

## Code Listings

*Listing 1: a7.m.*

```
1  epsilon = 1e-6;
2
3  disp('1. Rosenbrock function');
4  disp('2. Himmelblau function');
5  function_index = input('Choose the function to be minimized: ');
6  switch function_index
7      case 1
8          disp('=====');
9          disp('      Rosenbrock function      ');
10         disp('=====');
11         f = f_rosenbrock();
12         x0 = [-1.2; 1];
13         x_range = [-1.5, 1.5, -0.5, 2];
14     case 2
15         disp('=====');
16         disp('      Himmelblau function      ');
17         disp('=====');
18         f = f_himmelblau();
19         x0 = [4; 4];
20         x_range = [-6, 6, -6, 6];
21     otherwise
22         disp('Invalid function. ');
23         return;
24 end
25
26 disp('1. Gradient method');
27 disp('2. Globalized Newton's method');
28 disp('3. Globalized BFGS method');
29 disp('4. Globalized inexact Newton's method');
30 disp('5. All methods (for comparison)');
31 method_index = input('Choose the minimization method: ');
32 switch method_index
33     case 1
34         disp('Gradient method');
35         [x, k] = gradient_method(f, x0, epsilon);
36         display_solution(f, x, k);
37     case 2
38         disp('Globalized Newton's method');
39         [x, k] = newton_global(f, x0, epsilon);
40         display_solution(f, x, k);
41     case 3
42         disp('Globalized BFGS method');
43         [x, k] = bfgs_global(f, x0, epsilon);
44         display_solution(f, x, k);
45     case 4
46         disp('Globalized inexact Newton's method');
47         [x, k] = newton_inexact(f, x0, epsilon);
48         display_solution(f, x, k);
49     case 5
50         disp('-----');
51         disp('Gradient method');
52         [x, k] = gradient_method(f, x0, epsilon);
53         display_solution(f, x, k);
54         disp('-----');
```

```

55     disp('Globalized Newton''s method');
56     [x, k] = newton_global(f, x0, epsilon);
57     display_solution(f, x, k);
58     disp('-----')
59     disp('Globalized BFGS method');
60     [x, k] = bfgs_global(f, x0, epsilon);
61     display_solution(f, x, k);
62     disp('-----')
63     disp('Globalized inexact Newton''s method');
64     [x, k] = newton_inexact(f, x0, epsilon);
65     display_solution(f, x, k);
66     otherwise
67         disp('Invalid minimization method.');
```

```

68 end
```

*Listing 2: f\_rosenbrock.m.*

```

1 function f_rosenbrock = f_rosenbrock()
2 f_rosenbrock = @(x) 100 * (x(2) - x(1)^2)^2 + (1 - x(1))^2;
3 end
```

*Listing 3: f\_himmelblau.m.*

```

1 function f_himmelblau = f_himmelblau()
2 f_himmelblau = @(x) (x(1)^2 + x(2) - 11)^2 + (x(1) + x(2)^2 - 7)^2;
3 end
```

*Listing 4: gradient\_method.m.*

```

1 function [x, k] = gradient_method(f, x0, epsilon)
2 beta = 0.5;
3 sigma = 1e-4;
4
5 k = 0;
6 x = x0;
7 grad_val = gradest(f, x).';
8 n = norm(grad_val);
9 while n > epsilon
10     d = -grad_val;
11
12     % Armijo update
13     t = armijo(f, x, sigma, grad_val, d, beta);
14
15     x = x + t*d;
16     k = k + 1;
17     grad_val = gradest(f, x).';
18     n = norm(grad_val);
19 end
20 end
```

Listing 5: *newton\_global.m*.

```
1 function [x, k] = newton_global(f, x0, epsilon)
2 rho = 1e-8;
3 p = 2.1;
4 beta = 0.5;
5 sigma = 1e-4;
6 k_max = 200;
7
8 k = 0;
9 x = x0;
10 grad_val = gradest(f, x).';
11 n = norm(grad_val);
12 while n > epsilon && k <= k_max
13     hess = hessian(f, x);
14
15     d = -grad_val;
16     if cond(inv(hess)) > 1e-12
17         d_soln = hess \ (-grad_val);
18         if grad_val' * d_soln <= - rho * (norm(d_soln) ^ p)
19             d = d_soln;
20         end
21     end
22
23     % Armijo update
24     t = armijo(f, x, sigma, grad_val, d, beta);
25
26     x = x + t*d;
27     k = k + 1;
28     grad_val = gradest(f, x).';
29     n = norm(grad_val);
30 end
31 end
```

Listing 6: *bfgs\_global.m*.

```
1 function [x, k] = bfgs_global(f, x0, epsilon)
2 sigma = 1e-4;
3 rho = 0.9;
4 H_0 = eye(size(x0, 1));
5 t_0 = 1;
6 gamma = 2;
7
8 k = 0;
9 x = x0;
10 H = H_0;
11 grad_val = gradest(f, x).';
12 n = norm(grad_val);
13 while n > epsilon
14     d = H \ (-grad_val);
15
16     phi = @(t) f(x + t * d);
17     phi_prime_0 = gradest(phi, 0);
18     psi = @(t) phi(t) - phi(0) - sigma * t * phi_prime_0;
19
20     t_i = t_0;
21
22     stop = false;
```

```

23     while ~stop && psi(t_i) < 0
24         if gradest(phi, t_i) >= rho * phi_prime_0
25             stop = true;
26         else
27             t_i = gamma * t_i;
28         end
29     end
30
31     a = 0;
32     b = t_i;
33     while ~stop
34         if psi(t_i) >= 0
35             b = t_i;
36         elseif gradest(phi, t_i) < rho * phi_prime_0
37             a = t_i;
38         else
39             stop = true;
40         end
41         t_i = a + (b - a) / 2;
42     end
43
44
45     x_old = x;
46     x = x + t_i*d;
47     grad_val_new = gradest(f, x).';
48     s = x - x_old;
49     y = grad_val_new - grad_val;
50     H = H + (y * y') / (y' * s) - (H * (s * s') * H) / (s' * H * s);
51     k = k + 1;
52
53     k = k + 1;
54     grad_val = grad_val_new;
55     n = norm(grad_val);
56 end
57 end

```

*Listing 7: newton\_inexact.m.*

```

1  function [x, k] = newton_inexact(f, x0, epsilon)
2  beta = 0.5;
3  sigma = 1e-4;
4  rho = 1e-8;
5  p = 2.1;
6  c1 = 1e-2;
7  c2 = 1;
8
9  k = 0;
10 x = x0;
11 grad_val = gradest(f, x).';
12 n = norm(grad_val);
13 while n > epsilon
14     eta = min([c1 / (k + 1), c2 * n]);
15
16     % CG method (to solve inexact Newton equation)
17     d = cg(hessian(f, x), -grad_val, -grad_val, eta * n);
18
19     if grad_val' * d > - rho * (norm(d)^p)
20         d = -grad_val;
21     end

```

```

22
23     % Armijo update
24     t = armijo(f, x, sigma, grad_val, d, beta);
25
26     x = x + t*d;
27     k = k + 1;
28     grad_val = gradest(f, x).';
29     n = norm(grad_val);
30 end
31 end

```

*Listing 8: armijo.m.*

```

1  function t = armijo(f, x_k, sigma, grad_val, d, beta)
2  l = 0;
3  t = 1;
4  f_val = f(x_k);
5  rhs = sigma * grad_val.' * d;
6  while f(x_k + t*d) > f_val + t*rhs
7      l = l + 1;
8      t = beta ^ l;
9  end
10 end

```

*Listing 9: cg.m.*

```

1  function x = cg(A, x0, b, epsilon)
2  x = x0;
3  g = A * x - b;
4  d_cg = -g;
5  while norm(g) > epsilon
6      g_norm_2 = norm(g)^2;
7      t_cg = g_norm_2 / (d_cg' * A * d_cg);
8
9      x = x + t_cg * d_cg;
10     g = g + t_cg * A * d_cg;
11     beta_cg = norm(g)^2 / g_norm_2;
12     d_cg = -g + beta_cg * d_cg;
13 end
14 end

```

*Listing 10: display\_solution.m.*

```

1  function display_solution(f, x, k)
2  disp('Solution x:');
3  disp(x);
4  disp('Solution f(x):');
5  disp(f(x));
6  disp('Number of iterations:');
7  disp(k);

```