Assignment 9 OptReg

By Alexander Rambech

Task 1

a)

We define the MATLAB function solving the function in the bottom of this file. The condition in algorithm 3.1 is sometimes called the *Armijo condition* and ensures a sufficient decrease in the objective function.

```
% Minimizing the Rosenbrock function using line search

% Function
x = sym('x', [2 1]);
symbolic_f = 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
symbolic_grad = gradient(symbolic_f)
```

```
symbolic_grad =
```

$$\begin{pmatrix} 2\,x_1 - 400\,x_1\,\left(x_2 - {x_1}^2\right) - 2 \\ 200\,x_2 - 200\,{x_1}^2 \end{pmatrix}$$

```
symbolic_hessian = jacobian(symbolic_grad) % I think this is the way to find
```

```
symbolic_hessian =
```

$$\begin{pmatrix} 1200\,{x_1}^2 - 400\,x_2 + 2 & -400\,x_1 \\ -400\,x_1 & 200 \end{pmatrix}$$

```
                                        % the Hessian matrix.
% Initialising
x0_easy = [1.2; 1.2];
x0_difficult = [-1.2; 1];
a0 = 1;
num_iterations = 10000;

% Converting to MATLAB functions
f = matlabFunction(symbolic_f, "Vars", {x});
grad = matlabFunction(symbolic_grad, "Vars", {x});
J = matlabFunction(symbolic_hessian, "Vars", {x});

% Easy initial guess
sprintf("Alphas and number of iterations given the easy starting point")
```

```
ans =
"Alphas and number of iterations given the easy starting point"
```

```
easy_decent = Steepest(f, grad, x0_easy, a0, num_iterations);
```

```
ak = 1.0000e-03
ak = 1.0000e-03
ak = 1.0000e-03
ak = 1.0000e-03
```

```
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
```

```
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
```

```
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
```

```
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
```

```
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
```

```
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.0100
ak = 1.0000e-03
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1.0000e-03
ak = 0.1000
ak = 1.0000e-03
ak = 1
ak = 1.0000e-03
ak = 1.0000e-14
ans =
"Steepest decent solver converged on iteration: 345"
```

```
easy_direct_Newton = DirectNewton(f, grad, J, x0_easy, a0, num_iterations);
```

```
ak = 1
ak = 0.1000
ak = 0.1000
ak = 0.1000
ak = 0.1000
ak = 1
ak = 1
ak = 1
ak = 1
ak = 1
ak = 1
ak = 1
ak = 1
ans =
"Direct Newton solver converged on iteration: 13"
```

```
% Difficult initial guess
sprintf("Alphas and number of iterations given the difficult starting point")
```

```
ans =
"Alphas and number of iterations given the difficult starting point"
```

```
difficult_decent = Steepest(f, grad, x0_difficult, a0, num_iterations);
```

```
ak = 1.0000e-03
ans =
"Steepest decent solver converged on iteration: 1"
```

```
difficult_direct_Newton = DirectNewton(f, grad, J, x0_difficult, a0, num_iterations);
```

```
ak = 1
ans =
"Direct Newton solver converged on iteration: 1"
```

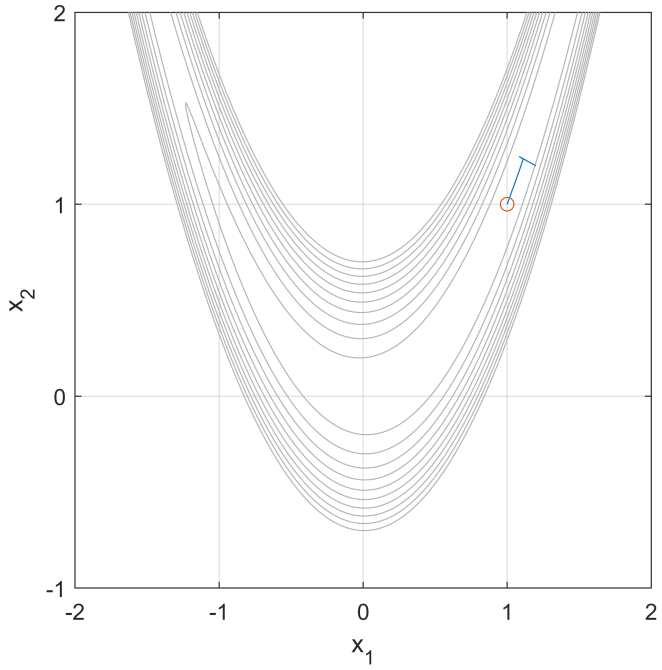
We see that both algorithms converge after some iterations, the Steepest Decent is a lot slower than the Direct Newton method. Using 345 iterations instead of only 13. However both struggle when met with the more difficult starting point. This could indicate that this is a local minima and that the initial step length α_0 as well as the constants ρ and c must be altered in order for these two functions to converge elsewhere.

b)

c)

It is common to use $\alpha = 1$ as initial value for the step length.

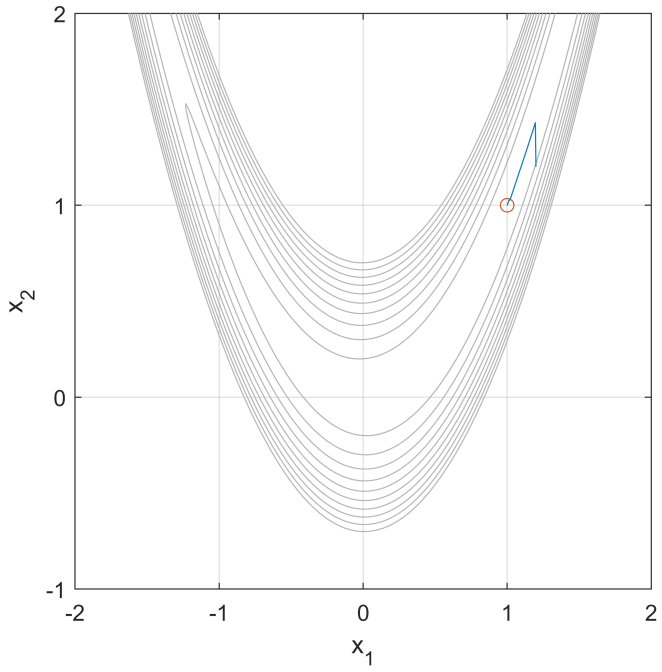
```
plot_iter_rosenbrock(easy_decent)
```




```
sprintf("Easy steepest decent")
```

```
ans =
"Easy steepest decent"
```

```
plot_iter_rosenbrock(easy_direct_Newton)
```




```
sprintf("Easy Newton")
```

```
ans =
"Easy Newton"
```

Task 2

Cholesky factorization is simply the decomposition of a positive-definite Hessian matrix into its lower triangular matix and it's conjugate transpose.

Task 3

a)

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{f(x+\epsilon e_1) - f(x)}{\epsilon} \\ \frac{f(x+\epsilon e_2) - f(x)}{\epsilon} \end{bmatrix} = \begin{bmatrix} 200(x_1 - x_2) + 2(x_1 - 1) + 101\epsilon \\ 200(x_2 - x_1) + 100\epsilon \end{bmatrix}$$

```
function x = Steepest(f, grad, x0, a0, iter)
    % Initialising
    rho = 0.1; % We choose to decrease alpha with a rate of
    c = 0.001; % We try with a very small c value
    x = zeros(length(x0), iter);
    x(:, 1) = x0;
    k = 1;

    while k < iter
        pk = -grad(x(:, k));
        a = a0;
```

```
        while gt(f(x(:, k) + a*pk), f(x(:, k)) + c*a*grad(x(:, k))'*pk)
            a = a*rho;
        end
        ak = a % With out semi-colon in order to print alpha_k
        x(:, k+1) = x(:, k) + ak*pk;
        k = k + 1;

        % In this numerical problem, we choose to break the loop when the
        % current value of x is larger or equal to the previous.
        % Assuming that the x is optimal or something is wrong.
        if x(:, k-1) <= x(:, k)
            sprintf("Steepest decent solver converged on iteration: %.0f", k-1)
            break;
        end
    end

    x(:, k-1:iter) = []; % We want to terminate with x containing valid values only
end
function x = DirectNewton(f, grad, J, x0, a0, iter)
    % Initialising
    rho = 0.1; % We choose to decrease alpha with a rate of
    c = 0.001; % We try with a very small c value
    x = zeros(length(x0), iter);
    x(:, 1) = x0;
    k = 1;
    while k < iter
        pk = -J(x(:, k))\grad(x(:, k));
        a = a0;
        while gt(f(x(:, k) + a*pk), f(x(:, k)) + c*a*grad(x(:, k))'*pk)
            a = a*rho;
        end
        ak = a % With out semi-colon in order to print alpha_k
        x(:, k+1) = x(:, k) + ak*pk;
        k = k + 1;

        % In this numerical problem, we choose to break the loop when the
        % current value of x is larger or equal to the previous.
        % Assuming that the x is optimal or something is wrong.
        if x(:, k-1) <= x(:, k)
            sprintf("Direct Newton solver converged on iteration: %.0f", k-1)
            break;
        end
    end

    x(:, k-1:iter) = []; % We want to terminate with x containing valid values only
end
function x = Quasi_Newton(f, J, x0, a0)

end
```