

```

1 function [x_opt, fval_opt, x_iter, f_iter, alpha] = BfgsDescent(x0, f, G, H0, ✓
maxiter, grad_tol)
2 % Inputs
3 % x0: initial point
4 % f: function handle
5 % G: function handle for the gradient of f
6 % H0: initial estimate of inverse hessian
7 % maxiter: maximum number of iterations (default 10000)
8 % grad_tol: minimum norm value of gradient (default 1e-4)
9 %
10 % Returns
11 % x_opt: x^*
12 % fval_opt: f(x^*)
13 % x_iter: all iterates k -- column k contains x_k
14 % f_iter: a vector of all function values f(x_k)
15 % alpha: a vector of all step lengths alpha_k
16
17 % Termination criteria
18 maxiter_default = 10000;
19 grad_tol_default = 1e-4;
20
21 if nargin == 4
22     maxiter = maxiter_default;
23     grad_tol = grad_tol_default;
24 elseif nargin == 5
25     grad_tol = grad_tol_default;
26 end
27
28 nx = size(x0,1); % Number of variables
29
30 % Declare some variables
31 x = NaN(nx,maxiter);
32 p = NaN(nx,maxiter);
33 grad = NaN(nx,maxiter);
34 Happrox = NaN(nx,nx,maxiter); % approximation of inverse hessian using bfgs
35 alpha = NaN(1,maxiter);
36 fval = NaN(1,maxiter);
37
38
39 % Do some calculations before the while loop. I.e., do the first iteration:
40 k = 1; % iteration number
41 x(:,k) = x0;
42 % look at the different functions below and finish them before continuing.
43 fval(k) = f(x0);
44 grad(:,k) = G(x0);
45 Happrox(:, :, k) = H0;
46 p(:,k) = bfgs(grad(:,k), Happrox(:, :, k));
47
48 alpha_0 = 1;
49
50 alpha(k) = linesearch(f, x(:,k), p(:,k), fval(k), grad(:,k), alpha_0);
51 x(:,k+1) = x(:,k) + alpha(k) * p(:,k);
52 grad(:,k+1) = G(x(:,k+1));
53 Happrox(:, :, k+1) = bfgs_step(Happrox(:, :, k), x(:,k), x(:,k+1), grad(:,k), grad(:, ✓
k+1));
54 k = k + 1;
55
56 while k < maxiter && norm(grad(:,k),2) >= grad_tol
57     fval(k) = f(x(:,k));
58     p(:,k) = bfgs(grad(:,k), Happrox(:, :, k));

```

```

59     alpha_0 = 1;
60     alpha(k) = linesearch(f, x(:,k), p(:,k), fval(k), grad(:,k), alpha_0); % ↙
Determine alpha using Alg. 3.1
61     x(:,k+1) = x(:,k) + alpha(k) * p(:,k);
62
63     grad(:,k+1) = G(x(:,k+1));
64     Happrox(:, :, k+1) = bfgs_step(Happrox(:, :, k), x(:,k), x(:,k+1), grad(:,k), grad ↙
(:,k+1));
65     k = k + 1;
66 end
67 fval(k) = f(x(:,k)); % Final function value
68
69 % Delete unused space
70 x = x(:,1:k);
71 p = p(:,1:k);
72 grad = grad(:,1:k);
73 Happrox = Happrox(:, :, 1:k);
74 alpha = alpha(1:k);
75 fval = fval(1:k);
76
77 % Return values
78 x_opt = x(:,end);
79 fval_opt = f(x_opt);
80 x_iter = x;
81 f_iter = fval;
82
83 end
84
85 % Function returning the steepest-descent direction based on the gradient
86 % of f
87 function p = bfgs(grad, hk)
88     p = -hk * grad;
89 end
90
91 function hk1 = bfgs_step(hk, xk, xk1, gradk, gradk1)
92     I = eye(size(hk));
93     sk = xk1 - xk;
94     yk = gradk1 - gradk;
95     rhok = (yk.') * sk;
96
97     hk1 = (I - sk * (yk.') / rhok) * hk * (I - yk * (sk.') / rhok) + sk * (sk.') / ↙
rhok;
98 end
99
100 % Function implementing Algorithm 3.1, page 37 in N&W
101 function alpha_k = linesearch(f, xk, pk, fk, gradk, alpha_0)
102     alpha = alpha_0;
103     rho = 0.9;
104     c1 = 0.5; % a constant for sufficient decrease
105     while f(xk + alpha*pk) > fk + c1*alpha*(gradk.')*pk
106         alpha = rho * alpha;
107     end
108     alpha_k = alpha;
109 end
110

```