Code Listings

Listing 1: a7.m.

```
epsilon = 1e-6;

disp('1. Rosenbrock function');
disp('2. Himmelblau function');
function_index = input('Choose the function to be minimized: ');
switch function_index
    case 1
        disp('==================================');
        disp('       Rosenbrock function        ');
        disp('==================================');
        f = f_rosenbrock();
        x0 = [-1.2; 1];
        x_range = [-1.5, 1.5, -0.5, 2];
    case 2
        disp('==================================');
        disp('       Himmelblau function        ');
        disp('==================================');
        f = f_himmelblau();
        x0 = [4; 4];
        x_range = [-6, 6, -6, 6];
    otherwise
        disp('Invalid function.');
        return;
end

disp('1. Gradient method');
disp('2. Globalized Newton''s method');
disp('3. Globalized BFGS method');
disp('4. Globalized inexact Newton''s method');
disp('5. All methods (for comparison)');
method_index = input('Choose the minimization method: ');
switch method_index
    case 1
        disp('Gradient method');
        [x, k] = gradient_method(f, x0, epsilon);
        display_solution(f, x, k);
    case 2
        disp('Globalized Newton''s method');
        [x, k] = newton_global(f, x0, epsilon);
        display_solution(f, x, k);
    case 3
        disp('Globalized BFGS method');
        [x, k] = bfgs_global(f, x0, epsilon);
        display_solution(f, x, k);
    case 4
        disp('Globalized inexact Newton''s method');
        [x, k] = newton_inexact(f, x0, epsilon);
        display_solution(f, x, k);
    case 5
        disp('----------------------------------')
        disp('Gradient method');
        [x, k] = gradient_method(f, x0, epsilon);
        display_solution(f, x, k);
        disp('----------------------------------')
```

```
        disp('Globalized Newton''s method');
        [x, k] = newton_global(f, x0, epsilon);
        display_solution(f, x, k);
        disp('----------------------------------')
        disp('Globalized BFGS method');
        [x, k] = bfgs_global(f, x0, epsilon);
        display_solution(f, x, k);
        disp('----------------------------------')
        disp('Globalized inexact Newton''s method');
        [x, k] = newton_inexact(f, x0, epsilon);
        display_solution(f, x, k);
    otherwise
        disp('Invalid minimization method.');
end
```

Listing 2: f_rosenbrock.m.

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

Listing 3: f_himmelblau.m.

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

Listing 4: gradient_method.m.

```
function [x, k] = gradient_method(f, x0, epsilon)
beta = 0.5;
sigma = 1e-4;

k = 0;
x = x0;
grad_val = gradest(f, x).';
n = norm(grad_val);
while n > epsilon
    d = -grad_val;

    % Armijo update
    t = armijo(f, x, sigma, grad_val, d, beta);

    x = x + t*d;
    k = k + 1;
    grad_val = gradest(f, x).';
    n = norm(grad_val);
end
end
```

Listing 5: newton_global.m.

```
function [x, k] = newton_global(f, x0, epsilon)
rho = 1e-8;
p = 2.1;
beta = 0.5;
sigma = 1e-4;
k_max = 200;

k = 0;
x = x0;
grad_val = gradest(f, x).';
n = norm(grad_val);
while n > epsilon && k <= k_max
    hess = hessian(f, x);

    d = -grad_val;
    if cond(inv(hess)) > 1e-12
        d_soln = hess \ (-grad_val);
        if grad_val' * d_soln <= - rho * (norm(d_soln) ^ p)
            d = d_soln;
        end
    end

    % Armijo update
    t = armijo(f, x, sigma, grad_val, d, beta);

    x = x + t*d;
    k = k + 1;
    grad_val = gradest(f, x).';
    n = norm(grad_val);
end
end
```

Listing 6: bfgs_global.m.

```
function [x, k] = bfgs_global(f, x0, epsilon)
sigma = 1e-4;
rho = 0.9;
H_0 = eye(size(x0, 1));
t_0 = 1;
gamma = 2;

k = 0;
x = x0;
H = H_0;
grad_val = gradest(f, x).';
n = norm(grad_val);
while n > epsilon
    d = H \ (-grad_val);

    phi = @(t) f(x + t * d);
    phi_prime_0 = gradest(phi, 0);
    psi = @(t) phi(t) - phi(0) - sigma * t * phi_prime_0;

    t_i = t_0;

    stop = false;
```

```
    while ~stop && psi(t_i) < 0
        if gradest(phi, t_i) >= rho * phi_prime_0
            stop = true;
        else
            t_i = gamma * t_i;
        end
    end

    a = 0;
    b = t_i;
    while ~stop
        if psi(t_i) >= 0
            b = t_i;
        elseif gradest(phi, t_i) < rho * phi_prime_0
            a = t_i;
        else
            stop = true;
        end
        t_i = a + (b - a) / 2;
    end


    x_old = x;
    x = x + t_i*d;
    grad_val_new = gradest(f, x).';
    s = x - x_old;
    y = grad_val_new - grad_val;
    H = H + (y * y') / (y' * s) - (H * (s * s') * H) / (s' * H * s);
    k = k + 1;

    k = k + 1;
    grad_val = grad_val_new;
    n = norm(grad_val);
end
end
```

Listing 7: newton_inexact.m.

```
function [x, k] = newton_inexact(f, x0, epsilon)
beta = 0.5;
sigma = 1e-4;
rho = 1e-8;
p = 2.1;
c1 = 1e-2;
c2 = 1;

k = 0;
x = x0;
grad_val = gradest(f, x).';
n = norm(grad_val);
while n > epsilon
    eta = min([c1 / (k + 1), c2 * n]);

    % CG method (to solve inexact Newton equation)
    d = cg(hessian(f, x), -grad_val, -grad_val, eta * n);

    if grad_val' * d > - rho * (norm(d)^p)
        d = -grad_val;
    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);
```