Rotinas Matlab

FSolve

```
opts = optimoptions(@fsolve, 'Algorithm', 'levenberg-marquardt');
opt = optimset("tolx", 10e-10, "tolfun", "10e-20", "display", "iter");
[xsol, fsol, exitflag, output] = fsolve(@func, x0, opts);

% função sistema
function [f] = func(x)
        f(1) = x(1) + 2;
        f(2) = x(2) + 4;
end

syms x a;
func1 = 0.125*(x+2)^3 - 0.25*(x+2)^2 + x + 3;
func2 = -0.375*x^3 + a*x^2 + 1.5*x + 3;

mt_func1 = matlabFunction(func1);
mt_func2 = matlabFunction(func2);

a_zero_1 = fsolve(@(a) mt_func1(0) - mt_func2(0, a), 0);

mt_func1_diff = matlabFunction(diff(mt_func1, x));
mt_func2_diff = matlabFunction(diff(mt_func2, x));

a_zero_2 = fsolve(@(a) mt_func1_diff(0) - mt_func2_diff(0, a), 0);
```

Spline

```
% spline básica
x = [1 (2) 3 5 7 9];
y = [1 3 2 12 21 6];

sol = spline(x, y); ou num ponto spline(x,y,5);
disp(sol.coefs);
% s23(x) = c1(x-2)^3 + c2(x-2)^2 + c3(x-2) + c4

% spline forçando derivadas
sol2 = spline(x, [k y k']);

% spline completa
% - calculo das derivadas
% diferenças divididas, ou a partir da expressao da função,
% caso seja dada

d0 = (y(1)-y(2))/(x(1)-x(2));
dn = (y(5)-y(6))/(x(5)-x(6));

sol3 = spline([1 3 5 9], [d0 1 2 12 6 dn], 1.5); % exemplo
```

Quad e Trapz

```
format long
x = [...];
y = [...];

integral = trapz(x, y);
```

```
% or
f = @(x) exp(-2.*x) - sin((x+7)./(x+1));
[q, nfc] = quad(f, 0, 8, 1.0e-15);
```

Polyfit

```
a = [...];
b = [...];

[p, s] = polyfit(a, b, 3); % grau 3
new_a = 1:0.1:11;
new_b = polyval(p, new_a); % ou para apenas um ponto polyval(p, 8);
plot(new_a, new_b, "red");

q = polyint(p); % returns the integral of the polynomial represented by the coefficients in p using a
constant of integration 0.
integral = diff(polyval(q, [0 5.4]));

% M(x) = c1 ln(x) + c2 sen(x) + c3 x^2

phi1 = @(x) log(x);
phi2 = @(x) sin(x);
phi3 = @(x) x.^2;

A = [sum(phi1(x).*phi1(x)) sum(phi1(x).*phi2(x)) sum(phi1(x).*phi3(x));
        sum(phi2(x).*phi1(x)) sum(phi2(x).*phi2(x)) sum(phi2(x).*phi3(x));
        sum(phi3(x).*phi1(x)) sum(phi3(x).*phi3(x)) sum(phi3(x).*phi3(x))];

B = [sum(b.*phi1(x)); sum(b.*phi2(x)); sum(b.*phi3(x))];
C = A\b;
M = @(x) c(1)*phi1(x) + c(2)*phi2(x) + c(3)*phi3(x);

res = b - M(a);
S = sum((res).^2); % somatório do quadrado dos resíduos

x = a;
fun = @(c, x) c(1)*log(x) + c(2)*sin(x) + c(3)*x.^2;
c0 = [1, 1, 1];
c_approx = lsqcurvefit(fun, c0, x, b); % aproximar polinomio a um modelo
```

Fminsearch

```
op=optimset("Display", "iter", "TolX", 10e-20, "MaxFunEvals", 10000, "MaxIter", 10000);
[x,f,exitflag,output]=fminsearch(@func,x,op);
% inline: fminsearch(@(t) abs(func(t, 0) - 8), initial_t, op);

function f = func(x)
        f = max(abs(x(1)), abs(x(2)-1));
end
```

Fminunc

```
[xval,fval,exitflag,output] = fminunc(@func, x, []);
fval = - fval; % ao utilizar o inverso da função para encontrar o mínimo,
% depois é necessário utilizar o simétrico do valor y

function f = func(x)
        a = [...];
        b = [...];
        f = -a(1)*... - a(2)*... + x(2); % simetrica à dada
```

```
end

---

i = 1:2:n; % números ímpares
x(i) = 2;

i = 2:2:n; % números pares
x(i) = 1;

op = optimset('display', 'iter', 'MaxFunEvals', 2000, 'hessupdate', 'dfp');
% por defeito é bfgs (agr atualiza a matriz hessiana com dfp)

[x,f,exitflag,output] = fminunc(@func2, x, op);

function [ f ] = func2(x)
        i = 1:2:n-1;
        y(i) = x(i) * ...;

        i = 2:2:n-1;
        y(i) = x(i) * ...;

        i = n;
        y(i) = x(i);

        i = 1:n;
        f = ...;
end

----

% fmincon
[x_opt, fval_opt] = fmincon(@func, x0, [], [], [], [], lb, ub, @constraints, options);
function [c, ceq] = constraints(x)
        c = x(1) - 2;
        ceq = x(3)^2 + x(4)^2 - 2;
end
```

Secante script

```
function [] = secant(func, x0, x1, n)
        x = 0;
        for 1=1:n
                x = x1 - ((func(x1)*(x0-x1))/func(x0)-func(x1));
                x0 = x1;
                x1 = x;
        end
end
```

#math