zero, logo o deslocamento também será zero. Além disso, “u” é um vetor relacionado à função senoidal, que dentro da função do java foi nomeado como deslocamento.

Partindo para o cálculo, a primeira função apresentada calcula a derivada de “x”, e não o” x” em si, portanto dentro do programa foi feito uma função que apenas calcula esta derivada, para não se ter que preocupar com seu cálculo várias vezes, ela foi nomeada de Calcularderivada(RealMatrix A, RealMatrix B, RealMatrix C, RealMatrix Oscilação).

Além disso foi adicionado uma função que lida com o tempo, já que precisamos lidar com o deslocamento conforme a passagem de tempo, logo o programa inicial ficou assim:

```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
```

```java
        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));
  }
```

3.2.Cálculo do x

Mas ainda é necessário fazer uma lógica para calcular o “x”, já que a função “CalcularDerivada” recebe como parâmetro o “x” e, por enquanto possui-se apenas o valor do x inicial, que é zero. Para isso será usado o método de Runge-Kutta de quarta ordem para lidar com matrizes de dimensão 4, basicamente ele segue a seguinte equação.

$$x = x + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \cdot dt$$

Onde o novo “x” é igual ao antigo “x” mais os parâmetros do método de Runge Kutta

```
        // 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);
```

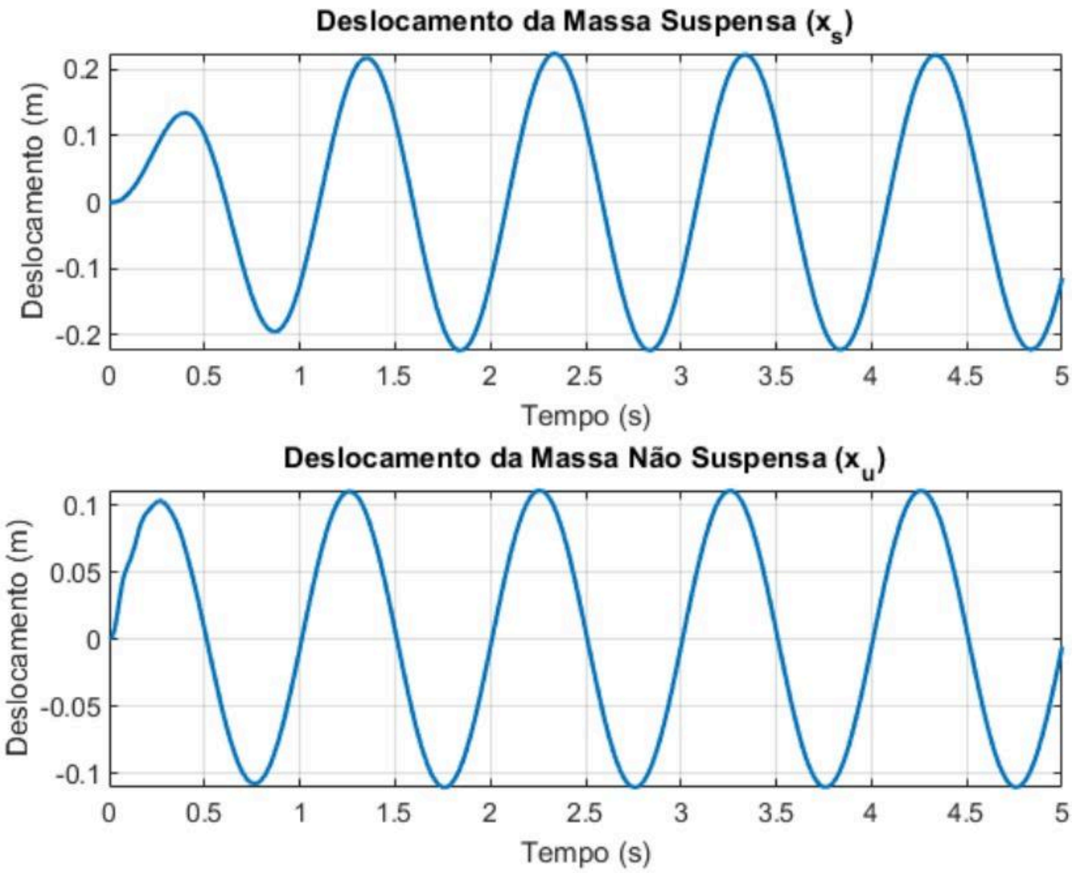
3.2.Cálculo do deslocamento

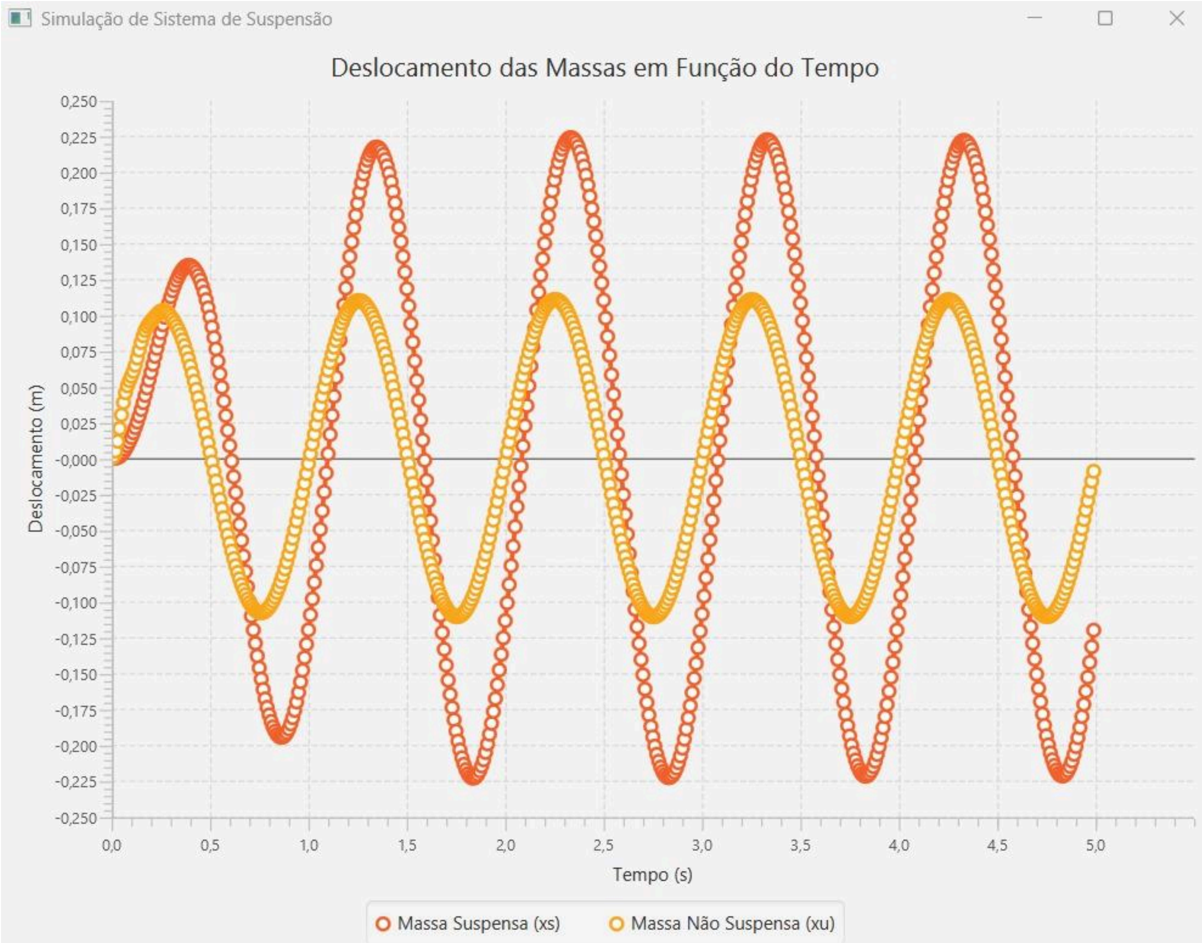
Por fim, pode-se calcular a segunda equação dos métodos de estados que gera como resultado o deslocamento, entretanto, como o matriz D possui apenas os valores de 0, ela não entrará na equação, pois quando ela for multiplicada com a matriz “u”, gerará uma matriz nula, dessa forma a matriz do deslocamento sera dada apenas por C multiplicada por “x”.

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

4.Resultados

Por meio dessa lógica, conseguiu-se simular o programa do MATLAB dentro de uma aplicação java de forma que comparando os dados obtidos dentro do programa no MATLAB e dentro do programa em java, os resultados são significativamente parecidos. Abaixo ficará anexado os gráficos das simulações feitas no MATLAB e abaixo ficará anexados os gráficos referentes às simulações feitas em java, comparando brevemente as duas, pode-se perceber que ambas possuem não só o formato da curva muito parecido, como também valores numéricos similares, o que corrobora com a ideia de que a simulação foi feita com sucesso.




Simulação de Sistema de Suspensão
Deslocamento das Massas em Função do Tempo
Deslocamento (m)
0,250
0,225
0,200
0,175
0,150
0,125
0,100
0,075
0,050
0,025
-0,000
-0,025
-0,050
-0,075
-0,100
-0,125
-0,150
-0,175
-0,200
-0,225
-0,250
0,0
0,5
1,0
1,5
2,0
2,5
3,0
3,5
4,0
4,5
5,0
Tempo (s)
Massa Suspensa (xs)
Massa Não Suspensa (xu)