Contents


```
function info_matrix = line_search2(method,f,x_initial,param)
```

LINE_SEARCH2 Introduction

```
         Displays the first 10 and last points found for the
         minimization process. Returns all points found.

method: The method of optimization used, given as a string, with
        the options of "steepd" for steepest descent method, and
        "newton" for Newton's method.

f:      The function to be evaluated, using symbolic variables

x_0:    The initial point to begin the minimization search, given
        as a vertical vector.

param:  The basic parameters, given in a vector with the format:
        [alpha_0, c1, tolerance]
```

Establishing Parameters

```
    format long

    a_0 = param(1);
    c1 = param(2);
    tolerance = param(3);

    x_k = x_initial;
    info_matrix = zeros(1,6);

    e1 = 1e-6;
    e2 = 1e-4;
```

Setting Up Functions

```
    x                   = sym('x', [2,1]);
    sfun_f(x)           = f;
    sfun_f_gradient(x)  = gradient(sfun_f);
    sfun_f_hessian(x)   = hessian(sfun_f);

    f_eval              = matlabFunction(sfun_f, 'Vars', {x});
```

```
f_grad              = matlabFunction(sfun_f_gradient, 'Vars', {x});
f_hess              = matlabFunction(sfun_f_hessian, 'Vars', {x});
```

Line Search Algorithm

```
i    = 1;

while abs(f_eval(x_k)) > tolerance && norm(f_grad(x_k)) > tolerance

    % FIND DESCENT DIRECTION
    if method == "steepest descent"
        p_k = -f_grad(x_k) / norm(f_grad(x_k));
    elseif method == "Newton"
        p_k = -inv(f_hess(x_k))' * f_grad(x_k);
    end

    % INITIAL STEP LENGTH STRATEGY #1
    if i>1 && method == "steepest descent"
        a_0 = a_2 * f_grad(x_k_old)' * p_k_old / ...
              (f_grad(x_k)' * p_k);
    end

    q_eval = @(a) f_eval(x_k + a * p_k);
    q_grad = @(a) f_grad(x_k + a * p_k);

    while 1

        % TEST WITH INITIAL GUESS a_0
        if armijo(q_eval, q_grad, p_k, a_0, c1)
            a_2 = a_0;
            break
        else
            a_1 = -(1/2) * p_k' * q_grad(0) * a_0^2 / ...
                  (q_eval(a_0)-q_eval(0)-p_k'*q_grad(0)*a_0);
            a_2 = a_1;
        end

        % REPEAT INTERPOLATION UNTIL ALPHA IS FOUND
        if armijo(q_eval, q_grad, p_k, a_2, c1)
            break
        else
            tmp = a_2;
            a_2 = zoom(q_eval, q_grad, p_k, a_0, a_1);
            a_0 = a_1;
            a_1 = tmp;

            % SAFEGUARD
            if abs(a_2 - a_1) < e1 || abs(a_2) < e2
                a_2 = a_1 / 2;
            end
        end
    end


    info_matrix(i,:) = [x_k', f_eval(x_k), p_k', a_2];
```

```
        x_k_old = x_k;
        p_k_old = p_k;

        x_k = x_k + a_2 * p_k;
        i = i + 1;

    end
```

Printing Results

```
    disp(i-1 + " iterations using " + method + " method,")
    disp("starting at point (" + x_initial(1) + ", " + x_initial(2) + "):")

    headers = {'x_1','x_2','f(x_0)','p_k1','p_k2','alpha'};
    s = '...';  space = {s, s, s, s, s, s};
    maxi = min(i-1,10);
    firsts = num2cell(info_matrix(1:maxi,:));
    last = num2cell(info_matrix(end,:));
    disp([headers; firsts; space; last]);
    fprintf('\n')

end
```

Published with MATLAB® R2018b