



AERO - UFSM

APOSTILAS INTERATIVAS

MATLAB & Simulink

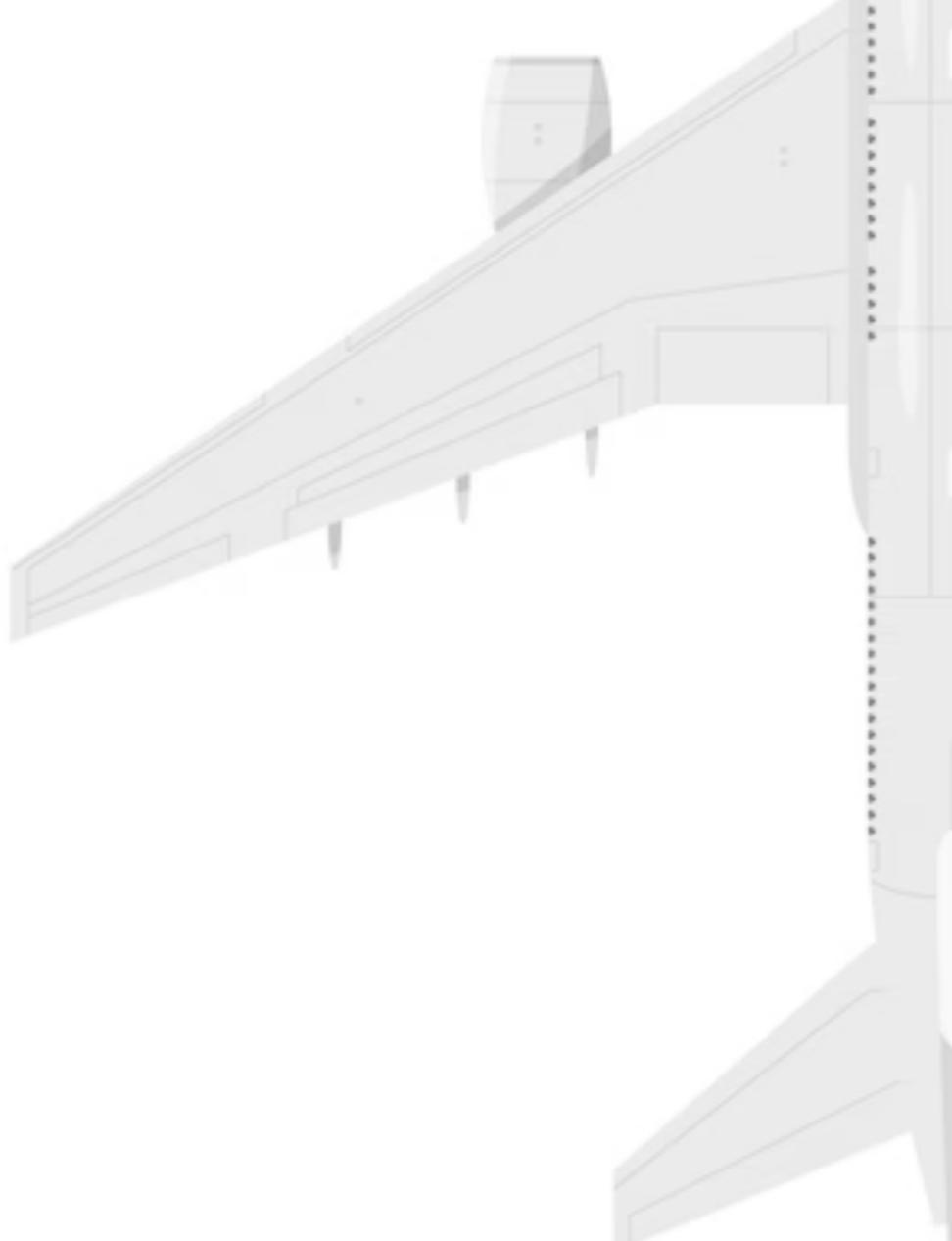
Edição: 2025

Desenvolvido por: Antônio Kipper e Guilherme
Jovino

Sumário

1	Introdução à apostila	1
2	MATLAB	2
2.1	Introdução ao MATLAB	2
2.2	Sintaxe básica	5
2.2.1	Tipos de dados	5
2.2.2	Operações matemáticas e matriciais	7
2.2.3	Indexação e manipulação de matrizes	10
2.2.4	Comandos básicos	14
2.3	Controle de fluxo	15
2.3.1	Estruturas condicionais: if, else, switch	15
2.3.2	Laços de repetição: for, while	16
2.3.3	Funções anônimas e personalizadas	17
2.4	Funções e ferramentas nativas	19
2.4.1	Uso de funções matemáticas nativas	19
2.4.2	Gráficos	20
2.4.3	Operações com arquivos	21
2.4.4	Debugging e boas práticas	22
3	Simulink	24
3.1	Introdução ao Simulink	24
3.2	Conceitos Fundamentais	25
3.2.1	Modelos Baseados em Blocos	25
3.2.2	O Conceito de Sinal	25
3.2.3	Sistemas Dinâmicos	26
3.3	Interface	27
3.3.1	Espaço de Modelagem	27
3.3.2	Simulation	28
3.3.3	Debug	30
3.3.4	Modeling	32
3.3.5	Apps	34
3.4	Principais Blocos	35
3.4.1	Fontes	35
3.4.2	Operações Matemáticas	38
3.4.3	Manipulação de Sinais Vetoriais	42
3.4.4	Blocos de Visualização e Exportação de Dados	45
3.4.5	Blocos Adicionais	47
3.5	Máscaras de Blocos e Criação de Submodelos	50
3.5.1	Máscaras de Blocos	50
3.5.2	Criação de Submodelos	51
3.6	Configurações da Simulação	53
3.6.1	Solver	54
3.6.2	Tempo de Simulação	55
3.6.3	Step Size	55

3.6.4	Parametrização por meio do MATLAB	56
3.7	Modelagem de Sistemas	57
3.7.1	Criação e Organização de Modelos	57
3.7.2	Testando e Validando o Modelo	59
4	Exemplo interativo: Simulação de lançamento de foguete pra inserção de carga útil em órbita baixa	61
4.1	Equações de movimento	61
4.2	Criação de máscaras e parametrização dos subsistemas	65
4.3	Demais módulos e integração	67
4.4	Configurações da simulação	69
4.5	Operação do modelo	71



1 Introdução à apostila

Esta apostila foi desenvolvida pela Escola Piloto de Engenharia Aeroespacial da Universidade Federal de Santa Maria e tem como objetivo apresentar os fundamentos teóricos e práticos das ferramentas MATLAB e Simulink, com foco em suas aplicações na simulação de sistemas dinâmicos e em diversas áreas da engenharia. Ao longo do material, serão abordadas as funcionalidades essenciais dessas ferramentas, com o intuito de capacitar os leitores a modelar, simular e analisar sistemas, além de explorar suas diversas aplicações no contexto acadêmico e industrial.

Destinada principalmente a estudantes e iniciantes nas áreas de engenharia, ciência da computação e áreas afins, esta apostila busca proporcionar uma introdução sólida às capacidades do MATLAB e Simulink. As ferramentas abordadas são amplamente utilizadas na academia e na indústria, sendo essenciais em atividades como análise e processamento de sinais, controle de sistemas, modelagem matemática, simulação de processos físicos, automação, robótica, sistemas embarcados e inteligência artificial. O conhecimento dessas ferramentas permite que os estudantes desenvolvam habilidades práticas para resolver problemas complexos de engenharia, além de facilitar a integração com projetos de pesquisa e desenvolvimento tecnológico.

A apostila está estruturada em duas grandes seções: a primeira dedicada ao MATLAB, abordando suas principais funcionalidades e comandos para manipulação de dados, visualização gráfica, desenvolvimento de algoritmos e automação de tarefas; a segunda, voltada ao Simulink, que explora a modelagem e simulação de sistemas dinâmicos, incluindo a criação de diagramas de blocos, integração com hardware e análise de desempenho de sistemas de controle. Ao final da apostila, será apresentado um projeto integrador que contemplará os conteúdos abordados nas seções anteriores. Esse projeto será disponibilizado em um repositório no Git, permitindo que os leitores acessem o material, testem suas implementações e compartilhem suas contribuições de forma colaborativa.

Com este material, espera-se que os leitores adquiram não apenas conhecimento teórico, mas também competências práticas que podem ser aplicadas tanto em contextos acadêmicos quanto industriais, reforçando a importância do MATLAB e Simulink como ferramentas de referência para a modelagem, simulação e análise de sistemas complexos.

2 MATLAB

2.1 Introdução ao MATLAB

MATLAB (Matrix Laboratory) é um ambiente de programação e linguagem de alto nível voltado principalmente para computação numérica, visualização de dados e desenvolvimento de algoritmos. Criado pela MathWorks, o MATLAB é amplamente utilizado em universidades, centros de pesquisa e na indústria, principalmente nas áreas de engenharia, física, matemática, economia e ciência de dados.

Entre suas principais aplicações, destacam-se:

- Análise e visualização de dados;
- Modelagem e simulação de sistemas dinâmicos;
- Processamento de sinais e imagens;
- Controle e automação;
- Algoritmos numéricos e computação científica;
- Desenvolvimento de protótipos e interfaces gráficas (GUI).

A interface gráfica do MATLAB é composta por diferentes janelas e painéis, cada um com funções específicas que auxiliam na interação com o ambiente de programação. A seguir, apresentam-se as principais áreas disponíveis:

- **Command Window** — Área destinada à execução direta de comandos. Nela, o usuário insere instruções, que são processadas imediatamente pelo MATLAB, retornando o resultado correspondente. Essa janela é adequada para testes rápidos, cálculos e execução de instruções pontuais.



Figura 1 – Janela *Command Window* no MATLAB

- **Workspace** — Apresenta as variáveis disponíveis na sessão atual do MATLAB, juntamente com informações como nome, tipo, dimensão e valor. Permite ao usuário monitorar e gerenciar os dados armazenados na memória durante a execução dos programas.

Workspace			
Name	Value	Size	Class
a	2	1x1	double
b	3	1x1	double

Figura 2 – Janela *Workspace* no MATLAB

- **Editor** — Ambiente utilizado para criação e modificação de arquivos de script e funções, geralmente com extensão `.m`. Diferentemente da *Command Window*, o Editor permite desenvolver códigos estruturados, documentados e reutilizáveis, sendo indicado para programas de maior complexidade.



```
% Primeiro programa em MATLAB
disp("Hello, World!")
```

Figura 3 – Editor de scripts no MATLAB

- **Current Folder** — Exibe o conteúdo do diretório de trabalho atual, incluindo arquivos e pastas. Facilita a organização e o acesso a dados, scripts e funções, permitindo também a navegação entre diferentes localizações no sistema de arquivos.



Figura 4 – Janela *Current Folder* no MATLAB

- **Figure** — Janela dedicada à apresentação de resultados gráficos. É utilizada para exibir visualizações geradas por funções como `plot` e `imshow`, sendo amplamente empregada na análise de dados, representação de sinais e criação de figuras para relatórios e apresentações.

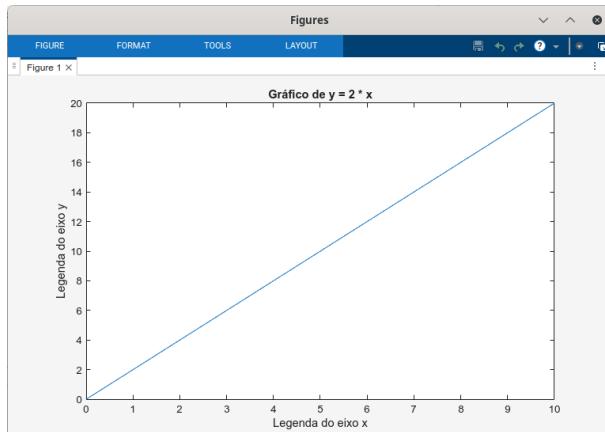


Figura 5 – Janela *Figure* no MATLAB

Essas áreas compõem o núcleo da interface do MATLAB, proporcionando um ambiente integrado que possibilita tanto a execução imediata de comandos quanto o desenvolvimento estruturado de códigos, além de oferecer recursos para gerenciamento de arquivos e visualização de resultados.

No MATLAB, é importante distinguir entre os três principais tipos de arquivos:

- **Scripts:** arquivos com extensão `.m` que contêm uma sequência de comandos executados em ordem. Eles compartilham o mesmo espaço de variáveis do `workspace`.
- **Funções:** também arquivos `.m`, mas com uma estrutura específica que permite encapsular código reutilizável. Funções têm suas próprias variáveis internas e não afetam diretamente o `workspace` principal.

- **Live Scripts:** arquivos com extensão `.mlx` que permitem combinar código, texto formatado, equações e gráficos em um mesmo documento interativo.

O exemplo a seguir mostra o tradicional primeiro programa em MATLAB:

```
1 % Primeiro programa em MATLAB
2 disp("Hello, World!")
```

Saída do programa:

```
>> script1
Hello, World!
```

2.2 Sintaxe básica

2.2.1 Tipos de dados

O MATLAB oferece diversos tipos de dados por padrão: escalares, vetores, matrizes, caracteres, datas, dicionários, etc.

Cada tipo de dado é melhor adaptado para uma aplicação específica.

- Escalares: por padrão, ao declarar um valor escalar no MATLAB, ele é definido como um número à vírgula flutuante, com precisão dupla (*double floating-point value*). Nesse caso, é possível guardar valores entre $-3.4 \cdot 10^{-38}$ e $+3.4 \cdot 10^{+38}$. Note como variáveis declaradas sem ponto e vírgula no final são exibidos na **Command Window** como saída, enquanto que aquelas que possuem o ponto e vírgula no final não são exibidos. No exemplo a seguir, a variável `a` não possui ponto e vírgula no final da linha, logo ela é exibida como saída. O oposto é verdadeiro para a variável `b`, que não é portanto exibida.

```
1 % Declaração de variáveis
2 a = 2
3 b = 3;
```

```
>> script2
a =
2
```

- Vetores: um vetor é uma estrutura de dados que guarda diversos valores escalares. É uma estrutura de tipo $1 \times N$ ou $N \times 1$, onde N representa o número de valores escalares armazenados dentro do vetor. Um vetor pode ser dito vetor-linha (caso seja da forma $1 \times N$) ou vetor-coluna (caso seja da forma $N \times 1$). Os vetores-linha usam opcionalmente vírgulas explícitas entre os elementos individuais. Nesse exemplo, a declaração das variáveis `vetor_linha` e `vetor_linha2` é equivalente.

```
1 % Declaração de vetores
2 vetor_linha = [1 2 3]
3 vetor_linha2 = [1, 2, 3]
4 vetor_coluna = [1; 2; 3]
```

```
>> script3

vetor_linha =
1      2      3

vetor_linha2 =
1      2      3

vetor_coluna =
1
2
3
```

- Matrizes: matrizes são estruturas de dados multi-dimensionais que armazenam valores escalares. Por exemplo, para matrizes bi-dimensionais (2D), seu tamanho é $M \times N$, onde M representa o número de linhas e N o número de colunas. O comando `size` pode ser usado para verificar o tamanho de uma matriz. O uso de vírgulas para separação de valores em uma mesma linha é opcional.

```
1 % Declaração de matrizes
2 matriz = [1, 2, 3;
3             4, 5, 6;
4             7, 8, 9;
5             10, 11, 12]
6 size(matriz)
```

```
>> script4

matriz =
1      2      3
4      5      6
7      8      9
10     11     12

ans =
4      3
```

- *String*: são textos, armazenados como vetores de caracteres. Podem ser declaradas usando aspas simples (vetor de caracteres) ou aspas duplas (*string array* - à partir de MATLAB R2017a).

```

1 % Declaração de strings
2 str1 = 'EP Aero'
3 str2 = "EP Aero"

```

```

>> script5

str1 =
'EP Aero'

str2 =
"EP Aero"

```

- Booleano: são valores lógicos que indicam verdadeiro (1) ou falso (0). Nesse caso, a variável condicao é falso (0) pois a é igual a 2, e não maior que 2.

```

1 % Declaração de booleanos
2 a = 2
3 condicao = (a > 2)

```

```

>> script6

a =
2

condicao =
logical

0

```

2.2.2 Operações matemáticas e matriciais

- Operações escalares: são operações básicas, como soma (+), subtração (-), multiplicação (*), divisão (/) e potência (^), realizadas entre dados do tipo escalar.

```

1 % Operações entre escalares
2 a = 2;
3 b = 3;
4 soma = a + b
5 subtracao = a - b
6 multiplicacao = a * b
7 divisao = a / b
8 potencia = a ^ b

```

```
>> script7

soma =
5

subtracao =
-1

multiplicacao =
6

divisao =
0.6667

potencia =
8
```

- Operações vetoriais e matriciais: são aquelas realizadas misturando escalares, vetores e matrizes. Nessas operações, podem ser utilizados operadores de adição (+), subtração (-), multiplicação matricial (*), multiplicação elemento a elemento (. *), divisão matricial (/ ou \) e divisão elemento a elemento (. / ou . \), entre outros.

```
1 % Operações vetoriais e matriciais
2 a = 2;
3 v1 = [1, 2];
4 v2 = [3; 4];
5 M1 = [1, 2;
       3, 4];
6 M2 = [5, 6;
       7, 8];
7
8 % Escalar x vetor
9 res1 = a * v
10
11 % Escalar x matriz
12 res2 = a * M1
13
14 % Vetor x vetor
15 res3 = v1 * v2 % v1 vetor-linha, v2 vetor-coluna
16
17
```

```

18 % Vetor x matriz
19 res4 = v * M1
20
21 % Matriz x matriz
22 res5 = M1 * M2
23
24 % Matriz x matriz (elemento à elemento)
25 res6 = M1 .* M2

```

```
>> script8
```

```
res1 =
```

```
2 4
```

```
res2 =
```

```
2 4
6 8
```

```
res3 =
```

```
11
```

```
res4 =
```

```
7 10
```

```
res5 =
```

```
19 22
43 50
```

```
res6 =
```

```
5 12
21 32
```

Os resultados obtidos são:

- **res1**: vetor resultante da multiplicação de cada elemento de **v1** pelo escalar **a**.
- **res2**: matriz resultante da multiplicação de cada elemento de **M1** pelo escalar **a**.

- **res3**: produto escalar entre $v1$ (vetor-linha) e $v2$ (vetor-coluna), resultando em um valor único.
- **res4**: resultado da multiplicação matricial entre $v1$ (vetor-linha) e $M1$, gerando um vetor-linha.
- **res5**: produto matricial padrão entre $M1$ e $M2$.
- **res6**: multiplicação elemento a elemento entre $M1$ e $M2$.

2.2.3 Indexação e manipulação de matrizes

Uma particularidade do MATLAB em relação à outras linguagens de programação, como C ou Java, é que a indexação dos elementos em vetores e matrizes é feita iniciando em 1.

```

1 % Indexação de vetores e matrizes
2 v = [1, 2, 3, 4];
3 M = [1, 2;
      3, 4];
5
6 elem1 = v(1) % Primeiro elemento (posição 1)
7 elem2 = M(2, 1) % Linha 2, coluna 1

```

```

>> script9

elem1 =
    1

elem2 =
    3

```

O MATLAB permite realizar indexação de forma avançada, permitindo selecionar subconjuntos de matrizes ou transformar matrizes em vetores. Um dos operadores mais úteis nesse contexto é o **operador “dois-pontos”** (`:`), que pode representar uma sequência de índices ou selecionar todos os elementos de uma dimensão.

Por exemplo:

- `1:4` representa a sequência de índices 1, 2, 3, 4.
- `:` sozinho indica “todos os elementos” daquela dimensão.

O exemplo a seguir mostra diferentes maneiras de extrair submatrizes ou vetores a partir de uma matriz original. Cada abordagem permite selecionar linhas e colunas específicas, extrair vetores-linha ou vetores-coluna, ou ainda linearizar toda a matriz em um vetor único.

```

1 % Indexação avançada
2 A = [ 1, 2, 3, 4;
3      5, 6, 7, 8;
4      9, 10, 11, 12;
5     13, 14, 15, 16];
6
7 % Extrair uma matriz de uma matriz (maneira 1)
8 B = A(1:2, 3:4) % Linhas 1 até 2, colunas 3 até 4 (matriz 2x2 no
9      canto superior direito de A)
10
11 % Extrair uma matriz de uma matriz (maneira 2)
12 C = A([1, 3], [2, 4]) % Linhas 1 e 3, colunas 2 e 4
13
14 % Extrair uma matriz de uma matriz (maneira 3)
15 D = A(:, 1:2) % Todas as linhas, colunas 1 até 2
16
17 % Extrair um vetor-linha de uma matriz
18 v_linha = A(1, :)
19
20 % Extrair um vetor-coluna de uma matriz
21 v_coluna = A(:, 1)
22
23 % Linearizar uma matriz
24 v = A(:)
```

```

>> script10

B =
3 4
7 8

C =
2 4
10 12

D =
1 2
5 6
9 10
13 14

v_linha =
```

```

1      2      3      4

v_coluna =
1
5
9
13

v =
1
5
9
13
2
6
10
14
3
7
11
15
4
8
12
16

```

Nesses exemplos:

- **B** contém a submatriz 2×2 do canto superior direito de A .
- **C** contém os elementos das linhas 1 e 3 e colunas 2 e 4.
- **D** contém todas as linhas e apenas as colunas 1 e 2.
- **v_linha** é o vetor-linha correspondente à primeira linha de A .
- **v_coluna** é o vetor-coluna correspondente à primeira coluna de A .
- **v** é o vetor-coluna formado por todos os elementos de A , dispostos na ordem de varredura por colunas (ordem *column-major*).

O MATLAB oferece a palavra-chave `end` para facilitar a manipulação de índices, especialmente quando se deseja referir ao último elemento de uma linha ou coluna. O uso de `end` permite criar expressões flexíveis que se adaptam automaticamente ao tamanho da matriz, evitando a necessidade de contar manualmente o número de linhas ou colunas.

```

1 % Indexação avançada (usando end)
2 A = [ 1, 2, 3, 4;
3     5, 6, 7, 8;
4     9, 10, 11, 12;
5     13, 14, 15, 16];
6
7 % Extrair uma matriz de uma matriz (exemplo 1)
8 B = A(2:end, 3:end)
9
10 % Extrair uma matriz de uma matriz (exemplo 2)
11 C = A(1:end-2, 2:end-1)
12
13 % Extrair um vetor de uma matriz
14 D = A(end, :)
15
16 % Extrair um elemento de uma matriz
17 a = A(end, end)

```

```

>> script11

B =
    7     8
   11    12
   15    16

C =
    2     3
    6     7

D =
    13    14    15    16

a =
    16

```

Nos exemplos, `end` é utilizado de diferentes maneiras:

- **B**: extraí uma submatriz formada pelas linhas 2 até a última linha e colunas 3 até a última coluna da matriz A.
- **C**: extraí uma submatriz formada pelas linhas 1 até a penúltima linha (`end-2`) e colunas 2 até a penúltima coluna (`end-1`) de A.

- **D**: extrai a última linha completa da matriz **A** como um vetor-linha.
- **a**: acessa o elemento localizado na última linha e última coluna da matriz **A**.

Dessa forma, o uso de `end` torna a indexação mais intuitiva e reduz a possibilidade de erros ao trabalhar com matrizes de tamanho variável.

2.2.4 Comandos básicos

O MATLAB possui diversos comandos úteis para controlar o ambiente de trabalho e explorar variáveis. Estes comandos ajudam a manter o `Workspace` organizado, facilitam a depuração e permitem acesso rápido à documentação das funções.

- `clc` – limpa a `Command Window`, removendo todas as mensagens e resultados anteriores exibidos no console.

```
1 clc
```

- `clear` – remove variáveis do `Workspace`. Pode-se especificar variáveis individuais ou usar `clear all` para apagar todas. Isso é útil para evitar conflitos de valores antigos com novos cálculos.

```
1 clear a b
2 clear all
```

- `who` – lista todas as variáveis atualmente armazenadas no `Workspace`. Serve para verificar rapidamente quais dados estão disponíveis.

```
1 who
```

```
>> who

Your variables are:

[...]
```

- `whos` – fornece informações detalhadas sobre as variáveis, incluindo tipo, tamanho, quantidade de memória utilizada e atributos. Pode ser usado para variáveis específicas ou todas, auxiliando no gerenciamento de memória e organização do `Workspace`.

```
1 whos a b
2 whos
```

```
>> whos a b
  Name      Size            Bytes  Class       Attributes
  a            1x1              8  double
  b            1x1              8  double
```

- **help** – exibe a documentação de funções e comandos diretamente no console, mostrando como usá-los, quais argumentos aceitam e exemplos de aplicação. É uma ferramenta essencial para aprendizado rápido e referência.

```
1 help disp
```

```
disp - Display value of variable
      This MATLAB function displays the value of variable X
      without printing
      the variable name.

[...]
```

Esses comandos básicos permitem manter o ambiente MATLAB limpo, inspecionar dados de forma organizada e acessar rapidamente informações sobre funções, melhorando a eficiência no desenvolvimento de scripts e funções.

2.3 Controle de fluxo

2.3.1 Estruturas condicionais: **if**, **else**, **switch**

As estruturas condicionais permitem executar diferentes blocos de código dependendo de uma condição lógica.

- **if**, **else**, **elseif**: permite verificar condições de forma sequencial.

```
1 % Estrutura condicional if-else
2 nota = 75;
3
4 if nota >= 90
5     resultado = "Excelente";
6 elseif nota >= 60
7     resultado = "Aprovado";
8 else
9     resultado = "Reprovado";
10 end
11
12 disp(resultado)
```

```
>> script12
Aprovado
```

- **switch**, **case**, **otherwise**: útil quando há múltiplas possibilidades discretas para uma variável.

```

1 % Estrutura switch-case
2 opcao = 2;
3
4 switch opcao
5     case 1
6         mensagem = "Opção 1 selecionada.";
7     case 2
8         mensagem = "Opção 2 selecionada.";
9     case 3
10        mensagem = "Opção 3 selecionada.";
11    otherwise
12        mensagem = "Opção inválida.";
13 end
14
15 disp(mensagem)

>> script13
Opção 2 selecionada.

```

2.3.2 Laços de repetição: for, while

Os laços de repetição são utilizados para executar um mesmo trecho de código múltiplas vezes.

- **for:** usado quando o número de repetições é conhecido de antemão.

```

1 % Loop for
2 for i = 1:5
3     fprintf("i = %d\n", i);
4 end

```

```

>> script14
i = 1
i = 2
i = 3
i = 4
i = 5

```

No exemplo apresentado, a variável **i** assume, sequencialmente, os valores de 1 a 5, e a função **fprintf** exibe cada valor na *Command Window*. Cada iteração do laço executa o comando contido dentro do **for**, resultando na impressão dos números de 1 a 5.

- **while:** usado quando o número de repetições depende de uma condição.

```

1 % Loop while
2 contador = 1;
3
4 while contador <= 3

```

```

5   disp("Contador = " + contador)
6   contador = contador + 1;
7 end

>> script15
Contador = 1
Contador = 2
Contador = 3

```

No exemplo, a variável `contador` inicia em 1 e o laço continua enquanto `contador <= 3`. A cada iteração, o valor atual de `contador` é exibido na tela e, em seguida, incrementado em 1. O laço termina automaticamente quando a condição deixa de ser verdadeira, neste caso após o contador atingir 4.

2.3.3 Funções anônimas e personalizadas

Funções permitem encapsular blocos de código que podem ser reutilizados.

- Funções anônimas: definidas em uma única linha, úteis para expressões simples.

```

1 % Função anônima
2 quadrado = @(x) x.^2;
3
4 resultado = quadrado(4)

```

```
>> script16
```

```
resultado =
```

16

No exemplo apresentado, a função anônima `quadrado` é definida utilizando a sintaxe `@(x) x.^2`, onde `x` é a entrada da função e `@(x) x.^2` é a operação realizada.

A sintaxe geral para a definição de uma função anônima no MATLAB é:

```
nome_funcao = @(parametros) expressao
```

- `nome_funcao`: identificador (nome) que você atribui à função anônima, usado para chamá-la posteriormente.
- `@(parametros)`: indica que está sendo criada uma função anônima; o símbolo `@` precede a lista de parâmetros de entrada entre parênteses. Se a função não tiver entradas, os parênteses ficam vazios: `@()`.
- `expressao`: operação ou cálculo que a função realizará e retornará. Deve ser uma única expressão, não é permitido conter múltiplas linhas de código.

A função é então chamada com o valor 4, retornando o resultado 16, que é exibido na *Command Window*.

- Funções personalizadas: definidas em arquivos separados com a palavra-chave `function`. O nome do arquivo deve ser o mesmo da função principal.

Arquivo: dobro.m

```
1 function y = dobro(x)
2     y = 2 * x;
3 end
```

Uso da função:

```
1 % Chamando a função dobro
2 resultado = dobro(5)
```

```
>> script17

resultado =

10
```

No exemplo apresentado, a função `dobro` recebe como entrada um valor `x` e retorna `y`, que é o dobro de `x`.

A sintaxe geral para a definição de uma função personalizada no MATLAB é:

```
function [valores_de_retorno] = nome_da_funcao(parametros)

– function: indica que se está definindo uma função.
– [valores_de_retorno]: especifica quais variáveis a função irá retornar. Se houver apenas um valor de retorno, os colchetes são opcionais; se houver múltiplos valores de retorno, todos devem ser listados entre colchetes, separados por vírgula.
– nome_da_funcao: identificador da função, que deve coincidir com o nome do arquivo .m.
– (parametros): define as entradas que a função receberá ao ser chamada, podendo haver múltiplos parâmetros separados por vírgula.
```

Essa estrutura permite criar funções flexíveis, que podem retornar um ou mais resultados a partir de uma ou mais entradas.

A função é armazenada em um arquivo chamado `dobro.m` e pode ser chamada de qualquer outro script ou da *Command Window* simplesmente utilizando seu nome, seguido dos argumentos necessários. No exemplo de uso, a função é chamada com o valor 5, e o resultado retornado é 10, que é exibido na *Command Window*.

Funções personalizadas permitem reutilizar código de forma organizada, facilitando a manutenção e a legibilidade do programa.

2.4 Funções e ferramentas nativas

O MATLAB oferece uma variedade de funções e ferramentas nativas para facilitar o desenvolvimento científico, técnico e de engenharia. Nesta seção, exploramos algumas das mais úteis.

2.4.1 Uso de funções matemáticas nativas

Funções matemáticas básicas, disponíveis por padrão no MATLAB, permitem realizar cálculos comuns sobre escalares, vetores e matrizes. Entre elas estão o seno (`sin`), o cosseno (`cos`), a exponencial (`exp`), o logaritmo natural (`log`), o logaritmo decimal (`log10`) e a raiz quadrada (`sqrt`). Essas funções podem ser aplicadas tanto a valores escalares quanto a vetores e matrizes, operando elemento a elemento quando necessário.

```

1 % Uso de funções nativas
2 x = pi / 4;
3 seno = sin(x)
4 cosseno = cos(x)
5 exponencial = exp(1)
6 logaritmo = log(exp(1))      % logaritmo natural
7 logaritmo10 = log10(1000)    % logaritmo base 10

```

```

>> script18

seno =
0.7071

cosseno =
0.7071

exponencial =
2.7183

logaritmo =
1

logaritmo10 =
3

```

Essas funções também são vetorizadas:

```

1 % Funções nativas vetorizadas
2 v = [0, pi/2, pi];
3 res = sin(v)

```

```
>> script19
res =
      0    1.0000      0
```

No exemplo apresentado, o vetor `v` contém três valores: 0, $\pi/2$ e π . Ao aplicar a função `sin` sobre `v`, o MATLAB calcula o seno de cada elemento do vetor, retornando outro vetor `res` com os resultados correspondentes, ou seja, [0, 1, 0]. Esse comportamento evita a necessidade de criar laços explícitos para aplicar a função a cada elemento.

2.4.2 Gráficos

MATLAB é amplamente utilizado para visualização de dados e criação de gráficos. Ele fornece diversas funções que permitem gerar representações visuais de vetores, matrizes e funções matemáticas de maneira rápida e personalizável.

```
1 % Gráficos em MATLAB
2 x = 0:0.1:2*pi;
3 y1 = sin(x);
4 y2 = cos(x);
5
6 figure
7 subplot(2,1,1)
8 plot(x, y1)
9 xlabel('x')
10 ylabel('sin(x)')
11 title('Gráfico do seno')
12
13 subplot(2,1,2)
14 plot(x, y2, 'r')
15 xlabel('x')
16 ylabel('cos(x)')
17 title('Gráfico do cosseno')
```

No exemplo apresentado, o vetor `x` é definido de 0 a 2π com passo 0.1, e os vetores `y1` e `y2` armazenam, respectivamente, os valores de $\sin(x)$ e $\cos(x)$ correspondentes. A função `figure` abre uma nova janela de figura, enquanto `subplot(2,1,1)` e `subplot(2,1,2)` permitem organizar múltiplos gráficos em uma mesma figura, dividindo-a em 2 linhas e 1 coluna, posicionando os gráficos no primeiro e segundo espaços.

A função `plot(x, y)` gera o gráfico de `y` em função de `x`. Os comandos `xlabel` e `ylabel` definem os rótulos dos eixos, enquanto `title` adiciona um título ao gráfico. No segundo gráfico, o argumento adicional '`r`' no `plot` indica que a curva será desenhada na cor vermelha.

Outras funções gráficas úteis incluem:

- `legend`: adiciona legenda para identificar curvas diferentes no mesmo gráfico.
- `grid on`: ativa a grade do gráfico, facilitando a leitura dos valores.

- **hold on/off**: permite sobrepor múltiplas curvas no mesmo gráfico ou resetar o gráfico atual.
- **axis**: define os limites e a escala dos eixos.
- **xlim** e **ylim**: definem separadamente os limites do eixo x e do eixo y.

O gráfico será gerado na janela *Figure*, como mostrado a seguir.

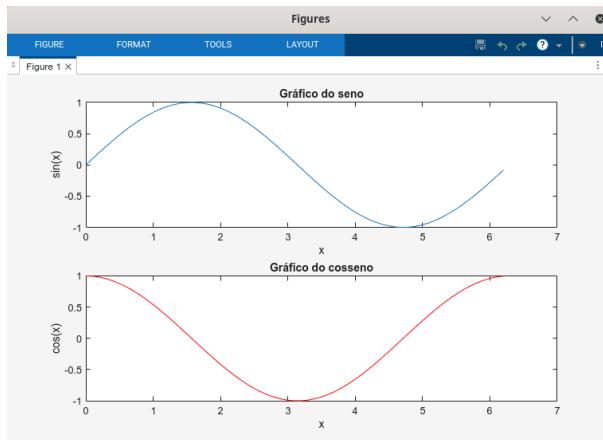


Figura 6 – Gráfico em MATLAB.

2.4.3 Operações com arquivos

MATLAB permite salvar e carregar variáveis em arquivos ‘.mat’, bem como ler e escrever arquivos de texto.

Leitura e escrita de arquivos .mat:

```

1 % Salvar e ler dados de arquivos .mat
2 a = 42;
3 b = [1, 2, 3];
4 save('dados.mat', 'a', 'b')
5 clear
6 load('dados.mat')
```

```

>> script21

>> who

Your variables are:

a b
```

Leitura e escrita de arquivos texto:

```

1 % Salvar e ler dados de arquivos texto
2 a = 42;
3 fid = fopen('saida.txt', 'w');
```

```

4 fprintf(fid, 'Valor de a: %d\n', a);
5 fclose(fid);
6
7 fid = fopen('saida.txt', 'r');
8 linha = fscanf(fid, '%s');
9 fclose(fid);
10 disp(linha)

```

>> script22

Valordea:42

2.4.4 Debugging e boas práticas

O MATLAB oferece diversas ferramentas para depuração (debugging) e boas práticas de programação que ajudam a identificar erros e melhorar a legibilidade do código.

Ferramentas de debugging:

- **Breakpoints:** pontos de interrupção que podem ser adicionados clicando na margem do editor do MATLAB. Quando o código atinge um breakpoint, a execução pausa, permitindo examinar o estado das variáveis e o fluxo do programa.
- **disp():** exibe o valor de uma variável ou mensagem no console, útil para acompanhar a execução do código sem interromvê-lo.
- **keyboard:** interrompe a execução do script ou função e entra em modo interativo, permitindo manipular variáveis e testar comandos diretamente.
- **dbstop:** permite definir breakpoints programaticamente, podendo parar a execução em determinadas linhas ou quando ocorrer um erro.
- **dbclear:** remove breakpoints definidos anteriormente.
- **dbstatus:** exibe todos os breakpoints ativos no código.

```

1 % script23.m
2 function resultado = soma_debug(a, b)
3     keyboard
4     resultado = a + b;
5 end

```

No exemplo acima, ao executar `soma_debug(3, 4)`, a execução será pausada no comando `keyboard`. Isso permite inspecionar os valores de `a` e `b`, testar operações interativamente e verificar se o resultado final será o esperado antes de continuar a execução da função.

Boas práticas de codificação:

- Nomear variáveis de forma descritiva, facilitando o entendimento do que cada variável representa.

- Comentar o código sempre que necessário, explicando trechos complexos ou decisões importantes.
- Evitar variáveis globais, pois podem gerar efeitos colaterais indesejados e dificultar o rastreamento de erros.
- Usar funções para modularizar o código, tornando-o mais organizado, reutilizável e fácil de testar.

Segundo essas ferramentas e práticas, o código MATLAB se torna mais confiável, fácil de manter e menos propenso a erros.

3 Simulink

3.1 Introdução ao Simulink

Simulink é uma plataforma gráfica de modelagem, simulação e análise de sistemas dinâmicos desenvolvida pela MathWorks. Ele permite que os engenheiros, cientistas e pesquisadores criem modelos de sistemas físicos e processos complexos por meio de um ambiente visual de arrastar e soltar blocos. Ao invés de escrever código manualmente, como no MATLAB, o Simulink usa uma interface gráfica onde você conecta blocos que representam componentes do sistema que está sendo modelado, como filtros, motores, circuitos elétricos, sistemas mecânicos, entre outros.

A principal função do Simulink é permitir a simulação de sistemas dinâmicos, ou seja, sistemas que evoluem no tempo, como:

- **Sistemas de controle:** Como um controlador PID para regular a velocidade de um motor.
- **Sistemas de comunicação:** Como modulação e demodulação de sinais em comunicação digital.
- **Sistemas mecatrônicos:** Como o comportamento de robôs ou veículos autônomos.
- **Circuitos elétricos:** Como a resposta de circuitos a diferentes sinais de entrada.

O Simulink é uma ferramenta robusta que pode ser utilizada para análise e verificação do desempenho de sistemas antes de serem implementados fisicamente, permitindo testar hipóteses e ajustes em projetos de forma rápida e econômica. Nesse contexto, algumas vantagens do Simulink incluem:

- **Modelagem visual:** Facilita a criação de modelos complexos sem a necessidade de programação intensa.
- **Simulação de sistemas dinâmicos:** Permite modelar e simular tanto sistemas contínuos (como um filtro analógico) quanto discretos (como sistemas digitais).
- **Validação e otimização de sistemas:** Permite testar diferentes parâmetros e cenários para validar o funcionamento de um sistema antes de sua implementação real.

O Simulink é projetado para ser usado em conjunto com o MATLAB, oferecendo uma combinação de ferramentas para análise e desenvolvimento de sistemas dinâmicos. O MATLAB, por si só, é uma plataforma de programação numérica e computação matemática que permite realizar cálculos, criar gráficos e processar dados. Por conta disso, A integração entre o MATLAB e o Simulink ocorre de maneira fluida e oferece uma série de vantagens:

- **Controle e script:** O MATLAB permite que você escreva scripts ou funções para controlar a execução da simulação no Simulink, parametrizar modelos e automatizar testes. Por exemplo, você pode definir variáveis ou parâmetros em um script MATLAB e usá-los diretamente em um modelo no Simulink.

- **Análise e visualização de dados:** Após a simulação de um modelo no Simulink, os dados podem ser facilmente exportados para o MATLAB, onde é possível realizar uma análise mais profunda, como cálculos avançados, otimização ou geração de gráficos. O MATLAB possui uma grande gama de funções para análise de dados que são complementadas pela simulação visual no Simulink.
- **Personalização e extensão:** Através do MATLAB, você pode criar funções personalizadas, usar algoritmos ou ferramentas de aprendizado de máquina, por exemplo, e incorporá-los dentro de um modelo Simulink. Isso é especialmente útil quando você precisa de uma funcionalidade mais complexa que não é fornecida pelos blocos padrão do Simulink.
- **Simulação de sistemas híbridos:** O Simulink pode simular sistemas contínuos e discretos, e o MATLAB pode ser usado para integrar simulações numéricas ou algoritmos em tempo real. Por exemplo, você pode utilizar o MATLAB para resolver equações diferenciais complexas e, em seguida, usar o Simulink para simular o sistema de maneira gráfica.

Em resumo, enquanto o Simulink oferece um ambiente visual e interativo para a modelagem e simulação de sistemas dinâmicos, o MATLAB serve como uma poderosa ferramenta de análise e programação que pode complementar e expandir as capacidades do Simulink. Juntas, essas duas ferramentas formam uma combinação robusta para quem deseja desenvolver, testar e otimizar sistemas de engenharia de maneira intuitiva.

3.2 Conceitos Fundamentais

3.2.1 Modelos Baseados em Blocos

No Simulink, **modelos baseados em blocos** são a forma principal de representar e simular sistemas. Cada bloco no Simulink representa uma operação, componente ou processo dentro do sistema maior que está sendo modelado. Esses blocos podem ser combinados para formar sistemas mais complexos, representando, por exemplo, circuitos elétricos, sistemas mecânicos ou processos de controle.

A ideia central por trás de **modelos baseados em blocos** é a representação de sistemas como um conjunto de funções matemáticas ou físicas, onde a entrada de cada bloco influencia a saída. O modelo é construído conectando blocos de forma a representar o fluxo de informações ou de sinais, o que permite analisar o comportamento do sistema como um todo.

Esses modelos podem ser de **tempo contínuo** ou **discreto**, dependendo da natureza do sistema que está sendo simulado.

3.2.2 O Conceito de Sinal

Um **sinal** pode ser entendido como uma função matemática que transmite informações através de uma variação de uma ou mais de suas propriedades, como amplitude, frequência ou fase, ao longo do tempo. No Simulink, os sinais são representados por **blocos fonte** que geram diferentes tipos de formas de onda. Exemplos de sinais comuns incluem:

- **Senoidal:** Um sinal com uma forma de onda suave e periódica.
- **Quadrado:** Um sinal que alterna entre dois valores com frequência constante.
- **Rampa:** Um sinal que aumenta ou diminui linearmente ao longo do tempo.
- **Impulso:** Um sinal com valor zero, exceto em um único instante de tempo, onde ele tem um pico.

Esses sinais podem ser utilizados para representar entradas em sistemas dinâmicos, como sinais de controle, dados de sensores, entre outros. Nesse contexto, os sinais em Simulink são utilizados para modelar e simular uma ampla gama de sistemas e fenômenos, como:

- **Sistemas de Controle:** No controle de sistemas dinâmicos, sinais de entrada, como sinais de referência e distúrbios, são processados para gerar um sinal de controle que regula a saída do sistema.
- **Circuitos Elétricos:** Em circuitos, sinais de tensão e corrente variam ao longo do tempo, e a simulação desses sinais é essencial para o design de circuitos elétricos.
- **Processamento de Sinais:** Em sistemas de processamento de sinais, sinais como áudio ou imagem são processados, filtrados ou analisados para extração de informações ou melhoria de qualidade.
- **Sistemas Mecânicos:** Em sistemas mecânicos, como motores ou veículos, sinais representam posições, velocidades e acelerações, permitindo simular o comportamento desses sistemas.

O Simulink oferece uma vasta gama de blocos e ferramentas para gerar, manipular e analisar esses sinais em tempo real, o que é crucial para o design e a otimização de sistemas de engenharia.

3.2.3 Sistemas Dinâmicos

Sistemas dinâmicos são sistemas cujas variáveis de estado mudam ao longo do tempo. Esses sistemas podem ser representados através de equações diferenciais ou de diferença, dependendo de sua natureza (contínuo ou discreto). No Simulink, você pode modelar esses sistemas utilizando blocos que representam operações matemáticas que, quando combinadas, formam um modelo completo de um sistema dinâmico.

Existem dois tipos principais de sistemas dinâmicos que podem ser modelados no Simulink:

- **Sistemas contínuos:** O comportamento do sistema é descrito por equações diferenciais, onde as variáveis de estado mudam continuamente ao longo do tempo. Exemplos típicos de sistemas contínuos incluem circuitos elétricos analógicos, sistemas mecânicos e processos industriais. No Simulink, blocos como "Integração" e "Derivada" são usados para representar operações em tempo contínuo. A simulação de um sistema contínuo envolve a solução de equações diferenciais, o que é feito automaticamente pelo Simulink durante a execução da simulação.

- **Sistemas discretos:** O comportamento do sistema é descrito por equações de diferença, onde o sistema é analisado em intervalos de tempo discretos. Isso é comum em sistemas digitais, onde os sinais são amostrados em momentos específicos. Exemplos de sistemas discretos incluem algoritmos de controle digital, sistemas de comunicação digital, e processamento de sinais discretos. No Simulink, blocos como "Delay" e "Discrete Transfer Function" são usados para modelar sistemas discretos. Esses sistemas são descritos por equações de diferença, e a simulação ocorre em intervalos de tempo específicos, com os sinais sendo atualizados a cada passo de tempo.

Em resumo, os **sistemas dinâmicos contínuos** são mais comuns em modelos de circuitos analógicos, sistemas mecânicos e controle de processos, enquanto os **sistemas discretos** são mais utilizados em sistemas digitais, como computadores e controladores digitais. Sendo assim, a principal diferença entre sistemas de tempo contínuo e discreto é a maneira como o tempo é tratado durante a simulação. Em um sistema contínuo, a evolução do sistema é descrita como uma função suave, enquanto, no tempo discreto, o sistema é descrito apenas em pontos específicos no tempo.

3.3 Interface

3.3.1 Espaço de Modelagem

O **espaço de modelagem** é a área onde você constrói e visualiza o seu modelo. Essa área oferece uma interface gráfica que permite arranjar, conectar e modificar os blocos de acordo com a necessidade do sistema que está sendo modelado. Aqui, você pode desenhar e configurar todo o seu modelo com base nos blocos disponíveis na biblioteca.

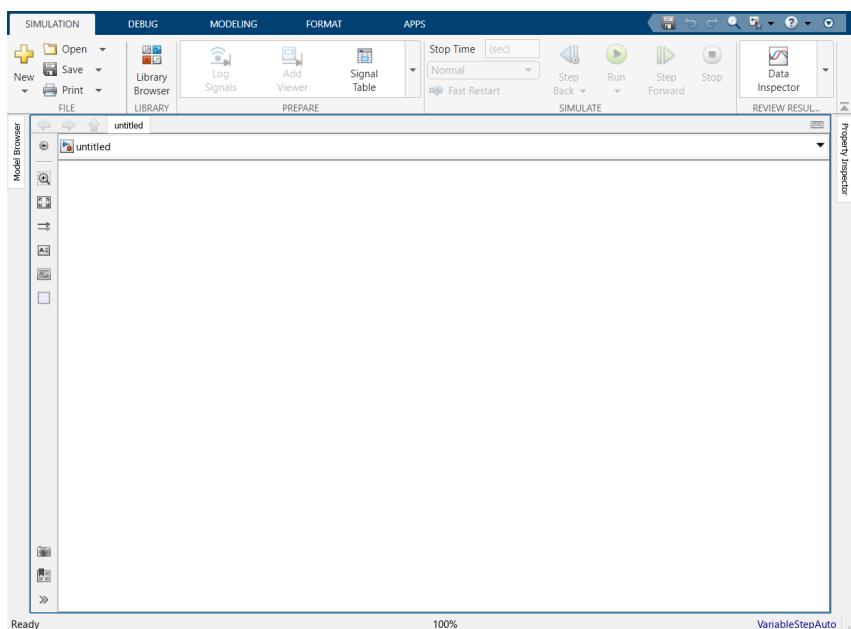


Figura 7 – Espaço de modelagem do Simulink

No espaço de modelagem, você pode:

- **Arranjar e conectar blocos:** Arraste blocos da biblioteca para a área de trabalho e conecte-os arrastando linhas entre as entradas e saídas de cada bloco.
- **Ajustar parâmetros:** Cada bloco tem uma série de parâmetros ajustáveis. Por exemplo, em um bloco de "Soma", você pode definir os sinais de entrada e o número de entradas.
- **Simular e visualizar:** Após configurar o modelo, você pode realizar simulações e visualizar os resultados diretamente na área de trabalho.

O espaço de modelagem é uma representação gráfica e intuitiva do sistema que você está projetando. Ele permite que você tenha uma visão sistêmica de como os sinais estão fluindo entre os blocos, facilitando o entendimento do comportamento do sistema.

Dentro do espaço de modelagem, é possível acessar a barra de menus, a qual fornece acesso a diversas funcionalidades e ferramentas para a simulação e análise dos sistemas modelados. Cada menu tem um conjunto de opções relacionadas a tarefas específicas, como simulação, configuração do modelo, análise de dados e outros. Vamos explorar os menus mais comuns e suas principais funcionalidades.

3.3.2 Simulation

O menu **Simulation** é uma das partes mais importantes da interface, pois ele contém várias funcionalidades relacionadas à execução e controle da simulação. As opções no menu Simulation permitem controlar como o modelo será simulado, configurar parâmetros da simulação e analisar os resultados.

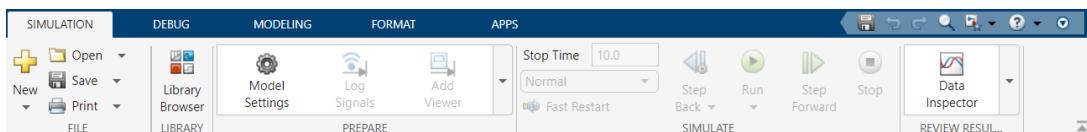


Figura 8 – Menu Simulation do Simulink

- **Run:** Inicia a simulação do modelo. Você pode clicar neste botão para executar o modelo com os parâmetros definidos.
- **Stop:** Interrompe a simulação atual. Isso é útil quando você deseja interromper o processo de simulação a qualquer momento, por exemplo, se os resultados se tornarem irrelevantes ou você precisar modificar o modelo.
- **Pause:** Pausa a simulação em andamento. A pausa é útil se você deseja interromper temporariamente a simulação, por exemplo, para inspecionar os resultados em determinado ponto do tempo, sem perder o estado atual.
- **Model Settings:** Esta opção abre uma janela de configuração onde você pode ajustar os parâmetros da simulação, como o tempo de simulação, os solvers utilizados, as opções de dados e muito mais.

- **Library Browser:** O Library Browser oferece uma visão completa de todos os blocos disponíveis no Simulink. Ele permite que você organize, acesse e busque blocos para arrastar e usar na construção de seu modelo. Você pode navegar por categorias como "Fontes", "Controle", "Matemática", e muito mais. O Library Browser também permite a criação de bibliotecas personalizadas para armazenar e reutilizar submodelos.

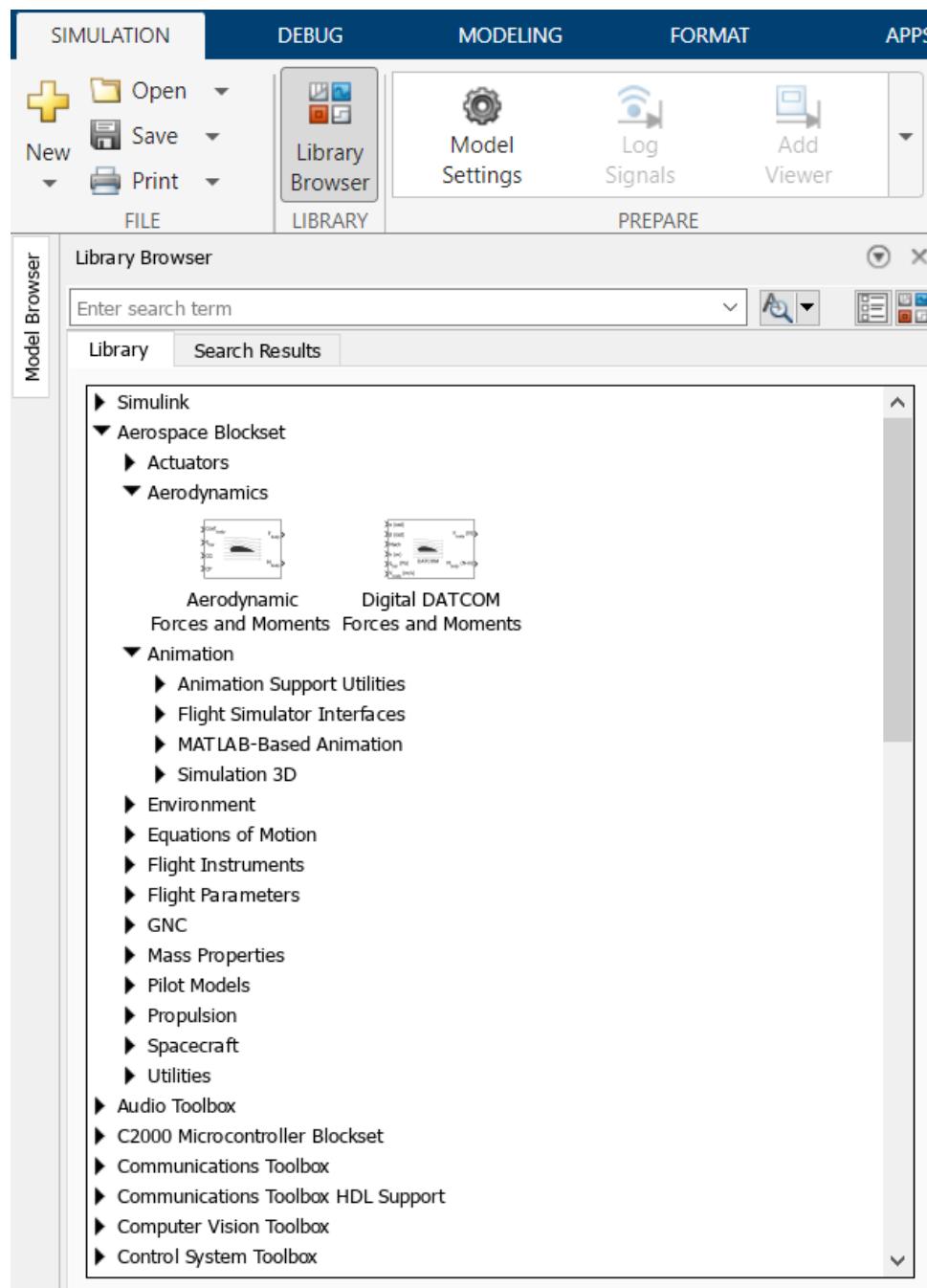


Figura 9 – Library Browser

- **Data Inspector:** O Data Inspector é uma ferramenta utilizada para inspecionar e analisar os dados gerados durante a simulação. Ele permite que você visualize sinais e variáveis ao longo do tempo, compare diferentes resultados e exporte dados para o MATLAB para análises adicionais. O Data Inspector é uma ferramenta poderosa para validar o comportamento do modelo e garantir que ele funcione conforme o esperado.

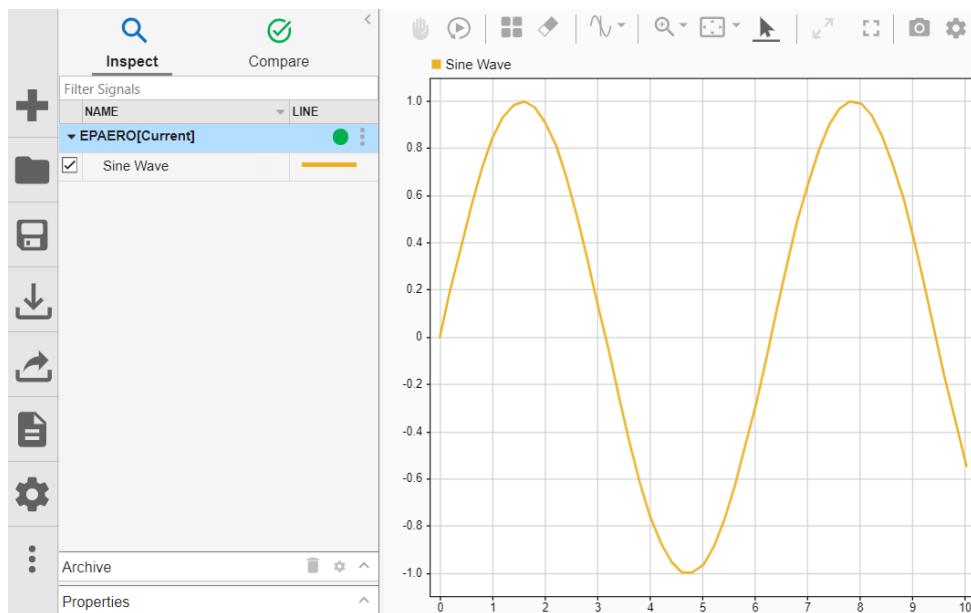


Figura 10 – Data Inspector

Essas são as principais funcionalidades do menu **Simulation**, que controlam o início, a pausa, a configuração e a análise de simulações. Utilizando essas opções, você pode testar, ajustar e otimizar os modelos que está desenvolvendo.

3.3.3 Debug

O menu **Debug** é uma parte essencial da interface do Simulink, pois fornece ferramentas que ajudam a depurar e inspecionar o modelo. Essas ferramentas são especialmente úteis para encontrar e corrigir erros durante a simulação, permitindo que você verifique a execução do modelo e o comportamento dos sinais em tempo real. As opções do menu Debug permitem realizar pausas na simulação, monitorar variáveis, e até mesmo adicionar pontos de interrupção para interromper a execução e verificar o estado do modelo.

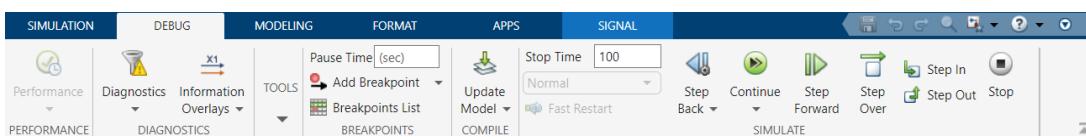


Figura 11 – Menu Debug no Simulink

- **Start Debugging:** Inicia a sessão de depuração. Essa opção permite que você inicie o processo de depuração para examinar o comportamento do seu modelo enquanto ele está sendo executado.
- **Pause:** Pausa a execução da simulação em qualquer ponto. Isso é útil para analisar o modelo em um determinado momento, inspecionar variáveis ou parâmetros de sistemas, e garantir que o comportamento do modelo esteja correto.
- **Step:** Avança a simulação passo a passo. Isso permite que você execute o modelo de forma controlada, um passo de cada vez, para observar o efeito de cada operação ou de cada bloco.
- **Step Over:** Similar à opção "Step", mas permite que você avance sem entrar nos blocos de função, facilitando o acompanhamento de blocos de alto nível sem precisar examinar o funcionamento interno de cada um.
- **Step Into:** Permite entrar nos blocos de função ou de submodelo, útil quando você deseja investigar mais a fundo o funcionamento de partes específicas do seu modelo.
- **Add Breakpoint:** Permite definir pontos de interrupção dentro do seu modelo. Esses pontos fazem com que a execução da simulação seja interrompida automaticamente no local onde o ponto de interrupção foi definido, permitindo uma análise detalhada do estado do modelo naquele momento.

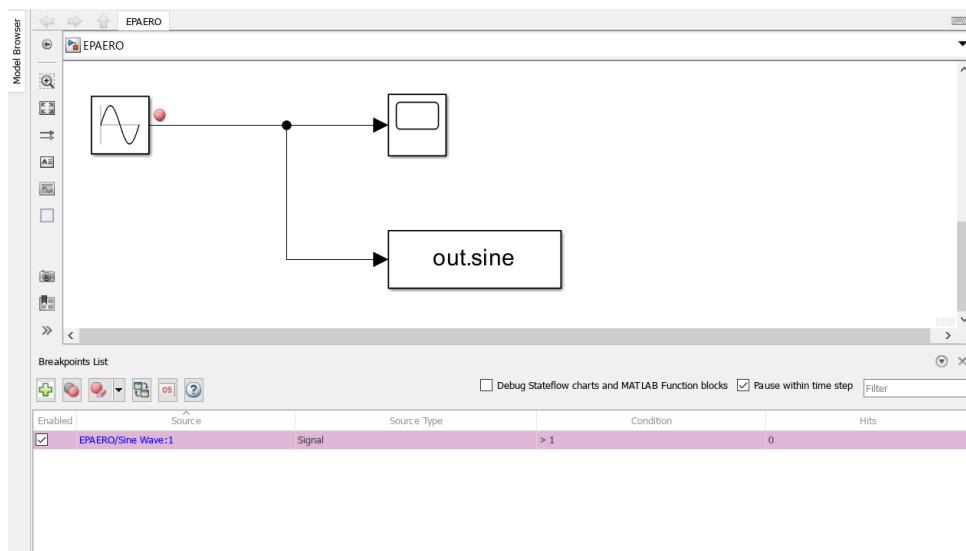


Figura 12 – Definição de um Ponto de Interrupção

- **Diagnostics:** Utilizado para monitorar e identificar erros e alertas durante a simulação. Essa opção permite configurar e visualizar os diagnósticos de erros e avisos do Simulink, oferecendo feedback sobre qualquer inconsistência ou problema no modelo.

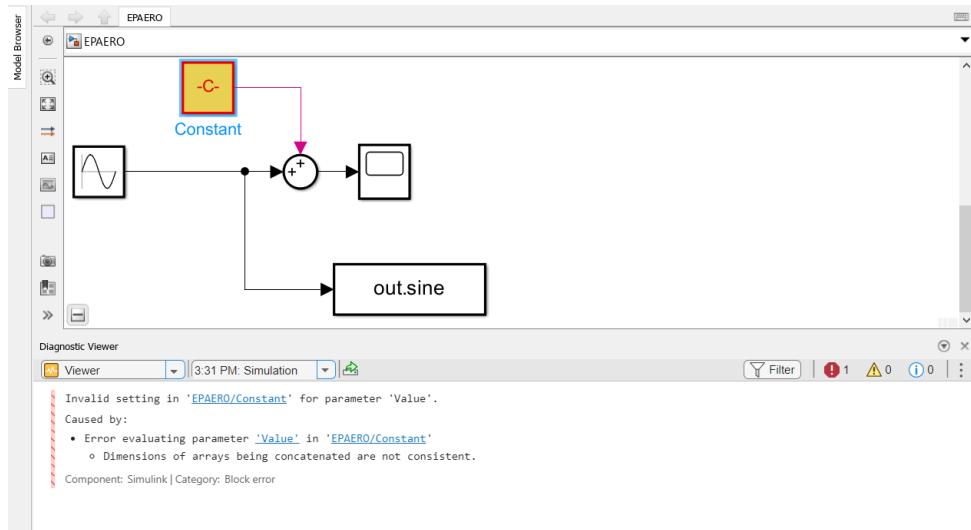


Figura 13 – Exemplo de erro exibido no **Diagnostics Viewer**

- **Information Overlays:** permite que você visualize informações adicionais sobre os blocos durante a simulação, exibindo sobreposições com detalhes relevantes, como o valor atual de uma variável ou o estado de um bloco.

Essas funcionalidades do menu **Debug** são essenciais para depurar e auxiliar no desenvolvimento do seu modelo no Simulink. A capacidade de pausar a simulação, avançar passo a passo e inspecionar sinais e variáveis torna a depuração muito mais eficaz, permitindo identificar e corrigir problemas durante o desenvolvimento do modelo.

3.3.4 Modeling

O menu **Modeling** reúne diversas ferramentas fundamentais para configurar, organizar e executar simulações dentro do ambiente do Simulink. Ele está dividido em grupos funcionais que abrangem desde a edição e gerenciamento de dados do modelo até a execução da simulação. Este menu é especialmente útil para preparar o ambiente antes de iniciar a simulação, configurar parâmetros importantes e estruturar o modelo de forma modular e eficiente.

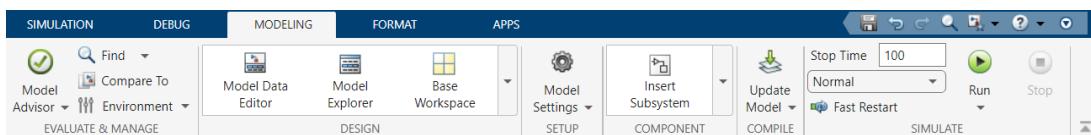


Figura 14 – Menu Modeling no Simulink

- **Model Advisor:** Ferramenta que verifica a conformidade do modelo com boas práticas, normas de projeto e diretrizes de modelagem. Ele gera relatórios com sugestões de melhorias ou ajustes a serem feitos no modelo.
- **Find / Compare To / Environment:**

- **Find:** Permite localizar blocos, parâmetros ou elementos específicos dentro do modelo.
- **Compare To:** Compara o modelo atual com uma versão anterior ou outro modelo, destacando as diferenças entre eles.
- **Environment:** Gerencia variáveis, caminhos e dependências do ambiente MATLAB associado ao modelo.
- **Model Data Editor:** Abre o editor de dados do modelo, uma interface que permite visualizar e editar sinais, parâmetros e variáveis do modelo de maneira organizada.
- **Model Explorer:** Ferramenta avançada que oferece uma visão estruturada de todos os elementos do modelo, incluindo subsistemas, blocos, variáveis e configurações. Facilita a navegação e o gerenciamento de componentes complexos.

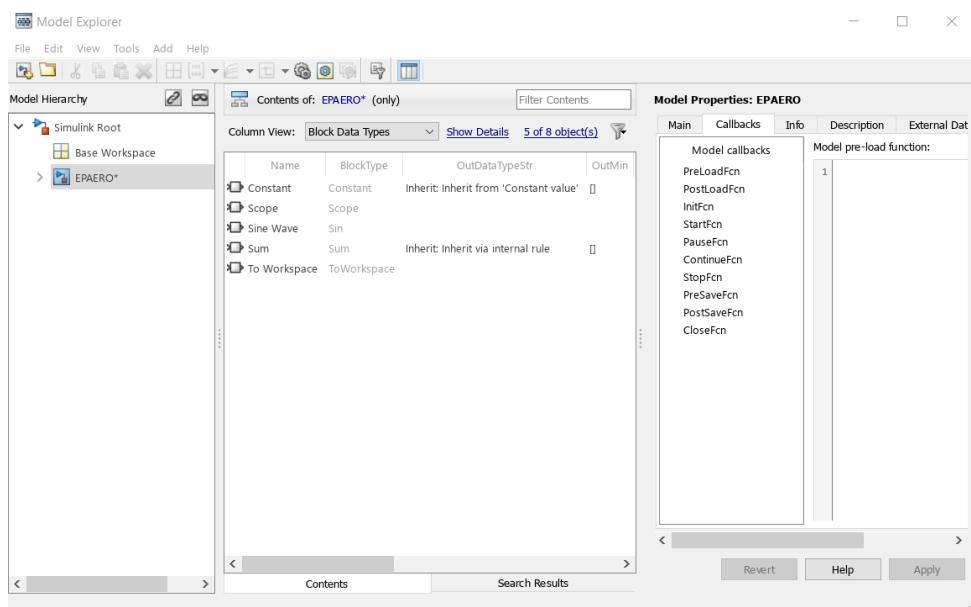


Figura 15 – Model Explorer

- **Base Workspace:** Acesso direto ao espaço de trabalho base do MATLAB, onde as variáveis utilizadas no modelo são armazenadas. Isso permite visualizar ou modificar os valores usados na simulação diretamente no ambiente MATLAB.
- **Insert Subsystem:** Permite criar um novo subsistema diretamente a partir do menu. Subsystems são usados para modularizar o modelo, organizando blocos relacionados dentro de um único bloco encapsulado.
- **Update Model:** Compila o modelo, atualizando suas dependências e verificando a consistência entre os blocos. Isso é necessário antes de executar simulações para garantir que o modelo esteja pronto para rodar.

Essas ferramentas disponíveis no menu **Modeling** são essenciais tanto para o desenvolvimento quanto para a execução de simulações eficientes e organizadas. Elas permitem desde o gerenciamento e análise de dados até a configuração detalhada do ambiente de simulação, tornando o processo de modelagem mais ágil, modular e controlado.

3.3.5 Apps

O menu **Apps** oferece acesso a uma série de aplicativos adicionais que estendem as funcionalidades do Simulink. Esses aplicativos são ferramentas especializadas que ajudam a modelar e analisar sistemas complexos de maneiras mais eficientes e detalhadas, oferecendo uma gama de recursos para diversas áreas de engenharia, como controle, otimização, simulação física, e análise avançada.

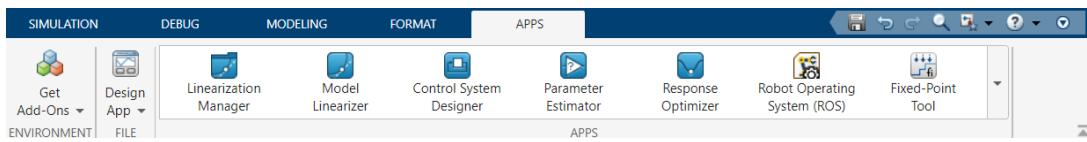


Figura 16 – Menu Apps no Simulink

Ao acessar o menu **Apps**, você pode explorar e utilizar uma variedade de ferramentas que podem ser adicionadas ao seu modelo para realizar tarefas como:

- **Design e Análise de Controle:** Ferramentas para projetar controladores e realizar análises de sistemas de controle, como resposta ao degrau, estabilidade e otimização de parâmetros.
- **Modelagem Física:** Aplicativos que permitem modelar sistemas físicos, como mecânicos, elétricos, hidráulicos e térmicos, com simulações detalhadas das interações entre componentes.
- **Animações e Visualização:** Ferramentas para criar animações 3D de modelos Simulink, úteis para a visualização de sistemas mecânicos e robóticos.
- **Otimização e Ajuste de Parâmetros:** Aplicativos que permitem otimizar os parâmetros de seu modelo, ajustando-os para atender a critérios de desempenho específicos.
- **Geração de Código:** Ferramentas para gerar código de controle para implementação em hardware, incluindo opções para sistemas embarcados e de tempo real.

Esses aplicativos são especialmente valiosos quando você precisa de uma solução pronta para problemas específicos de engenharia e não deseja desenvolver esses recursos do zero. Eles ajudam a acelerar o processo de desenvolvimento, permitindo a utilização de ferramentas especializadas. Vale ressaltar que a disponibilidade dos aplicativos está atrelada às **toolboxes** que estão instaladas.

3.4 Principais Blocos

No Simulink, os blocos são elementos essenciais para a construção de modelos dinâmicos e sistemas de controle. Cada bloco tem uma função específica e pode ser configurado de maneira a atender aos requisitos do seu modelo. Nesta seção, vamos explorar alguns dos principais blocos que você utilizará ao criar modelos no Simulink.

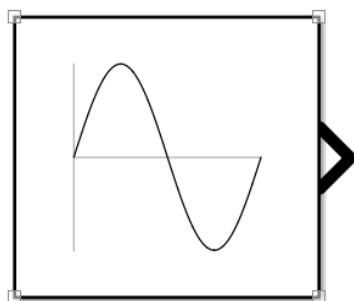
3.4.1 Fontes

Os blocos de **Fonte** são fundamentais para a construção de modelos no Simulink, pois eles fornecem os sinais de entrada que os sistemas simulados processarão. Cada tipo de fonte tem uma aplicação específica, e entender como configurá-las corretamente pode melhorar significativamente a eficácia das suas simulações. A flexibilidade desses blocos permite simular uma grande variedade de cenários, desde sinais periódicos e constantes até sinais aleatórios e pulsantes.

Além dos blocos mencionados a seguir, o Simulink oferece outros tipos de fontes que podem ser explorados dependendo das necessidades do seu modelo, como fontes baseadas em funções matemáticas ou sinais personalizados gerados a partir de dados externos. É importante se familiarizar com essas fontes para poder criar modelos realistas e precisos de sistemas dinâmicos.

O bloco **Sine Wave** (Onda Senoidal) gera um sinal senoidal contínuo, que pode ser usado para simular sinais alternados, como os encontrados em sistemas elétricos, de controle e comunicação.

- **Como funciona:** O bloco gera uma onda senoidal com uma frequência e amplitude ajustáveis. Ele também permite definir uma fase inicial, que determina o ponto de partida da onda.
- **Aplicações:** O bloco **Sine Wave** é comumente usado para modelar sinais de entrada que variam sinusoidalmente ao longo do tempo, como tensões alternadas em circuitos elétricos, vibrações mecânicas ou qualquer sistema que envolva oscilações periódicas.

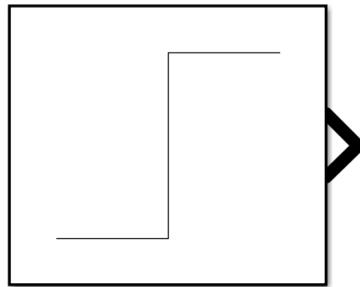


Sine Wave

Figura 17 – Bloco Sine Wave no Simulink

O bloco **Step** (Degrau) gera um sinal que muda de valor instantaneamente a partir de um certo ponto no tempo, simulando uma mudança abrupta. Esse tipo de sinal é útil quando você precisa modelar mudanças em um sistema que ocorrem de forma repentina, como uma chave sendo ligada ou um sistema de controle que sofre uma mudança de *setpoint*.

- **Como funciona:** O bloco **Step** tem um valor inicial (normalmente zero) e, após um determinado tempo (tempo de passo), o sinal pula para um valor final, que pode ser configurado.
- **Aplicações:** O bloco **Step** é utilizado para testar a resposta de um sistema a uma entrada súbita, como a resposta de um sistema de controle a um novo valor de referência ou a ativação de um dispositivo eletrônico.

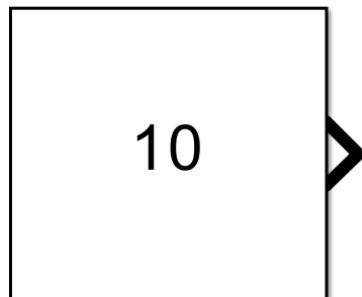


Step

Figura 18 – Bloco Step no Simulink

O bloco **Constant** (Constante) gera um sinal constante ao longo do tempo, ou seja, o valor do sinal não muda enquanto o modelo estiver em simulação. Esse tipo de bloco é útil para simular condições de entrada que não variam, como uma tensão ou fluxo constante.

- **Como funciona:** O bloco **Constant** permite que você defina um valor fixo para o sinal gerado. Esse valor será mantido durante toda a simulação.
- **Aplicações:** O bloco **Constant** é amplamente utilizado quando se precisa de um sinal estático em um modelo, como para representar uma fonte de alimentação fixa, um parâmetro de configuração ou um sinal de entrada que não varia ao longo do tempo.

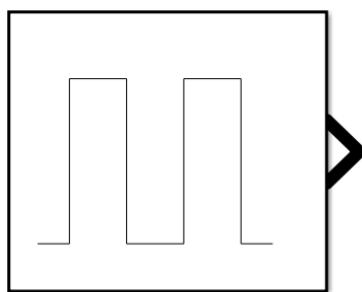


Constant

Figura 19 – Bloco Constant no Simulink

O bloco **Pulse Generator** (Gerador de Pulso) gera uma onda quadrada com uma sequência de pulsos de altura e largura configuráveis. Ele é muito útil quando você precisa gerar sinais com comportamentos de chaveamento ou pulsos repetitivos, como em sistemas digitais ou eletrônicos.

- **Como funciona:** O bloco **Pulse Generator** permite ajustar a largura do pulso, o período (tempo entre dois pulsos consecutivos) e a fase de início. A altura do pulso pode ser configurada para representar a magnitude do sinal.
- **Aplicações:** Utilizado em sistemas de controle digital, processamento de sinais, ou em circuitos eletrônicos que utilizam sinais pulsados.

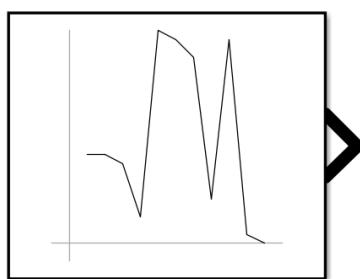


Pulse Generator

Figura 20 – Bloco Pulse Generator no Simulink

O bloco **Random Number** (Número Aleatório) gera sinais com base em uma distribuição aleatória de valores, o que é útil quando se deseja modelar ruídos ou incertezas em sistemas. Ele pode ser configurado para gerar números aleatórios com distribuições específicas, como distribuição uniforme ou normal.

- **Como funciona:** O bloco **Random Number** gera números aleatórios dentro de um intervalo específico, com uma média e desvio padrão ajustáveis. Isso permite simular a aleatoriedade de variáveis em sistemas, como o ruído em sistemas de comunicação ou em medições de sensores.
- **Aplicações:** É frequentemente utilizado em simulações de sistemas estocásticos, onde o comportamento de variáveis aleatórias precisa ser modelado, como em processos de ruído em sinais ou simulações de Monte Carlo.



Random Number

Figura 21 – Bloco Random Number no Simulink

3.4.2 Operações Matemáticas

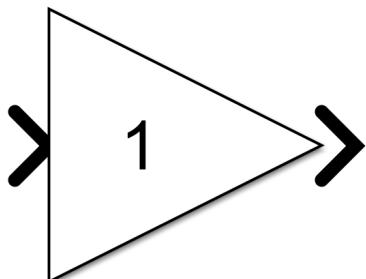
Os blocos de **Operações Matemáticas** são utilizados para realizar cálculos entre sinais e variáveis. Esses blocos são amplamente usados para ajustar, combinar ou manipular dados dentro de um modelo. O Simulink oferece diversos blocos para realizar operações matemáticas, como multiplicação, soma, ganho e divisão. A seguir, veremos alguns dos blocos de operação matemática mais comuns e suas aplicações.

Esses blocos são frequentemente utilizados em sistemas de controle, circuitos elétricos, processamento de sinais, e uma grande variedade de aplicações de engenharia. Além dos blocos que serão mencionados, o Simulink oferece outras operações matemáticas avançadas, que podem ser exploradas dependendo das necessidades do seu modelo. Familiarizar-se com esses blocos é essencial para poder construir modelos mais completos e funcionais.

O bloco **Gain** é utilizado para multiplicar um sinal de entrada por um valor constante, conhecido como "ganho". Esse bloco é útil quando você precisa ajustar a amplitude de um sinal ou realizar um escalonamento.

- **Como funciona:** O bloco **Gain** recebe um sinal de entrada e o multiplica por um valor constante que pode ser configurado no parâmetro do bloco. Esse valor de ganho pode ser positivo, negativo ou até mesmo variável, dependendo da necessidade do seu modelo.
- **Aplicações:** O bloco **Gain** é utilizado em sistemas de controle, amplificadores, filtros e onde quer que seja necessário ajustar a intensidade de um sinal. Ele também

pode ser usado para modelar amplificações ou atenuações de sinais em sistemas de controle e eletrônicos.



Gain

Figura 22 – Bloco Gain no Simulink

O bloco **Sum** realiza a soma ou subtração de dois ou mais sinais de entrada. Esse bloco é utilizado para combinar vários sinais, seja para somá-los ou subtrair um de outro.

- **Como funciona:** O bloco **Sum** pode ser configurado para realizar operações de soma ou subtração, dependendo da sua configuração. Você pode adicionar ou subtrair múltiplos sinais de entrada e também definir a ordem e o número de entradas.
- **Aplicações:** O bloco **Sum** é usado para modelar sistemas com múltiplas entradas, como sistemas de controle com feedback, onde várias variáveis precisam ser somadas ou subtraídas. É comum também em circuitos elétricos, sistemas de navegação e análise de sinais.

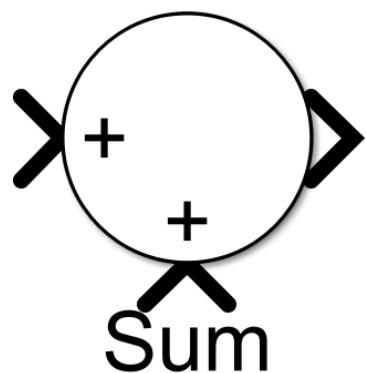
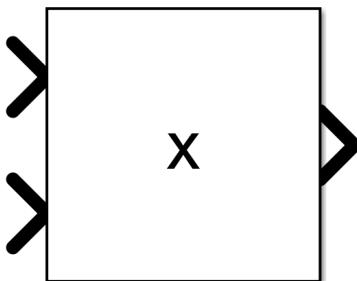


Figura 23 – Bloco Sum no Simulink

O bloco **Product** realiza a multiplicação de dois ou mais sinais de entrada. Ele é útil para representar processos que envolvem multiplicação ou interação entre variáveis.

- **Como funciona:** O bloco **Product** multiplica os sinais de entrada e retorna o produto dos sinais. Ele pode ser configurado para multiplicar dois ou mais sinais de entrada, com a possibilidade de ajustar parâmetros como o sinal de ganho.

- **Aplicações:** O bloco **Product** é usado em sistemas de controle, circuitos elétricos (como amplificadores), sistemas mecânicos (para calcular forças ou velocidades resultantes de interações entre componentes) e em sistemas de processamento de sinais, onde a multiplicação entre sinais é necessária.

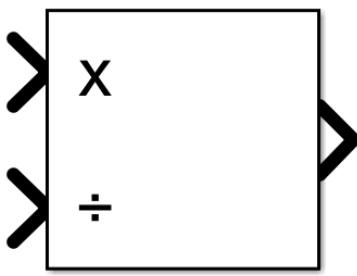


Product

Figura 24 – Bloco Product no Simulink

O bloco **Divide** realiza a divisão entre dois sinais de entrada. Este bloco é útil quando você precisa calcular uma razão ou fração entre variáveis.

- **Como funciona:** O bloco **Divide** recebe dois sinais de entrada e realiza a divisão entre eles, retornando o quociente. A operação de divisão é definida como o sinal de entrada de cima dividido pelo sinal de entrada de baixo.
- **Aplicações:** O bloco **Divide** é utilizado em sistemas que requerem o cálculo de razões, como sistemas de controle de velocidade, cálculos de eficiência de sistemas e circuitos elétricos, onde você precisa calcular a relação entre duas variáveis.

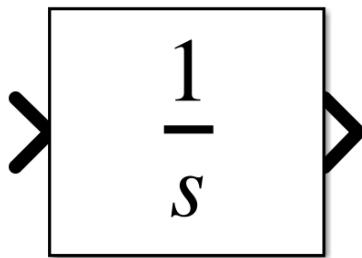


Divide

Figura 25 – Bloco Divide no Simulink

O bloco **Integrator** é um dos blocos mais fundamentais no Simulink, utilizado para representar a integração de um sinal ao longo do tempo. Esse bloco é essencial em modelos de sistemas dinâmicos, onde a acumulação ou a mudança gradual de uma variável ao longo do tempo precisa ser modelada. A integração é a operação inversa da diferenciação e é frequentemente utilizada em muitos tipos de sistemas, como os mecânicos e elétricos.

- **Como funciona:** O bloco **Integrator** recebe um sinal de entrada e calcula o valor acumulado ao longo do tempo. Em termos simples, ele ”acumula” o sinal de entrada, ou seja, ele soma o valor do sinal ao longo do tempo, considerando o valor inicial do sinal e a taxa de variação ao longo da simulação. A variável de saída do integrador, portanto, será o valor integrado desse sinal.
- **Aplicações:** O bloco **Integrator** é amplamente utilizado para modelar fenômenos que dependem de acumulação ou mudanças contínuas ao longo do tempo. Exemplos comuns incluem:
 - Em sistemas mecânicos, para calcular a posição a partir da velocidade, uma vez que a posição é a integral da velocidade em relação ao tempo.
 - Em circuitos elétricos, para calcular a carga acumulada em um capacitor a partir da corrente elétrica, já que a carga é a integral da corrente.
 - Em sistemas de controle, para modelar o comportamento de variáveis que mudam de forma contínua ao longo do tempo, como o controle de temperatura ou pressão, onde as variáveis de estado dependem da soma gradual de entradas.

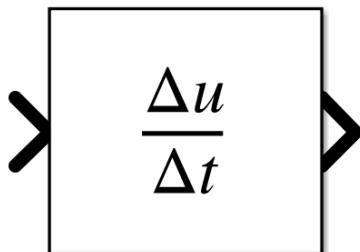


Integrator

Figura 26 – Bloco Integrator no Simulink

O bloco **Derivative** é utilizado para calcular a derivada de um sinal de entrada em relação ao tempo. Ele é essencial para modelar sistemas onde a taxa de variação de uma variável é importante, como sistemas de controle de velocidade e aceleração.

- **Como funciona:** O bloco **Derivative** calcula a taxa de variação de um sinal de entrada. Ele retorna a derivada do sinal, ou seja, a mudança instantânea do valor do sinal em relação ao tempo.
- **Aplicações:** O bloco **Derivative** é comumente usado em sistemas de controle (como no controle de posição e velocidade), modelagem de sistemas mecânicos (onde a aceleração é a derivada da velocidade), e análise de sinais.



Derivative

Figura 27 – Bloco Derivative no Simulink

3.4.3 Manipulação de Sinais Vetoriais

Os blocos de **Manipulação de Sinais Vetoriais** são utilizados para organizar e manipular sinais compostos por múltiplos componentes. Em muitos modelos, é necessário trabalhar com sinais vetoriais ou matriciais, e esses blocos facilitam o agrupamento, o acesso e a reorganização de dados. A seguir, veremos alguns dos blocos de manipulação de sinais vetoriais mais comuns e suas aplicações.

Os blocos **Mux** e **Demux** são operações inversas. O **Mux** é usado para combinar múltiplos sinais em um único vetor, enquanto o **Demux** realiza a operação inversa, separando um vetor em seus componentes individuais.

- **Como funciona:**

- O bloco **Mux** recebe múltiplos sinais de entrada e os combina em um único vetor de saída.
- O bloco **Demux** recebe um vetor ou sinal composto como entrada e o separa em múltiplos sinais escalares de saída.

- **Aplicações:**

- O bloco **Mux** é utilizado quando é necessário combinar sinais, como em sistemas de controle com múltiplas variáveis de entrada, ou quando se trabalha com vetores ou matrizes em sistemas que requerem agrupamento de sinais para simplificar o modelo.
- O bloco **Demux** é utilizado em sistemas onde é necessário separar um sinal composto em suas partes individuais para processamento posterior. É comumente usado em sistemas de controle e simulações que envolvem vetores de estados ou múltiplos sinais de entrada em um único ponto de processamento.

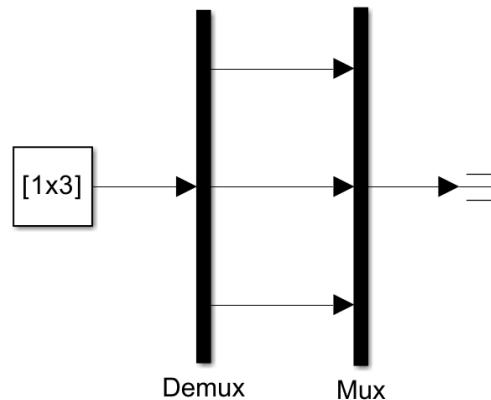
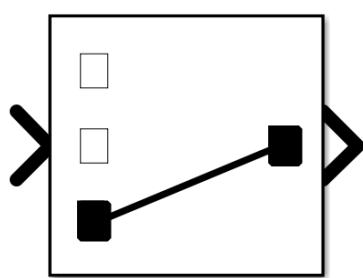


Figura 28 – Blocos Mux e Demux no Simulink

O bloco **Selector** permite acessar elementos específicos dentro de um vetor ou matriz. Ele é usado quando você precisa extraír um ou mais componentes de um sinal vetorial ou matricial.

- **Como funciona:** O bloco **Selector** seleciona e extraí componentes de um vetor ou matriz com base nos índices configurados. É possível selecionar um único valor ou um subconjunto de valores do vetor de entrada.
- **Aplicações:** O bloco **Selector** é utilizado em sistemas de controle e modelagem de sistemas onde é necessário acessar variáveis específicas dentro de um vetor de estados ou em matrizes de parâmetros. Ele é útil quando você só precisa de uma parte de um vetor ou de uma matriz para processamento posterior, como em análise de sinais ou sistemas dinâmicos.



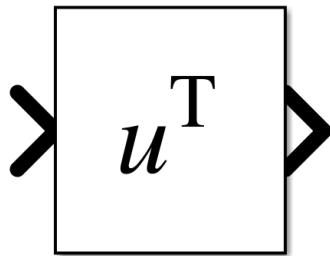
Selector

Figura 29 – Bloco Selector no Simulink

O bloco **Transpose** realiza a transposição de um vetor ou matriz, ou seja, ele troca suas linhas por colunas. Esse bloco é utilizado em sistemas que exigem manipulação de dimensões de matrizes ou vetores.

- **Como funciona:** O bloco **Transpose** recebe uma matriz ou vetor como entrada e retorna a transposta desse vetor ou matriz, ou seja, ele inverte as dimensões da entrada. Para um vetor $1 \times N$, ele se torna um vetor $N \times 1$.

- **Aplicações:** O bloco **Transpose** é útil quando se trabalha com transformações matriciais, como em sistemas de controle multivariáveis, cálculos de vetores em sistemas mecânicos ou elétricos, ou quando se precisa ajustar a orientação de dados em uma simulação.

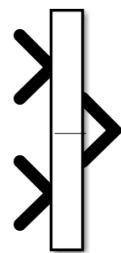


Transpose

Figura 30 – Bloco Transpose no Simulink

O bloco **Concatenate** é utilizado para combinar dois ou mais vetores ou matrizes em uma única variável. Ele pode ser usado para juntar sinais de diferentes origens, formando um vetor ou matriz maior.

- **Como funciona:** O bloco **Concatenate** junta múltiplos sinais de entrada, seja na direção das linhas ou das colunas, criando um vetor ou matriz maior. O número de entradas e a orientação da concatenação podem ser configurados.
- **Aplicações:** O bloco **Concatenate** é usado em situações onde você precisa agrupar sinais de diferentes fontes ou estados, como em sistemas de controle com múltiplas variáveis ou em sistemas de comunicação que exigem a combinação de sinais para processamento.



Vector Concatenate

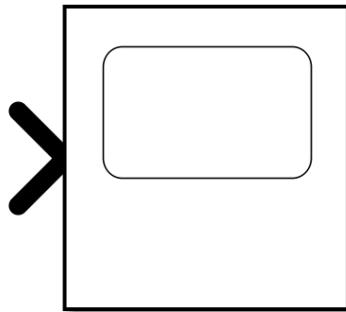
Figura 31 – Bloco Vector Concatenate no Simulink

3.4.4 Blocos de Visualização e Exportação de Dados

Os blocos de **Visualização e Exportação de Dados** são utilizados para observar e registrar os resultados das simulações. Esses blocos são fundamentais para o monitoramento de variáveis durante a simulação, bem como para a exportação de dados para análise posterior, seja para gráficos, relatórios ou para trabalhar com os dados no MATLAB. A seguir, veremos alguns dos blocos mais comuns usados para visualização e exportação de dados.

O bloco **Scope** é uma das ferramentas mais utilizadas no Simulink para visualizar sinais em tempo real. Ele exibe gráficos dinâmicos das variáveis simuladas ao longo do tempo, permitindo que você veja como os sinais evoluem durante a simulação.

- **Como funciona:** O bloco **Scope** recebe sinais de entrada e os exibe em um gráfico de linha, mostrando a evolução dos valores ao longo do tempo. O número de entradas e o tipo de visualização podem ser ajustados conforme as necessidades da simulação.
- **Aplicações:** O **Scope** é amplamente utilizado para monitorar o comportamento de variáveis durante a simulação, como a saída de sistemas de controle, variações de sinais de entrada ou a evolução de sistemas dinâmicos em geral. É particularmente útil em projetos de controle, eletrônicos e mecatrônica.



Scope

Figura 32 – Bloco Scope no Simulink

O bloco **Display** é utilizado para exibir valores de sinais em tempo real diretamente na interface do Simulink, sem a necessidade de abrir gráficos. Ele é útil quando você precisa monitorar rapidamente o valor de uma variável em particular durante a simulação.

- **Como funciona:** O bloco **Display** recebe um sinal de entrada e exibe seu valor atual durante a execução da simulação. O valor pode ser visualizado na interface gráfica do Simulink, em tempo real.
- **Aplicações:** O **Display** é útil quando você precisa monitorar variáveis específicas em tempo real sem a necessidade de gráficos complexos. Ele é frequentemente usado em sistemas de controle para monitoramento de variáveis chave ou em simulações simples onde o foco está em observar valores instantâneos.

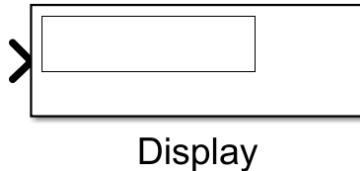


Figura 33 – Bloco Display no Simulink

O bloco **To Workspace** permite exportar dados da simulação para o MATLAB, facilitando a análise posterior dos resultados. Ele é essencial para quem deseja trabalhar com os dados em scripts ou funções no MATLAB para análise adicional, como plotagem, análise estatística ou manipulação de dados.

- **Como funciona:** O bloco **To Workspace** coleta os dados de um sinal durante a simulação e os envia para o espaço de trabalho do MATLAB. Você pode especificar o nome da variável e o formato de armazenamento, como "array", "timeseries", ou "struct".
- **Aplicações:** O **To Workspace** é usado para exportar resultados da simulação para análise posterior no MATLAB, permitindo que você gere gráficos, realize cálculos adicionais ou salve os dados para relatórios ou testes. Ele é essencial em simulações complexas onde os dados precisam ser manipulados fora do ambiente Simulink.

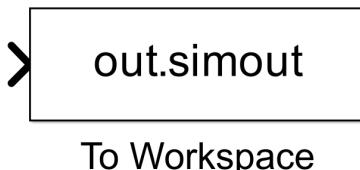


Figura 34 – Bloco To Workspace no Simulink

O bloco **To File** é utilizado para salvar os resultados de simulação em um arquivo de dados, como um arquivo .mat, que pode ser carregado e analisado posteriormente no MATLAB. Esse bloco permite gravar dados em arquivos, facilitando o armazenamento e compartilhamento de grandes volumes de dados de simulação.

- **Como funciona:** O bloco **To File** grava os dados de sinais ou variáveis durante a simulação em um arquivo no formato especificado (geralmente .mat). Ele também permite configurar o nome do arquivo e o formato dos dados.
- **Aplicações:** O **To File** é utilizado quando você deseja salvar os dados de simulação para análise posterior, arquivamento ou compartilhamento com outros colegas. Ele é útil em simulações longas ou em experimentos que geram grandes volumes de dados que precisam ser armazenados para análises futuras.

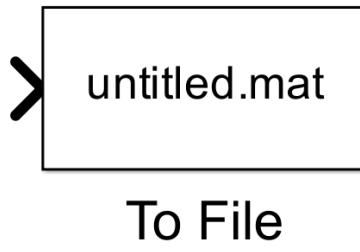


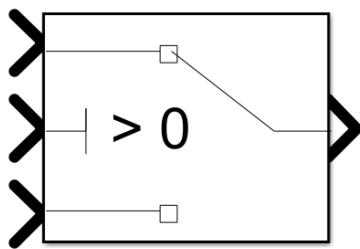
Figura 35 – Bloco To File no Simulink

3.4.5 Blocos Adicionais

Nesta seção, serão apresentados blocos que podem ser utilizados para introduzir funcionalidades mais avançadas no seu modelo, como controle lógico, implementação de funções personalizadas, e manipulação de dados temporais. A seguir, veremos alguns blocos adicionais mais comuns e suas aplicações.

O bloco **Switch** é utilizado para realizar uma operação condicional entre dois sinais, semelhante a um "if-else" em programação. Ele permite selecionar um dos dois sinais de entrada com base em uma condição de controle.

- **Como funciona:** O bloco **Switch** recebe três sinais: uma condição de controle e dois sinais de entrada. Se a condição de controle for verdadeira, o bloco passa o primeiro sinal de entrada; caso contrário, ele passa o segundo sinal de entrada.
- **Aplicações:** O bloco **Switch** é amplamente utilizado em sistemas de controle, onde é necessário alterar o comportamento do sistema com base em uma condição, como em sistemas de controle com limites ou em sistemas que alteram seu funcionamento com base em uma variável de controle.



Switch

Figura 36 – Bloco Switch no Simulink

O bloco **MATLAB Function** permite escrever código MATLAB diretamente dentro do seu modelo Simulink. Isso é útil quando você deseja implementar funções personalizadas ou algoritmos que não estão disponíveis diretamente nos blocos do Simulink.

- **Como funciona:** O bloco **MATLAB Function** permite que você insira código MATLAB para definir o comportamento do bloco. Esse código pode incluir funções matemáticas, lógicas, ou até mesmo manipulação de dados complexos.

- **Aplicações:** O **MATLAB Function** é usado em modelos que requerem cálculos ou lógica que não pode ser realizada facilmente com os blocos padrão do Simulink. Ele é útil quando você precisa de flexibilidade para criar funções específicas ou integrar algoritmos desenvolvidos em MATLAB.

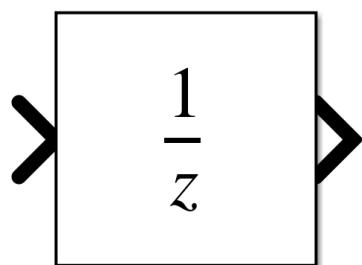


MATLAB Function

Figura 37 – Bloco MATLAB Function no Simulink

O bloco **Unit Delay** armazena o valor de uma variável e a mantém durante um ciclo de simulação. Ele é útil para implementar sistemas que dependem de variáveis de estado, como sistemas dinâmicos e sistemas de controle discretos.

- **Como funciona:** O bloco **Unit Delay** recebe um sinal de entrada e o armazena durante um ciclo de simulação. O valor armazenado é então passado para a próxima etapa do modelo na simulação.
- **Aplicações:** O **Unit Delay** é usado para implementar sistemas de controle discretos ou sistemas dinâmicos, onde o estado do sistema depende dos valores anteriores. Ele é frequentemente utilizado em sistemas de controle com feedback ou em sistemas que envolvem integração discreta.

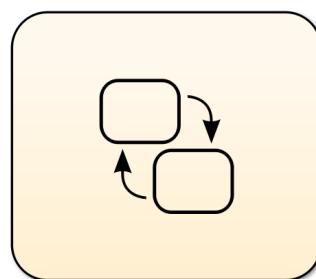


Unit Delay

Figura 38 – Bloco Unit Delay no Simulink

O bloco **Stateflow Chart** é utilizado para modelar sistemas baseados em estados, como máquinas de estados finitas ou sistemas lógicos. Esse bloco é essencial para modelar o comportamento sequencial de sistemas complexos.

- **Como funciona:** O bloco **Stateflow Chart** permite criar um gráfico de estados, onde o modelo é dividido em estados e transições entre eles, com base em condições lógicas. Ele permite implementar lógica de controle avançada, como sequências de eventos e comportamento condicional.
- **Aplicações:** O **Stateflow Chart** é amplamente utilizado em sistemas que requerem tomada de decisão baseada em eventos ou condições, como sistemas de controle sequencial, automação industrial, ou sistemas de controle de tráfego.

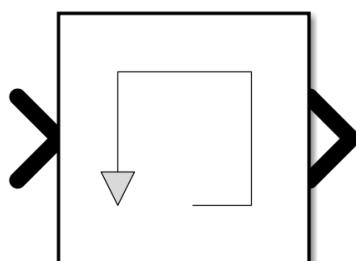


Stateflow Chart

Figura 39 – Bloco Stateflow Chart no Simulink

O bloco **Memory** armazena temporariamente um valor para que ele possa ser acessado em uma etapa posterior da simulação. Ele é útil para manter informações sobre o estado de uma variável ao longo da simulação.

- **Como funciona:** O bloco **Memory** armazena o valor de um sinal de entrada e o disponibiliza para ser utilizado em um estágio posterior da simulação. A informação armazenada é mantida durante o ciclo de simulação.
- **Aplicações:** O **Memory** é usado para modelar sistemas onde uma variável precisa ser armazenada para ser utilizada em uma etapa futura, como em sistemas de controle com valores passados, ou quando é necessário reter valores para comparar com novos sinais durante a simulação.



Memory

Figura 40 – Bloco Memory no Simulink

3.5 Máscaras de Blocos e Criação de Submodelos

Quando você cria modelos no Simulink, pode se deparar com a necessidade de esconder detalhes ou de reutilizar partes do modelo em diferentes cenários. Para isso, o Simulink oferece duas funcionalidades: as **máscaras de blocos** e a **criação de subsistemas**. Ambas são ferramentas que ajudam a organizar e simplificar o modelo, além de permitir a reutilização e a personalização de blocos.

3.5.1 Máscaras de Blocos

A **máscara de bloco** é uma ferramenta que permite personalizar a aparência e o comportamento de um bloco no Simulink, ocultando ou alterando as suas propriedades visuais e interativas. Com as máscaras, você pode transformar um bloco simples em um bloco mais complexo e visualmente adequado, além de criar interfaces de configuração personalizadas.

- **Como funciona:**

- Para criar uma máscara, você seleciona um bloco e clica em "Create Mask" no menu de contexto do bloco. Isso cria uma máscara para o bloco, permitindo que você adicione entradas e saídas personalizadas, como parâmetros de entrada configuráveis.
- Você pode editar a aparência do bloco, adicionando rótulos, campos de entrada e opções de configuração, facilitando a interação com o modelo. As máscaras também permitem a inclusão de scripts ou funções MATLAB para definir o comportamento do bloco de forma dinâmica.

- **Aplicações:**

- As máscaras são úteis quando você deseja ocultar a complexidade de um submodelo ou agrupar múltiplos blocos em um único bloco com uma interface configurável. Isso torna o modelo mais legível e permite uma melhor organização do sistema.
- Elas são frequentemente usadas em modelos que envolvem componentes repetitivos, como circuitos elétricos ou sistemas de controle com estruturas similares, onde você pode criar uma máscara para representar uma parte complexa do modelo de forma compacta e reutilizável.



Figura 41 – Exemplo de Máscara de Bloco no Simulink

3.5.2 Criação de Submodelos

Os **submodelos** (ou subsistemas) são uma maneira eficiente de organizar modelos complexos, agrupando blocos relacionados em uma estrutura hierárquica. Ao criar submodelos, você pode simplificar o modelo principal e tornar a estrutura geral mais clara e fácil de entender.

- **Como funciona:**

- Para criar um submodelo, basta selecionar um conjunto de blocos no espaço de modelagem e agrupá-los. Após agrupar, você pode clicar com o botão direito e selecionar a opção “Create Subsystem”. O Simulink irá criar automaticamente um novo bloco que contém todos os blocos selecionados.
- O submodelo pode ser configurado para aceitar entradas e fornecer saídas, assim como um bloco comum, e você pode interagir com ele da mesma forma que com qualquer outro bloco. O submodelo pode ser aberto para edição, permitindo que você altere sua estrutura interna, mas sem modificar o modelo principal.

- **Aplicações:**

- Submodelos são úteis quando você tem uma parte do sistema que é complexa, mas que precisa ser reutilizada em diferentes pontos do modelo. Isso ajuda a manter o modelo limpo e organizado, ao mesmo tempo que oculta a complexidade interna.
- Eles são frequentemente usados em modelos de sistemas de controle, circuitos, ou simulações de sistemas físicos, onde diferentes partes do sistema possuem estruturas semelhantes ou idênticas.

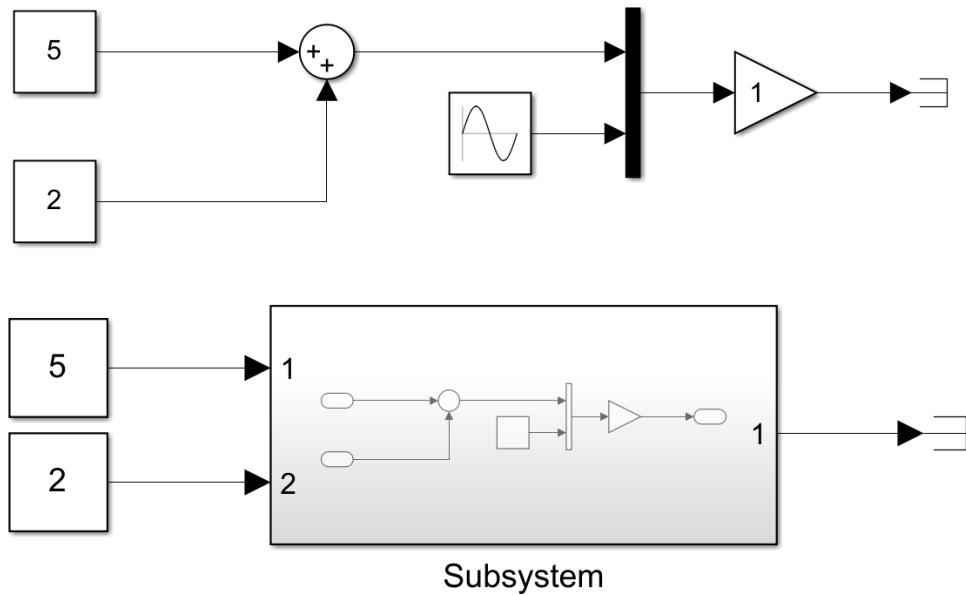


Figura 42 – Exemplo de Submodelo no Simulink

Você pode combinar o uso de **máscaras** e **submodelos** para criar blocos altamente configuráveis e reutilizáveis. Isso permite que você personalize completamente a interface de um submodelo e forneça aos usuários uma maneira de interagir com ele de forma intuitiva.

- **Como funciona:**

- Primeiro, crie um submodelo com os blocos necessários e, em seguida, aplique uma máscara para personalizar a interface. A máscara pode incluir entradas e saídas configuráveis que correspondem aos parâmetros do submodelo.
- Isso é útil quando você deseja criar um componente reutilizável que, ao ser inserido em outros modelos, tenha uma interface simplificada para configuração e parametrização.

- **Aplicações:**

- O uso combinado de máscaras e submodelos é excelente para criar bibliotecas de blocos personalizados que podem ser facilmente reutilizados em diferentes projetos. Isso é especialmente importante em grandes projetos de engenharia, onde componentes comuns precisam ser integrados.



Figura 43 – Exemplo de Submodelo com Máscara no Simulink

As **máscaras de blocos** e a **criação de submodelos** são ferramentas utilizadas para organizar e simplificar modelos complexos no Simulink. Elas não só ajudam a esconder detalhes desnecessários e a melhorar a legibilidade, mas também permitem a reutilização de componentes em diferentes modelos. O uso dessas ferramentas torna o processo de modelagem mais ágil e organizado, além de possibilitar a criação de componentes personalizados e configuráveis, prontos para serem integrados em outros projetos.

Sendo assim, ao utilizar máscaras e submodelos, você pode criar modelos mais limpos, modulares e escaláveis, o que facilita a manutenção e a compreensão do sistema ao longo do tempo. Além disso, essas técnicas ajudam a reduzir o risco de erros, já que as partes complexas do modelo são isoladas e podem ser testadas independentemente antes de serem integradas.

3.6 Configurações da Simulação

Quando você executa uma simulação no Simulink, é importante configurar corretamente os parâmetros da simulação para garantir que os resultados sejam precisos e que o processo de simulação seja eficiente. As configurações de simulação determinam como o Simulink resolve o modelo e como o tempo é gerido ao longo da execução. A seguir, vamos abordar as principais configurações: **solver**, **tempo de simulação** e **step size**; as quais podem ser acessadas na aba **Model Settings**, disponível tanto no menu **Simulation** quanto no **Modeling**.

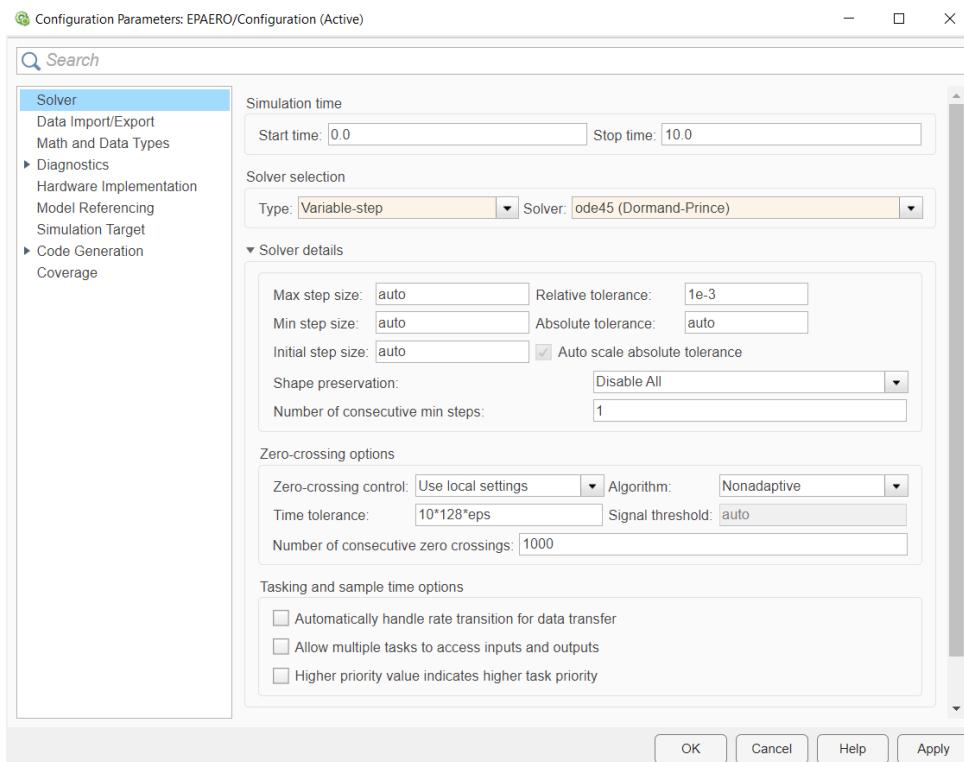


Figura 44 – Aba de configurações da simulação no Simulink

3.6.1 Solver

O solver é responsável por calcular as soluções das equações diferenciais que governam o comportamento do seu sistema durante a simulação. O tipo de solver que você escolhe tem um impacto direto na precisão e no desempenho da simulação. Existem dois tipos principais de solvers: **solvers contínuos** e **solvers discretos**.

- **Como funciona:**

- O solver contínuo resolve as equações diferenciais em sistemas que evoluem continuamente no tempo. Os solvers contínuos podem usar métodos como o **ode45** (Runge-Kutta de 4^a ordem), **ode15s** (para sistemas rígidos), ou **ode23** (para sistemas de ordem inferior).
- O solver discreto resolve as equações de sistemas que são definidos por equações de diferença. Isso é usado principalmente em modelos que envolvem sistemas digitais ou amostragem de sinais.
- O Simulink oferece também solvers híbridos, que podem lidar com sistemas que têm tanto componentes contínuos quanto discretos.

- **Aplicações:**

- A escolha do solver é crucial para garantir que a simulação seja precisa e eficiente. Solvers mais precisos geralmente levam mais tempo de computação, enquanto solvers mais rápidos podem sacrificar a precisão.

- O solver contínuo é ideal para sistemas dinâmicos com variações suaves ao longo do tempo, como circuitos elétricos analógicos ou sistemas mecânicos. Já o solver discreto é adequado para sistemas com variações abruptas ou sistemas de controle digital.

3.6.2 Tempo de Simulação

O tempo de simulação define o intervalo durante o qual o Simulink resolve o modelo. A definição do tempo de simulação é essencial para garantir que o modelo seja simulado pelo período desejado, seja para uma simulação curta ou longa.

- **Como funciona:**

- O tempo de simulação pode ser definido no **simulink model configuration** ou diretamente na interface de configuração do modelo. Você define um **tempo inicial** e um **tempo final** de simulação.
- O Simulink então resolve o modelo ao longo desse intervalo de tempo, calculando a evolução dos estados do sistema durante a simulação.
- O tempo de simulação também pode ser configurado para simulações baseadas em eventos (onde o tempo é disparado por eventos específicos do sistema) ou contínuas, onde o tempo flui linearmente de acordo com as equações diferenciais.

- **Aplicações:**

- A definição do tempo de simulação é fundamental para qualquer tipo de análise, pois ela determina por quanto tempo os resultados serão computados e armazenados.
- Um tempo de simulação mais longo pode ser necessário para sistemas que possuem transientes lentos ou que requerem análise de longo prazo. Para sistemas rápidos, um tempo de simulação curto pode ser suficiente.

3.6.3 Step Size

O **step size** ou tamanho do passo é o intervalo de tempo entre os pontos de amostragem da solução durante a simulação. A escolha do tamanho do passo tem grande influência na precisão da simulação e na eficiência computacional.

- **Como funciona:**

- Em simuladores contínuos, o tamanho do passo é um fator importante para a precisão da solução. Passos menores geralmente resultam em simulações mais precisas, mas mais demoradas, pois o solver precisa calcular mais pontos.
- O Simulink oferece uma opção de **passo fixo** (onde o passo de tempo é constante durante toda a simulação) e **passo variável** (onde o passo de tempo é ajustado dinamicamente pelo solver, dependendo da complexidade da solução e do comportamento do sistema).

- Para sistemas de controle digital ou sistemas discretos, o passo de amostragem deve ser definido com base na taxa de amostragem do sistema.

- **Aplicações:**

- O passo fixo é adequado para sistemas com comportamento previsível e controlado, como em sistemas digitais ou controladores discretos.
- O passo variável é melhor para sistemas que têm comportamento altamente não linear ou cujos estados mudam rapidamente em alguns intervalos de tempo, como sistemas mecânicos ou sistemas de controle de alta precisão.
- O passo de tempo deve ser ajustado para equilibrar precisão e eficiência computacional. Passos muito pequenos podem tornar a simulação muito lenta, enquanto passos grandes podem comprometer a precisão.

As configurações de simulação no Simulink, como o **solver**, o **tempo de simulação** e o **step size**, são parâmetros cruciais que determinam a precisão e a eficiência da simulação. Escolher o solver adequado, definir o tempo de simulação corretamente e ajustar o tamanho do passo são decisões importantes que afetam tanto os resultados quanto o tempo de execução da simulação.

A escolha do solver deve ser feita com base no tipo de sistema que está sendo modelado (contínuo ou discreto), e o **tempo de simulação** e o **step size** devem ser ajustados para garantir uma simulação precisa, sem comprometer o desempenho computacional. Com essas configurações bem ajustadas, você poderá realizar simulações de sistemas dinâmicos com maior confiança nos resultados e maior eficiência computacional.

3.6.4 Parametrização por meio do MATLAB

A parametrização de dados de simulação é um processo importante para personalizar e controlar as variáveis do seu modelo. O MATLAB oferece uma forma de configurar parâmetros de simulação, ajustando valores de entrada, variáveis de estado e outras condições de simulação. Integrar o MATLAB com o Simulink para parametrizar dados de simulação oferece flexibilidade e controle sobre os parâmetros de simulação durante a execução do modelo.

Nesse contexto, o Simulink permite o uso de variáveis definidas no MATLAB como parâmetros nos blocos de um modelo. Isso significa que você pode criar um script MATLAB para definir variáveis e, em seguida, usar essas variáveis no seu modelo Simulink.

- **Como funciona:**

- Defina variáveis no MATLAB, como **Amplitude = 10** ou **Fase = 0,3**, e então use essas variáveis nos blocos do Simulink, substituindo valores constantes diretamente nos blocos.
- O Simulink automaticamente reconhece as variáveis definidas no espaço de trabalho do MATLAB durante a simulação e as usa como entradas nos blocos do modelo.

- **Aplicações:**

- O uso de variáveis no Simulink é útil para ajustes dinâmicos e simulações paramétricas, permitindo que você altere o comportamento do modelo facilmente sem a necessidade de reconfigurar manualmente os parâmetros do bloco.
- Esse método é particularmente útil em sistemas de controle, onde os parâmetros do controlador podem ser ajustados dinamicamente a partir de um script MATLAB, facilitando o processo de otimização de parâmetros.

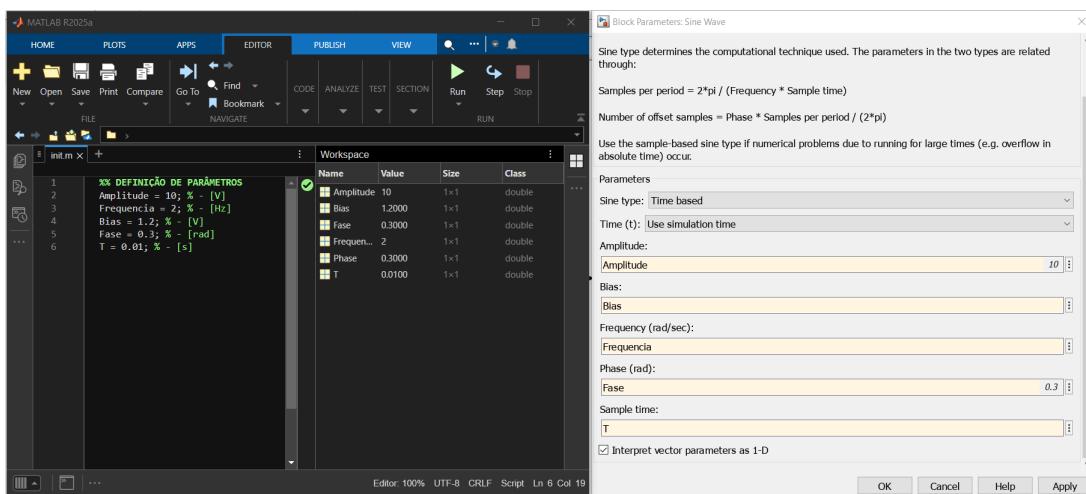


Figura 45 – Exemplo de Uso de Variáveis Definidas no Matlab como parâmetros de um bloco **Sine Wave** no Simulink

A integração entre MATLAB e Simulink é uma das características mais poderosas da plataforma, permitindo que você combine a facilidade de modelagem do Simulink com a flexibilidade do MATLAB. Essa combinação facilita a criação de modelos complexos, a análise de grandes volumes de dados e a automação de tarefas repetitivas, tornando o processo de simulação mais eficiente.

3.7 Modelagem de Sistemas

Nesta seção, abordaremos os conceitos e métodos para a **criação e organização de modelos** no Simulink. A criação de modelos no Simulink é uma etapa fundamental no desenvolvimento de simulações de sistemas dinâmicos, como sistemas mecânicos, elétricos e de controle. Organizar corretamente um modelo pode facilitar a simulação e a análise dos resultados. Vamos explorar as principais práticas e ferramentas para criar e estruturar modelos no Simulink.

3.7.1 Criação e Organização de Modelos

A criação de modelos no Simulink envolve a utilização de blocos que representam os diferentes componentes do sistema que você deseja modelar. Esses componentes podem ser físicos (como motores, circuitos elétricos, etc.) ou abstratos (como variáveis de

controle). Além disso, a organização desses blocos é crucial para garantir que o modelo seja compreensível, escalável e fácil de modificar. A seguir, vamos explorar como criar e organizar um modelo de forma eficiente no Simulink.

A criação de um modelo no Simulink começa com a definição do problema a ser simulado e a escolha dos blocos apropriados para representar os elementos do sistema. Aqui estão os passos principais para criar um modelo no Simulink:

- **Escolha dos Blocos:** O primeiro passo na criação de um modelo é selecionar os blocos necessários para representar as diferentes partes do sistema. Por exemplo, se você estiver modelando um circuito elétrico, pode escolher blocos como **Resistor**, **Capacitor**, **Indutor**, etc.
- **Arranjo dos Blocos:** Após escolher os blocos, o próximo passo é arranjar esses blocos no espaço de modelagem do Simulink. Isso pode ser feito arrastando os blocos para o espaço de trabalho e conectando-os para representar as interações entre os componentes do sistema.
- **Conexão de Blocos:** Use as linhas de conexão para conectar os blocos entre si. As linhas representam o fluxo de sinal ou informações entre os blocos. A ordem em que os blocos são conectados é importante, pois reflete a sequência de operações que o sistema realiza.
- **Configuração dos Parâmetros:** Cada bloco no Simulink possui parâmetros configuráveis que determinam o comportamento do sistema. Por exemplo, você pode definir a resistência de um resistor, o valor de um ganho ou o tempo de simulação.

Uma vez que o modelo foi criado, a organização eficiente dos blocos é crucial para garantir que o modelo seja legível e fácil de modificar. A seguir, discutiremos algumas práticas recomendadas para organizar modelos no Simulink.

- **Uso de Submodelos (Subsystems):** Para manter o modelo organizado, é uma boa prática agrupar blocos relacionados em **submodelos**. Isso ajuda a reduzir a complexidade visual do modelo, escondendo detalhes de implementação que não precisam ser visualizados no nível principal. Por exemplo, um sistema de controle pode ter um submodelo para o controlador e outro para o sistema físico.
- **Utilização de Bibliotecas de Blocos:** O Simulink permite criar bibliotecas de blocos personalizados que podem ser reutilizados em diferentes modelos. Isso é útil para sistemas complexos que possuem componentes comuns, pois você pode criar um bloco uma vez e reutilizá-lo em vários modelos.
- **Identificação Clara dos Blocos:** Certifique-se de nomear adequadamente todos os blocos e subsistemas, para que o modelo seja fácil de entender. Utilizar descrições claras nos blocos pode ajudar a identificar rapidamente o papel de cada parte do sistema.

- **Uso de Blocos de Anotação:** O Simulink oferece blocos de anotação que podem ser usados para adicionar explicações e comentários ao modelo. Esses blocos ajudam a documentar o modelo, facilitando a compreensão do sistema para outras pessoas ou para futuras modificações.
- **Organização Visual:** Organize os blocos de forma lógica e clara no espaço de modelagem. Evite sobreclarregar a tela com muitos blocos próximos uns dos outros. O uso de alinhamento e espaçamento adequado ajuda a manter o modelo limpo e fácil de entender.

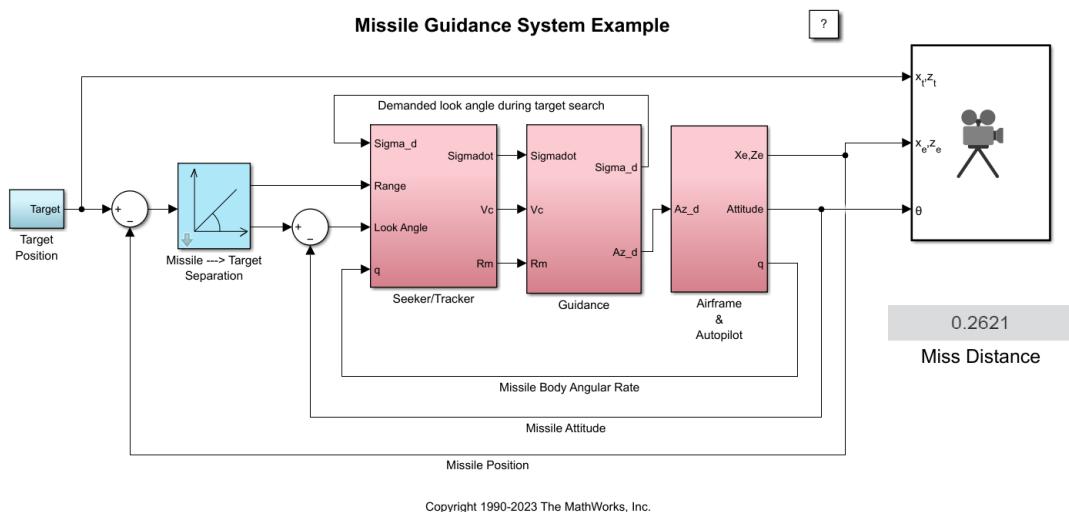


Figura 46 – Exemplo de Modelo Organizado com Submodelos no Simulink - Adaptado de MathWorks (2025)

3.7.2 Testando e Validando o Modelo

Após criar e organizar o modelo, é importante testá-lo e validá-lo para garantir que ele funciona como esperado. A validação pode ser feita por meio de simulações, onde você define as entradas e observa as saídas do modelo. Algumas práticas recomendadas para testar e validar seu modelo incluem:

- **Definição de Condições Iniciais:** Para testar o modelo, defina condições iniciais adequadas para as variáveis de estado. Isso ajudará a simular o comportamento do sistema a partir de um estado inicial específico.
- **Verificação de Resultados:** Após rodar a simulação, verifique os resultados usando blocos como **Scope**, **Display** ou **To Workspace** para visualizar as saídas do modelo e verificar se elas estão dentro do esperado.
- **Ajustes no Modelo:** Se necessário, ajuste os parâmetros ou a estrutura do modelo com base nos resultados da simulação. Isso pode envolver a modificação de parâmetros de blocos ou a reorganização dos subsistemas para otimizar a simulação.

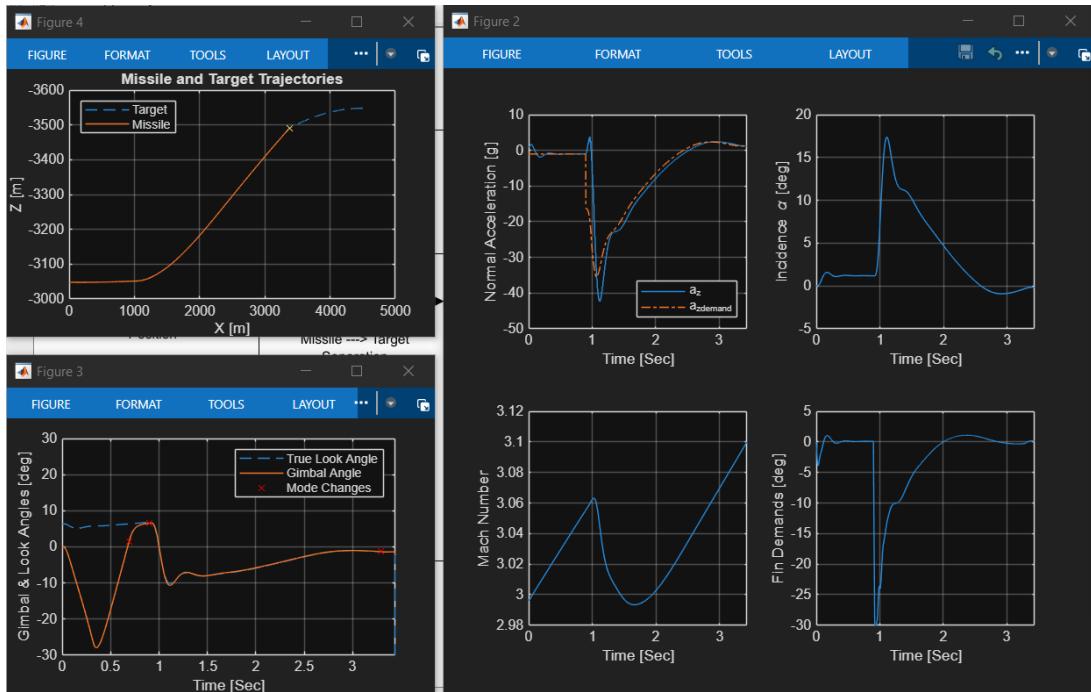


Figura 47 – Visualização de Resultados de uma Simulação feita pelo Simulink - Adaptado de Mathworks (2025).

A criação e organização de modelos no Simulink é uma habilidade essencial para a modelagem de sistemas dinâmicos e de controle. Ao seguir as práticas recomendadas para escolha e organização dos blocos, como o uso de submodelos e bibliotecas de blocos, você pode criar modelos mais eficientes, legíveis e fáceis de modificar. Além disso, a validação do modelo por meio de simulações é crucial para garantir que o modelo esteja correto e produza os resultados esperados.

A organização adequada do modelo também é importante para facilitar futuras modificações e otimizações, garantindo que o sistema seja escalável e comprehensível, tanto para você quanto para outros usuários que possam precisar trabalhar com o modelo no futuro.

4 Exemplo interativo: Simulação de lançamento de foguete para inserção de carga útil em órbita baixa

Como forma de aplicar os conhecimentos adquiridos ao longo desta apostila, foi desenvolvido um projeto integrador que simula, por meio das ferramentas MATLAB e Simulink, o lançamento de um foguete com o objetivo de inserir uma carga útil em órbita. Esse projeto busca unir conceitos teóricos e práticos relacionados à modelagem, simulação e análise de sistemas dinâmicos no contexto da engenharia aeroespacial.

O lançamento de veículos espaciais é um dos desafios mais multidisciplinares da engenharia, exigindo a integração de diversas áreas como dinâmica, controle, propulsão, sistemas embarcados e aerodinâmica. Nesse cenário, ferramentas como o MATLAB e o Simulink se tornam úteis para prever o comportamento do sistema antes da construção e dos testes físicos, permitindo iterar rapidamente sobre o projeto e minimizar riscos e custos.

O modelo desenvolvido contempla as principais fases de um lançamento orbital, desde a ignição e subida atmosférica até a separação de estágios e inserção em órbita. Foram consideradas variáveis como massa do foguete, empuxo dos motores, resistência do ar, gravidade e aceleração ao longo da trajetória. Além disso, a simulação foi estruturada para ser modular e parametrizável, permitindo ajustes e reusabilidade dos blocos para estudos complementares.

Através da combinação das ferramentas vistas nas seções de MATLAB e Simulink, o leitor poderá compreender como modelar o comportamento de sistemas reais complexos, além de visualizar, monitorar e ajustar os parâmetros envolvidos no lançamento. O projeto está disponível em um repositório Git, permitindo que outros estudantes explorem, modifiquem e expandam a simulação conforme seus próprios objetivos acadêmicos ou profissionais.

O modelo integrado em MATLAB e Simulink utilizou conceitos desenvolvidos na disciplina Mecânica de Voo Aeroespacial, ministrada pelo professor André Luis da Silva para o curso de engenharia aeroespacial da Universidade Federal de Santa Maria. Sendo assim, os códigos fontes utilizados para a implementação do modelo são de sua autoria.

4.1 Equações de movimento

Para descrever a trajetória do foguete durante seu lançamento e inserção orbital, adotou-se um modelo simplificado de **três graus de liberdade (3-DOF)**. Nesse modelo, desconsidera-se a rotação do corpo do foguete em torno de seus próprios eixos, assumindo que a velocidade do centro de massa está sempre alinhada com o eixo longitudinal (ou eixo de simetria) do veículo. Essa simplificação é comum em análises iniciais de desempenho, especialmente quando o objetivo é avaliar a dinâmica translacional do foguete em trajetória atmosférica e orbital, sem considerar os efeitos aerodinâmicos de controle ou instabilidades laterais.

A abordagem 3-DOF permite a modelagem do movimento do foguete levando em conta apenas sua posição e orientação da velocidade no espaço tridimensional. O movimento é expresso em um sistema de coordenadas geodésico (referenciado à superfície da Terra).

As **variáveis de estado** adotadas no modelo são:

- **Velocidade relativa (magnitude)**: velocidade do foguete em relação à superfície do planeta, levando em conta a rotação da Terra.
- **Azimute da velocidade**: ângulo entre a projeção da velocidade sobre o plano horizontal e o norte geográfico.
- **Elevação da velocidade**: ângulo entre a direção da velocidade e o plano local do horizonte.
- **Distância radial (raio)**: distância do centro da Terra até o centro de massa do foguete.
- **Latitude**: posição angular do foguete em relação ao equador terrestre.
- **Longitude**: posição angular do foguete em relação ao meridiano de referência (Greenwich).

Esse conjunto de variáveis permite descrever completamente a trajetória do foguete em termos de sua posição e direção de movimento ao longo do tempo. A modelagem das equações de movimento considera os principais efeitos físicos envolvidos, como a força gravitacional, o empuxo dos motores, a resistência do ar (durante o voo atmosférico) e a curvatura da Terra.

As equações de movimento completas que regem a evolução dessas variáveis ao longo do tempo serão apresentadas a seguir.

$$\dot{v} = \frac{1}{m} (-D + f_T \cos \varepsilon \cos \mu - mg_c \sin \phi + mg_\delta \cos A \cos \phi) + \\ - r\omega_e^2 \cos \delta (\cos A \sin \delta \cos \phi - \cos \delta \sin \phi) \quad (23)$$

$$\dot{A} = \frac{1}{mv \cos \phi} (-f_y + f_T \sin \mu - mg_\delta \sin A) - \frac{1}{v \cos \phi} 2\omega_e (\cos A \cos \delta \sin \phi - \sin \delta \cos \phi) + \\ \frac{1}{v \cos \phi} \left(r\omega_e^2 \sin A \sin \delta \cos \delta + \frac{v^2}{r} \sin A \tan \delta \cos^2 \phi \right) \quad (24)$$

$$\dot{\phi} = \frac{1}{mv} (L + f_T \cos \mu \sin \varepsilon - mg_c \cos \phi - mg_\delta \cos A \sin \phi) + \\ \frac{1}{v} \left(2\omega_e \sin A \cos \delta + \frac{v^2}{r} \cos \phi + r\omega_e^2 \cos \delta (\cos A \sin \delta \sin \phi + \cos \delta \cos \phi) \right) \quad (25)$$

Complementam-se essas expressões com as equações de **cinemática de translação**, que definem a variação da posição do foguete em termos da sua velocidade:

$$\dot{r} = v \sin \phi \quad (26)$$

$$\dot{\lambda} = \frac{v \cos \phi \sin A}{r \cos \delta} \quad (27)$$

$$\dot{\delta} = \frac{v \cos \phi \cos A}{r} \quad (28)$$

As equações de movimento apresentadas anteriormente foram implementadas dentro de um bloco **MATLAB Function** no ambiente Simulink. Esse tipo de bloco permite a inserção direta de código em linguagem MATLAB, possibilitando a definição de equações diferenciais personalizadas e a execução de algoritmos que não podem ser representados facilmente utilizando blocos gráficos padrão.

A escolha por utilizar o *MATLAB Function* se justifica pela complexidade e densidade matemática das equações envolvidas, que demandariam uma estrutura extremamente extensa se fossem montadas apenas com blocos convencionais do Simulink. Além disso, essa abordagem oferece maior flexibilidade para a manutenção e ajustes do modelo, uma vez que o código pode ser facilmente editado, comentado e validado.

Dentro do bloco, as equações foram escritas em sua forma diferencial, conforme apresentadas anteriormente, e implementadas como um sistema de equações de estados, onde as derivadas das variáveis são calculadas a partir dos valores instantâneos das próprias variáveis de estado e das entradas (como empuxo, massa, e forças aerodinâmicas). O bloco então devolve como saída os vetores de derivadas.

```

function Xp = dinamica_foguete(X, F_t, m, D, f_y, L, g_c, g_d, omega_e, Re, l_trilho)
% Função para a dinâmica de translacão de um foguete com respeito ao referencial PCPF
% Sistema de referência: aerodinâmico
% Sistema de coordenadas: esférico

% Vetor de estado
V = X(1); % Velocidade
A = X(2); % Ângulo de azimute
phi = X(3); % Ângulo de elevação
r = X(4); % Distância radial
delta = X(5); % Latitude
lon = X(6); % Longitude
h = r - Re; % Altitude

% Equações de cinemática de translacão
rp = V * sin(phi);
deltap = (V / r) * cos(phi) * cos(A);
lonp = (V * cos(phi) * sin(A)) / (r * cos(delta));

% Equações de dinâmica de translacão
Vp = (1 / m) * (F_t * cos(phi) * cos(A) - D - m * g_c * sin(phi) + m * g_d * cos(phi) * cos(A) - ...
m * omega_e^2 * r * cos(delta) * (cos(phi) * cos(A) * sin(delta) - sin(phi) * cos(delta)));
Ap = (1 / m) * (V^2 / r) * cos(phi)^2 * sin(A) * tan(delta) + F_t * sin(phi) + ...
f_y - m * g_d * sin(A) + m * omega_e^2 * r * sin(A) * sin(delta) * cos(delta) - ...
2 * m * omega_e * V * (sin(phi) * cos(A) * cos(delta) - cos(phi) * sin(delta));
phiip = (1 / (m * V)) * (m * (V^2 / r) * cos(phi) + F_t * sin(phi) * cos(0) + L - m * g_c * cos(phi) - ...
m * g_d * sin(phi) * cos(A) + m * omega_e^2 * r * cos(delta) * (sin(phi) * cos(A) * sin(delta) + ...
cos(phi) * cos(delta)) + 2 * m * omega_e * V * sin(A) * cos(delta));

```

Figura 48 – Código das equações de movimento

Para resolver numericamente o sistema de equações diferenciais definido no bloco *MATLAB Function*, foi utilizado o bloco **Integrator** do Simulink. Esse bloco é responsável por realizar a integração das derivadas das variáveis de estado ao longo do tempo, ou

seja, ele acumula as variações instantâneas calculadas em cada passo da simulação para atualizar o valor atual de cada variável.

Para que o integrador possa iniciar corretamente a simulação, é necessário definir as **condições iniciais** do sistema — ou seja, os valores das variáveis de estado no instante $t = 0$. Esses valores refletem a situação inicial do foguete no momento do lançamento e são fundamentais para garantir que a simulação produza resultados coerentes com o cenário real que se deseja estudar.

As condições iniciais definidas para o modelo incluem, por exemplo:

- Velocidade inicial v_0 , geralmente igual a zero no lançamento vertical.
- Altura inicial r_0 , correspondente ao raio da Terra somado à altitude da plataforma de lançamento.
- Azimute inicial A_0 e elevação inicial ϕ_0 , que definem a direção de lançamento do foguete.
- Latitude e longitude iniciais (δ_0, λ_0), correspondentes à localização geográfica da base de lançamento.

Esses valores foram parametrizados em um código de inicialização da simulação, escrito em Matlab. Dessa forma, as condições iniciais do lançamento podem ser alteradas conforme desejado.

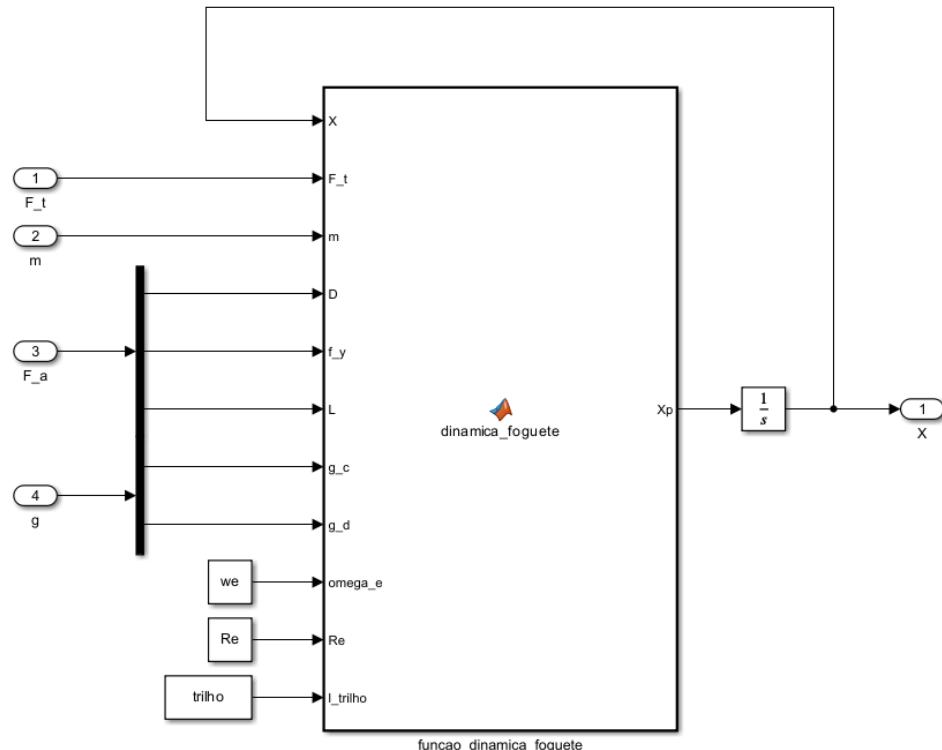


Figura 49 – Bloco Matlab Function das equações de movimento com integrador

4.2 Criação de máscaras e parametrização dos subsistemas

Para tornar o modelo mais organizado e reutilizável, o subsistema que contém o bloco *MATLAB Function* (onde estão implementadas as equações de movimento) e os integradores foi encapsulado e recebeu uma **máscara**. Essa máscara permite que parâmetros importantes, como os estados iniciais e o comprimento do trilho de lançamento, sejam definidos de forma centralizada e facilmente acessível pelo usuário, sem a necessidade de abrir o interior do subsistema.

A seguir, apresenta-se um **passo a passo** para criar uma máscara e parametrizar variáveis externas como entradas para os blocos internos:

1. Encapsular o modelo em um subsistema

Antes de criar a máscara, é necessário agrupar os blocos desejados em um subsistema:

1. Selecione os blocos desejados (por exemplo, o integrador e o bloco *MATLAB Function*).
2. Clique com o botão direito e selecione **Create Subsystem from Selection**.
3. O Simulink criará automaticamente um novo bloco de subsistema encapsulando os blocos selecionados.

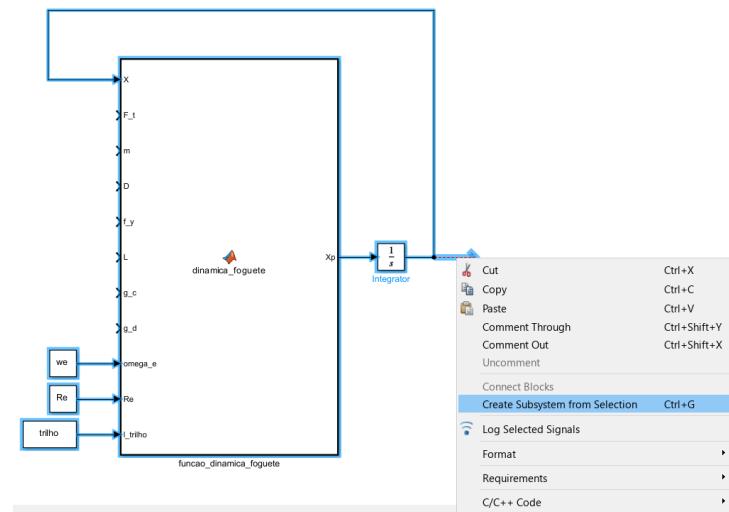


Figura 50 – Criação de um subsistema

2. Criar uma máscara para o subsistema

1. Clique com o botão direito no bloco do subsistema criado e selecione **Mask → Create Mask**.
2. Será aberta a janela de edição da máscara, com as abas *Icon*, *Parameters* & *Dialog*, *Initialization*, entre outras.

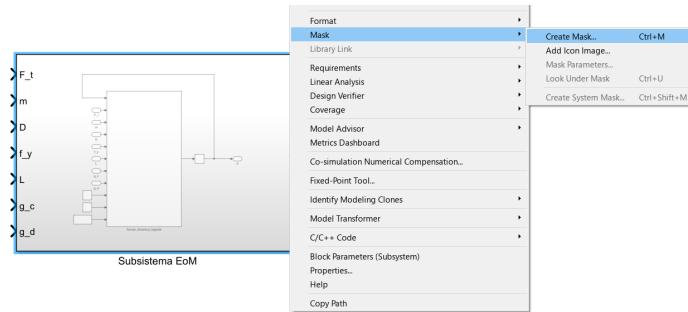


Figura 51 – Criação da máscara

3. Adicionar parâmetros de entrada à máscara

1. Na aba **Parameters & Dialog**, clique em **Add Parameter** (ícone de “+”) para cada variável que deseja tornar parametrizável.
2. Para cada parâmetro, defina:
 - **Prompt:** o nome que será exibido na interface da máscara (ex: “Massa inicial”).
 - **Name:** o nome da variável (ex: `m0`).
3. Você pode adicionar campos do tipo *edit*, *popup*, *checkbox*, conforme a necessidade do tipo de entrada.

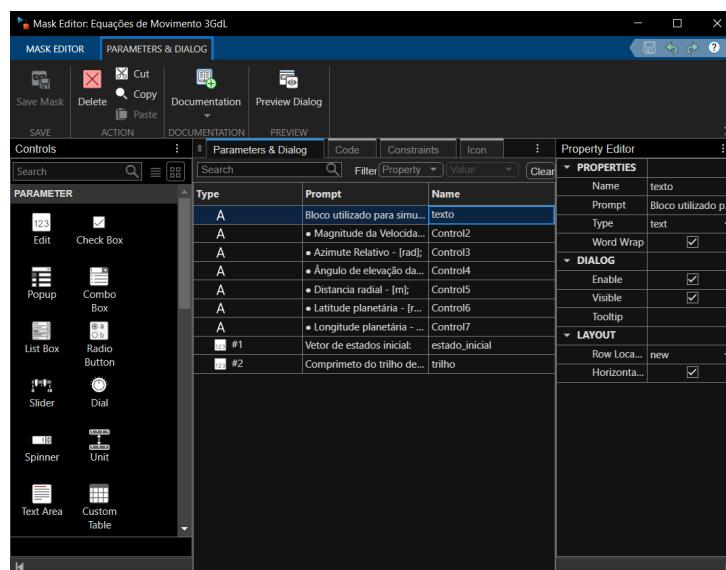


Figura 52 – Interface de criação da máscara com parâmetros configuráveis

4. Associar os parâmetros às variáveis internas do subsistema

Os nomes das variáveis definidos na aba **Parameters & Dialog** estarão disponíveis dentro do subsistema e podem ser utilizados:

- Diretamente nos blocos internos (por exemplo, nos blocos *Gain*, *Integrator* ou *Constant*).
- No código do bloco *MATLAB Function*, onde podem ser passadas como entradas da função.

Para isso, conecte blocos **Constant** no subsistema e nomeie-os com os mesmos nomes dos parâmetros da máscara (por exemplo, `trilho`). Esses blocos receberão os valores definidos na interface da máscara e os repassarão aos blocos internos.

4.3 Demais módulos e integração

Além do módulo principal responsável pelas equações de movimento, o simulador completo desenvolvido também inclui outros módulos fundamentais para representar com fidelidade os principais efeitos físicos atuantes sobre o foguete durante o lançamento. Cada um desses módulos foi implementado em um **subsistema próprio**, com uma **máscara parametrizável**, seguindo a mesma abordagem detalhada anteriormente na seção das equações de movimento.

Os módulos implementados são:

- **Modelo Atmosférico:** responsável por fornecer a densidade do ar, pressão atmosférica e outros parâmetros importantes em função da altitude. Essas variáveis são essenciais para o cálculo da força de arrasto.
- **Modelo Aerodinâmico:** calcula a força de arrasto que atua sobre o foguete com base na velocidade relativa, densidade do ar, área de referência e coeficiente de arrasto. Este módulo fornece a força aerodinâmica como uma saída, que alimenta o sistema dinâmico principal.
- **Modelo Propulsivo:** calcula o empuxo gerado pelos motores, considerando os parâmetros do motor de cada estágio. Também fornece a taxa de variação da massa do foguete. Vale ressaltar que o modelo implementado é válido unicamente para um foguete com 3 estágios.
- **Modelo Gravitacional:** determina as componentes radial e latitudinal da força gravitacional. Essas forças são utilizadas nas equações de movimento.

Cada um desses módulos foi encapsulado em um **subsistema com máscara**, de forma que seus parâmetros podem ser alterados.

A construção dessas máscaras seguiu o mesmo processo descrito na seção anterior, utilizando a funcionalidade *Mask Editor* do Simulink para criar campos de entrada configuráveis e expor os parâmetros necessários para uso externo.

A integração entre os módulos foi feita por meio da interconexão lógica das **entradas**, **saídas** e **realimentações** de cada subsistema. A comunicação entre os módulos segue o fluxo natural da simulação física:

- O **modelo atmosférico** fornece dados para o **modelo aerodinâmico**.

- O **modelo propulsivo** fornece empuxo e variação de massa para o sistema de equações de movimento.
- O **modelo aerodinâmico** fornece a força de arrasto que também alimenta o sistema dinâmico.
- O **modelo gravitacional** calcula a gravidade local usada diretamente nas equações de movimento.
- Os estados de interesse são realimentados aos blocos que são necessários.

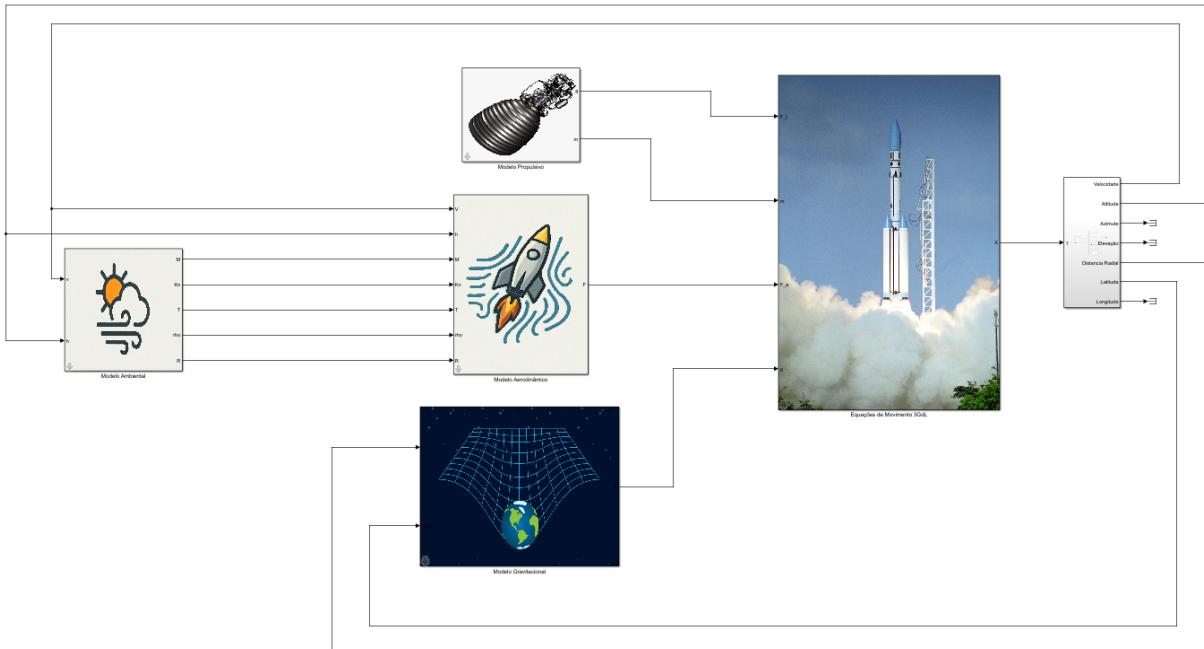


Figura 53 – Exemplo da integração entre os módulos do simulador no ambiente Simulink

Essa arquitetura modular facilita o entendimento, a depuração e a reutilização do modelo em outros projetos. Além disso, permite realizar simulações com diferentes níveis de fidelidade, substituindo ou atualizando apenas partes específicas do sistema, sem comprometer a estrutura geral da simulação. Um ponto a se ressaltar é o subsistema que recebe os estados calculados pelas equações de movimento. Nele, foi utilizado um bloco **To Workspace**, com o intuito de salvar os resultados da simulação para pós processamento.

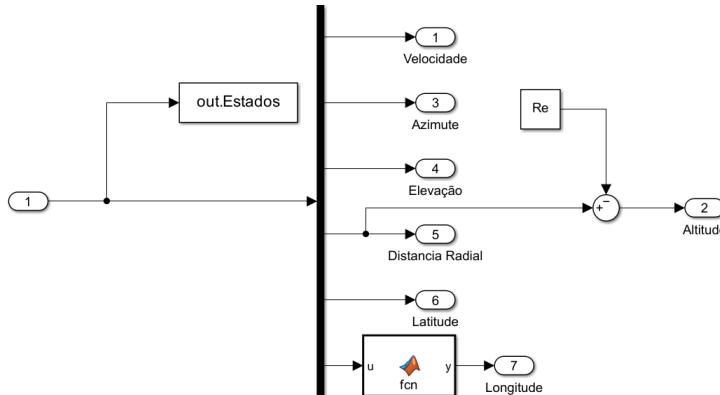


Figura 54 – Subsistema utilizado para armazenamento e adequação dos estados

4.4 Configurações da simulação

Para garantir que o modelo seja executado corretamente, com controle sobre os parâmetros e estados do sistema ao longo do tempo, foram realizadas configurações específicas na simulação, diretamente nas propriedades do modelo do Simulink. Nesta seção, apresentamos duas dessas configurações importantes: o uso de funções de **callback** e a definição de **parâmetros do solver**.

O Simulink permite que sejam executadas instruções MATLAB automaticamente no início ou no final de uma simulação. Essas instruções são definidas nas chamadas **funções de callback**, e são extremamente úteis para realizar tarefas como:

- Inicializar variáveis no **base workspace**;
- Limpar gráficos ou arquivos antes de iniciar uma nova simulação;
- Salvar dados ou gerar relatórios ao final da simulação;
- Configurar caminhos ou parâmetros automaticamente.

No projeto desenvolvido, foram criadas duas funções específicas:

- Uma função de **callback de inicialização**, responsável por definir os valores iniciais de variáveis importantes na **Workspace**, nomeada **init.m**.
- Uma função de **callback de parada**, utilizada para executar uma rotina de exibição dos estados e da trajetória do foguete ao longo do tempo, nomeada como **post_processing.m**.

Para definir essas funções:

1. Acesse o menu **Modeling > Model Properties**.
2. No painel superior, selecione a seção **Callbacks**.
3. Em **InitFcn**, insira o código MATLAB que será executado automaticamente no início da simulação.

4. Em **StopFcn**, insira o código a ser executado quando a simulação for finalizada.

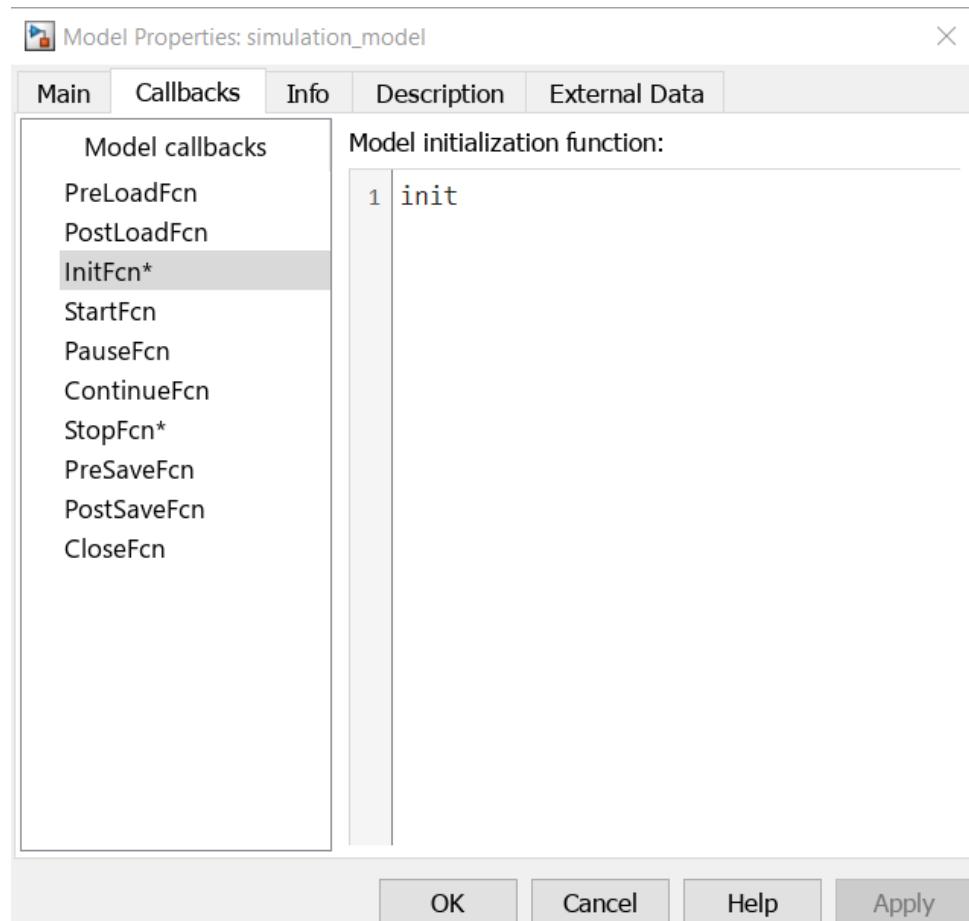


Figura 55 – Exemplo de configuração de callbacks no menu de simulação

Outra configuração importante realizada no projeto foi a definição do **solver** e do **passo de integração temporal**. O solver é o algoritmo numérico responsável por resolver as equações diferenciais do sistema ao longo do tempo. Para este projeto, foi selecionado o solver:

- **ode4 (Runge-Kutta de quarta ordem)** – um método explícito e amplamente utilizado, que oferece um bom equilíbrio entre precisão e desempenho computacional.

Além disso, foi definido um **passo de tempo fixo** de 0.5 segundos, garantindo consistência na resolução temporal da simulação. Essa escolha de passo fixo, em vez de adaptativo, oferece maior previsibilidade na coleta de dados e na análise posterior dos resultados. Essas configurações foram realizadas no menu:

- **Modeling > Model Settings > Solver**

As principais opções definidas foram:

- **Solver type:** Fixed-step
- **Solver:** ode4 (Runge-Kutta)
- **Fixed-step size:** 0.5

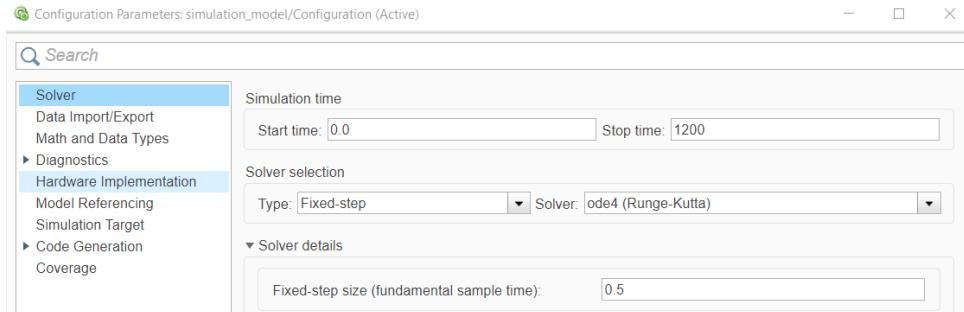


Figura 56 – Configurações do solver definidas no projeto

4.5 Operação do modelo

O modelo desenvolvido foi projetado para operar de forma parametrizada. A operação da simulação se dá principalmente por meio de dois arquivos auxiliares escritos em linguagem MATLAB, que estão conectados ao Simulink através de **funções de callback**: o arquivo `init.m`, executado no início da simulação, e o arquivo `post_processing.m`, executado ao final.

As **condições iniciais do lançamento**, como posição geográfica (latitude e longitude), altitude da plataforma, ângulos iniciais de lançamento (azimute e elevação) e velocidade inicial, podem ser modificadas diretamente no arquivo `init.m`. Esse arquivo corresponde ao **callback de inicialização** da simulação e é automaticamente executado antes de o modelo começar a rodar.

```

95 % CONDIÇÕES INICIAIS
96 h0 = 0; % Altitude no local de lançamento
97 V0 = 7.2921150e-5; % Magnitude da velocidade inicial - [m/s]
98 A0 = 85.2759789202593; % Ângulo de azimute inicial - [º]
99 phi0 = 75.5; % Ângulo de elevação da velocidade inicial - [º]
100 r0 = Re + h0; % Distância radial inicial - [m]
101 delta0 = -2.3267844; % Latitude do local de lançamento - [º]
102 lon0 = -44.4111042; % Longitude do local de lançamento - [º]
103

```

Figura 57 – Configurações do solver definidas no projeto

Alterando essas variáveis no `init.m`, o usuário pode simular diferentes cenários de lançamento e observar como as trajetórias e órbitas resultantes se modificam em função das condições iniciais adotadas.

Além das condições de lançamento, também estão presentes no `init.m` os **parâmetros do foguete**, como:

- Massas de propelente e estrutura de cada estágio;
- Impulso específico de cada estágio;

- Instantes temporais de ignição, término da queima e separação de cada estágio;
- Massa da carga útil
- Áreas e comprimento de referência;
- Demais parâmetros utilizados no modelo.

Esses parâmetros alimentam diretamente os subsistemas do modelo (como o propulsivo e aerodinâmico), permitindo que o comportamento do foguete seja personalizado de acordo com diferentes configurações físicas e de missão.

Após a finalização da simulação, o arquivo `post_processing.m` é automaticamente executado como **callback de parada**. Esse script é responsável por processar os dados gerados durante a simulação e apresentar os resultados de forma gráfica e interpretável.

As principais funcionalidades implementadas no `post_processing.m` incluem:

- **Plotagem das variáveis de estado** ao longo do tempo.
- **Cálculo dos elementos orbitais keplerianos**, com base nas condições finais da trajetória.
- **Visualização da trajetória do foguete no globo terrestre**.

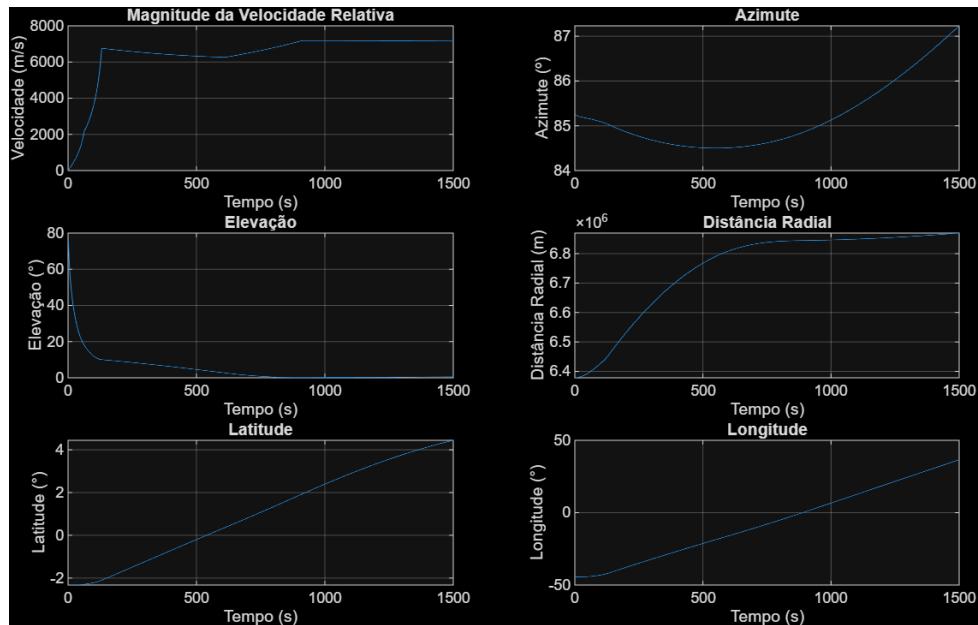


Figura 58 – Exemplo de visualização temporal dos estados do foguete

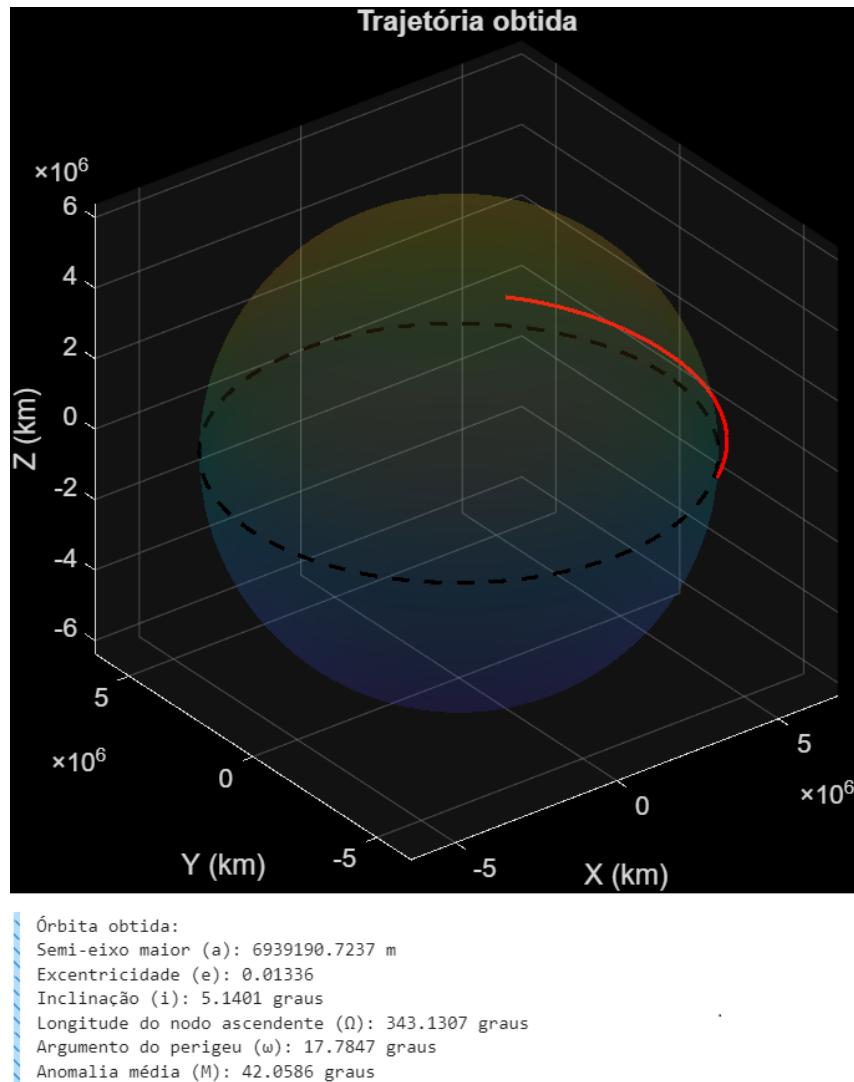


Figura 59 – Exemplo de visualização da trajetória orbital e parâmetros orbitais obtidos

Em resumo, a operação do modelo é feita por meio da edição do arquivo `init.m`, a qual deve ser feita com o intuito de adequar a simulação ao cenário desejado. Posteriormente deve-se iniciar a simulação do modelo no Simulink, a qual irá gerar os dados que serão processados no código `post_processing.m`. Essa estrutura modular permite alterar rapidamente o cenário da missão e obter análises dos resultados, consolidando a integração entre o ambiente MATLAB e Simulink.