# Apêndice: Funções MATLAB Utilizadas

## 1. polyfit

**Descrição**: A função polyfit realiza uma regressão polinomial nos dados fornecidos, ajustando um polinômio de grau especificado.

**Sintaxe**:

```
p = polyfit(x, y, n)
```

- **Entradas**:
  - x: Vetor com os valores das variáveis independentes.
  - y: Vetor com os valores das variáveis dependentes.
  - n: Grau do polinômio a ser ajustado.
- **Saída**:
  - p: Vetor de coeficientes do polinômio ajustado, ordenados do maior para o menor grau.

**Exemplo**:

```
x = [0, 1, 2, 3];
y = [1, 2, 1.5, 0.5];
p = polyfit(x, y, 2); % Ajusta um polinômio de grau 2 aos dados
```

## 2. polyval

**Descrição**: Avalia um polinômio em pontos específicos utilizando seus coeficientes.

**Sintaxe**:

```
y_fit = polyval(p, x)
```

- **Entradas**:
  - p: Vetor de coeficientes do polinômio (saída de polyfit).
  - x: Valores nos quais o polinômio será avaliado.
- **Saída**:
  - y_fit: Valores do polinômio ajustado em cada ponto de x.

**Exemplo**:

```
y_fit = polyval(p, x); % Avalia o polinômio em x
```

## 3. fminsearch

**Descrição**: Realiza minimização de uma função sem restrições utilizando o método de Nelder-Mead.

**Sintaxe**:

```
x_min = fminsearch(fun, x0)
```

- **Entradas**:
  - fun: Função objetivo que será minimizada. Deve ser definida como uma função anônima ou arquivo de função.
  - x0: Chute inicial para a minimização.
- **Saída**:
  - x_min: Valores das variáveis que minimizam a função objetivo.

**Exemplo**:

```
obj_fun = @(x) (x - 3)^2; % Função objetivo
x_min = fminsearch(obj_fun, 0); % Encontra o mínimo
```

## 4. simulannealbnd

**Descrição**: Implementa o algoritmo de recozimento simulado (simulated annealing) para encontrar mínimos globais de funções com limites.

**Sintaxe**:

```
x_min = simulannealbnd(fun, x0, lb, ub, options)
```

- **Entradas**:
  - fun: Função objetivo que será minimizada.
  - x0: Vetor inicial.
  - lb: Vetor de limites inferiores.
  - ub: Vetor de limites superiores.
  - options: Estrutura de opções (configurável com optimoptions).
- **Saída**:
  - x_min: Parâmetros que minimizam a função objetivo.

**Exemplo**:

```
opts = optimoptions('simulannealbnd', 'Display', 'iter');
x_min = simulannealbnd(@(x) x^2, 10, -5, 5, opts);
```

## 5. penalty

**Descrição**: Função definida pelo usuário para adicionar penalidades a parâmetros fora dos limites especificados.

**Sintaxe**:

```
penalty = @(params) sum(max(0, params - upper_limit).^2 + max(0, lower_limit - params).^2);
```

- **Entradas**:

    - params: Vetor de parâmetros.
    - upper_limit: Limites superiores.
    - lower_limit: Limites inferiores.
- **Saída**:
    - Valor da penalidade acumulada.

**Exemplo**:

```
penalty_value = penalty([10, 5], [9, 6], [3, 4]);
```

## 6. fill

**Descrição**: Cria áreas preenchidas em um gráfico, utilizadas para destacar intervalos de confiança.

**Sintaxe**:

```
fill(X, Y, Color, 'FaceAlpha', alpha)
```

- **Entradas**:
    - X: Coordenadas horizontais do polígono.
    - Y: Coordenadas verticais do polígono.
    - Color: Cor do preenchimento.
    - FaceAlpha: Transparência do preenchimento (0 a 1).

**Exemplo**:

```
fill([x; flipud(x)], [y_upper; flipud(y_lower)], 'b', 'FaceAlpha', 0.2);
```

## 7. gradient

**Descrição**: Calcula o gradiente de uma função ou vetor.

**Sintaxe**:

```
g = gradient(F, x)
```

- **Entradas**:
    - F: Vetor ou matriz cujos gradientes serão calculados.
    - x: Espaçamento entre os pontos.
- **Saída**:
    - g: Gradiente da função.

**Exemplo**:

```
x = linspace(0, 10, 100);
y = sin(x);
g = gradient(y, x);
```

## 8. sum

**Descrição:** Calcula a soma dos elementos de um vetor ou matriz.

**Sintaxe:**

```
s = sum(A)
```

- **Entradas:**
  - A: Vetor ou matriz cujos elementos serão somados.
- **Saída:**
  - s: Soma dos elementos.

**Exemplo:**

```
A = [1, 2, 3, 4];
s = sum(A); % Retorna 10
```

## 9. std

**Descrição:** Calcula o desvio padrão dos elementos de um vetor ou matriz.

**Sintaxe:**

```
s = std(A)
```

- **Entradas:**
  - A: Vetor ou matriz cujos desvios padrão serão calculados.
- **Saída:**
  - s: Desvio padrão dos elementos de A.

**Exemplo:**

```
A = [1, 2, 3, 4];
s = std(A); % Calcula o desvio padrão
```

## 10. errorbar

**Descrição:** Cria um gráfico com barras de erro.

**Sintaxe:**

```
errorbar(x, y, err)
```

- **Entradas:**
  - x: Vetor de valores da variável independente.
  - y: Vetor de valores da variável dependente.
  - err: Barras de erro para cada ponto de y.
- **Saída:**
  - Gráfico com barras de erro.

**Exemplo:**

```
x = 1:5;
y = [2, 3, 5, 4, 6];
err = [0.5, 0.3, 0.4, 0.2, 0.6];
```

```
errorbar(x, y, err);
```

## 11. rand

**Descrição:** Gera números aleatórios com distribuição uniforme no intervalo

[0,1].

**Sintaxe:**

```
r = rand
r = rand(m, n)
```

- **Entradas:**
  - m, n: Dimensões do vetor ou matriz de números aleatórios gerados.
- **Saída:**
  - r: Número ou matriz com valores aleatórios.

**Exemplo:**

```
r = rand(1, 10); % Gera um vetor com 10 números aleatórios
```

## 12. randn

**Descrição:** Gera números aleatórios com distribuição normal (média 0, desvio padrão 1).

**Sintaxe:**

```
r = randn
r = randn(m, n)
```

- **Entradas:**
  - m, n: Dimensões do vetor ou matriz de números aleatórios gerados.
- **Saída:**
  - r: Número ou matriz com valores aleatórios.

**Exemplo:**

```
r = randn(1, 10); % Gera um vetor com 10 números normalmente distribuídos
```