Contents


```
function info_matrix = line_search3(method,f,x_initial,param)
```

LINE_SEARCH3 Introduction

```
        Returns all points found in "info_matrix" for current
        selected method.

method: The method of optimization used, given as a string, with
        the options of "steepest descent" for steepest descent
        method, "Newton" for Newton's method, and "trust region"
        for the trust region 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);
    a_2 = 1;
    c1 = param(2);
    tolerance = param(3);
    delta = 1;

    x_k = x_initial;
    p = 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);
```

```
    mfun_m_k(x)         = sfun_f + p'*sfun_f_gradient...
                          + 0.5 * p' * sfun_f_hessian * p;
    mfun_m_g(x)         = gradient(mfun_m_k);

    f_eval              = matlabFunction(sfun_f, 'Vars', {x});
    f_grad              = matlabFunction(sfun_f_gradient, 'Vars', {x});
    f_hess              = matlabFunction(sfun_f_hessian, 'Vars', {x});
    m_k                 = matlabFunction(mfun_m_k, 'Vars', {x});
    m_grad              = matlabFunction(mfun_m_g, 'Vars', {x});
```

Line Search Algorithm

```
    i = 1;
    info_matrix(i,:) = [x_k', f_eval(x_k), 0, 0, a_0];

    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);

        elseif method == "trust region"
            % FITTING RADIUS TO TRUST REGION %

            if norm(f_hess(x_k) \ f_grad(x_k)) <= delta
                p_k = -f_hess(x_k) \ f_grad(x_k);
            else
                temp = solve(m_grad(x) == [0;0],x);
                p_k = [temp.x1 ; temp.x2];
            end

            %x_k = x_k + p_k;

            % THIS LOOP DOES NOT FUNCTION CORRECTLY
%             while norm(f_hess(x_k) \ f_grad(x_k)) > delta
%
%                 [p_k, rho] = rad_fit(f, m_k, x_k, delta);
%
%                 if rho <= 1/4
%                     delta = 1/4 * delta;
%                 elseif rho >= 3/4
%                     delta = min(2*delta, 2);
%                 end
%             end

        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 method ~= "trust region"

            % 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

        x_k = x_k + a_2 * p_k;

        x_k_old = x_k;
        p_k_old = p_k;

        i = i + 1;
        info_matrix(i,:) = [x_k', f_eval(x_k), p_k', a_2];

    end

end
```