Linear least-squares problem:

```
X = [1, 274, 2450; ...
     1, 180, 3254; ...
     1, 375, 3802; ...
     1, 205, 2838; ...
     1, 86, 2347];
 Y = [162; ...
      120; ...
      223; ...
      131; ...
      67];
```

Solution using Cholesky algorithm

```
R = chol(X'*X);
z = X'*Y;
p = R'\z;
beta_hat_chol = R\p
```

```
beta_hat_chol = 3×1
    7.0325
    0.5044
    0.0070
```

Solution using QR factorization

```
[Q, R] = qr(X);
z = Q'*Y;
beta_hat_qr = R\z
```

```
beta_hat_qr = 3×1
    7.0325
    0.5044
    0.0070
```

Solution using SVD approach

```
[U, S, V] = svd(X);
b = U'*Y;
z = S\b;
beta_hat_svd = V*z
```

```
beta_hat_svd = 3×1
    7.0325
    0.5044
    0.0070
```

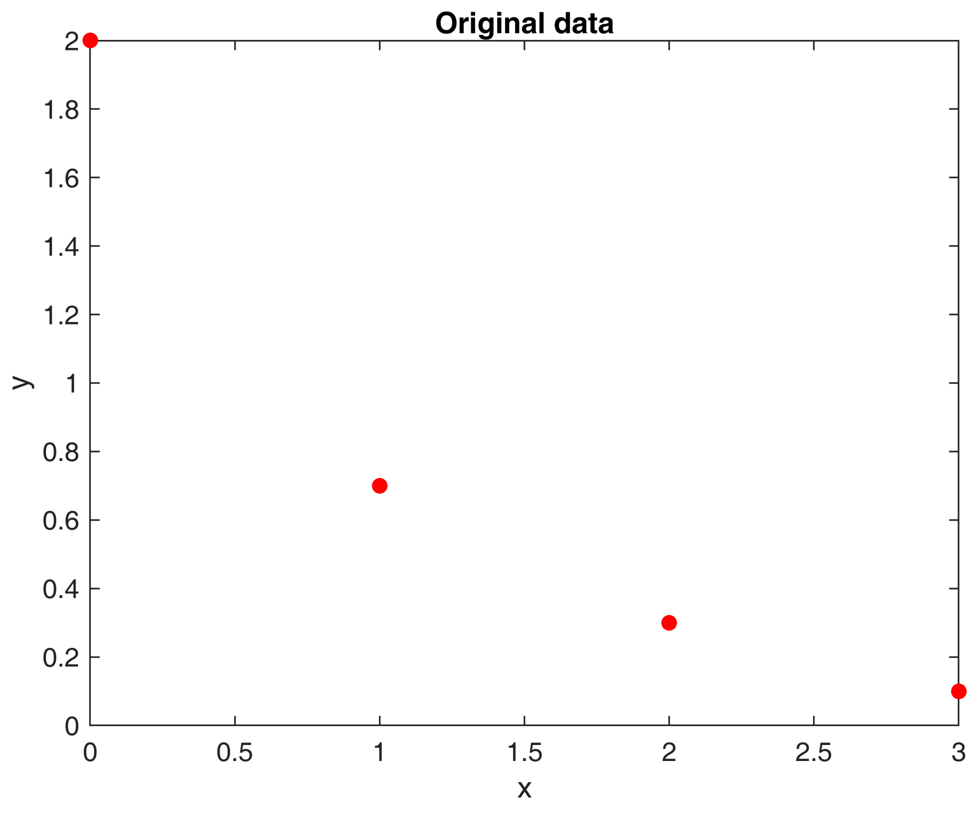
Nonlinear least-squares problem

$$y = \theta_1 e^{\theta_2 x}$$

Original data

```
x = [0, 1, 2, 3]';
```

```
y = [2, 0.7, 0.3, 0.1]';
figure()
plot(x,y, '.r', 'markersize', 20)
xlabel("x")
ylabel("y")
title("Original data")
```



Newton's method

Define residual function

```
syms theta_1 theta_2
residual = y - theta_1*exp(theta_2*x);
residual_func = matlabFunction(residual);
```

Define Jacobian

```
Jacobian = jacobian(residual, [theta_1, theta_2]);
jacobian_func = matlabFunction(Jacobian);
```

Gradient function

```
Gradient = Jacobian'* residual;
gradient_func = matlabFunction(Gradient);
```

Hessian

```
hessian_second_term = zeros(2,2);
```

```
for i = 1:length(x)
    hessian_second_term = hessian_second_term + residual(i)* hessian(residual(i),...
        [theta_1, theta_2]);
end
Hessian = Jacobian'*Jacobian + hessian_second_term;
hessian_func = matlabFunction(Hessian);
```

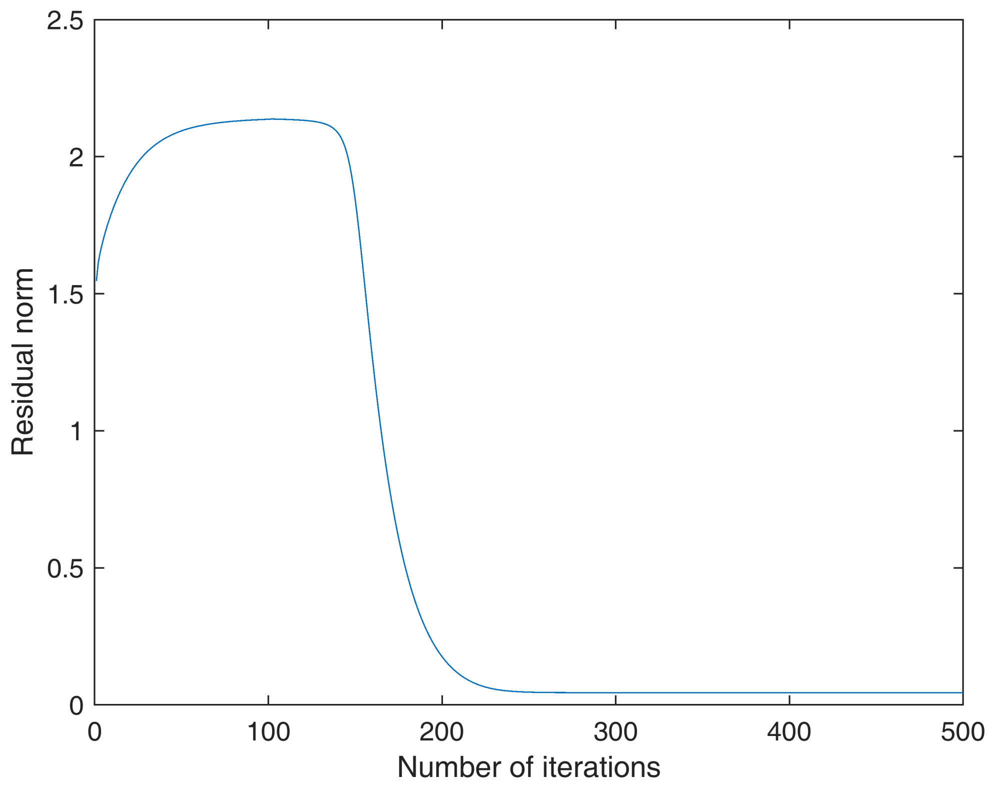
Applying Newton's algorithm:

The function can be found at the end of this document. Note that this function is custom made for this example. When the number of variable changes, we have to modify the function before applying it the problem.

```
initial_theta = [1;0];
[theta_newton, residual_norms, succ_approx_newton] = Newton_method(residual_func,...
    gradient_func, hessian_func, initial_theta, 0.05, 500, 1e-6);
theta_newton
```

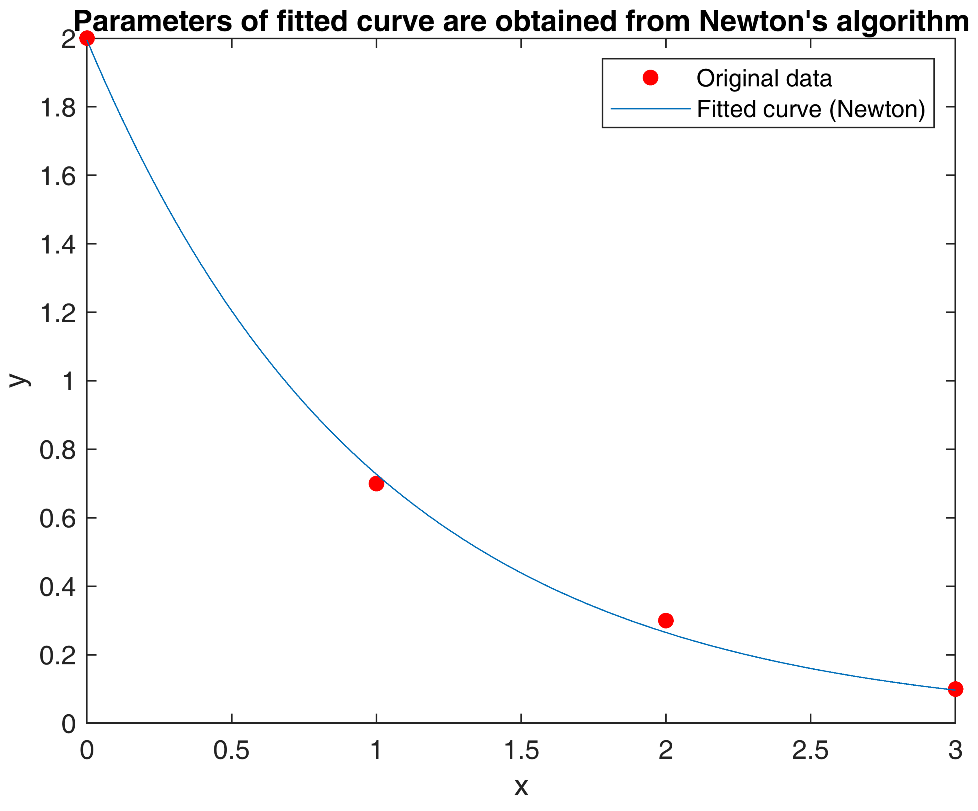
```
theta_newton = 2×1
    1.9950
   -1.0095
```

```
figure()
plot(residual_norms)
xlabel("Number of iterations")
ylabel("Residual norm")
```



```
figure()
plot(x,y, '.r', 'markersize', 20); hold on
x_dat = linspace(0, 3, 100);
```

```
y_newton = theta_newton(1) * exp(theta_newton(2)*x_dat);
plot(x_dat, y_newton); hold off
xlabel("x")
ylabel("y")
legend("Original data", "Fitted curve (Newton)")
title("Parameters of fitted curve are obtained from Newton's algorithm")
```



Gauss-Newton method

Our MATLAB codes for Gauss-Newton and Levenberg-Marqudrat algorithms are similar in line to the ones given at this link. The function implementing Gauss-Newton method can be found at the end of this document.

```
initial_theta = [1;0];
[theta_gauss_newton, norm_squares, succ_approx_gauss_newton] = ...
    gauss_newton_method(residual_func, jacobian_func, initial_theta, 10, 1e-6);
theta_gauss_newton
```

```
theta_gauss_newton = 2×1
    1.9950
   -1.0095
```

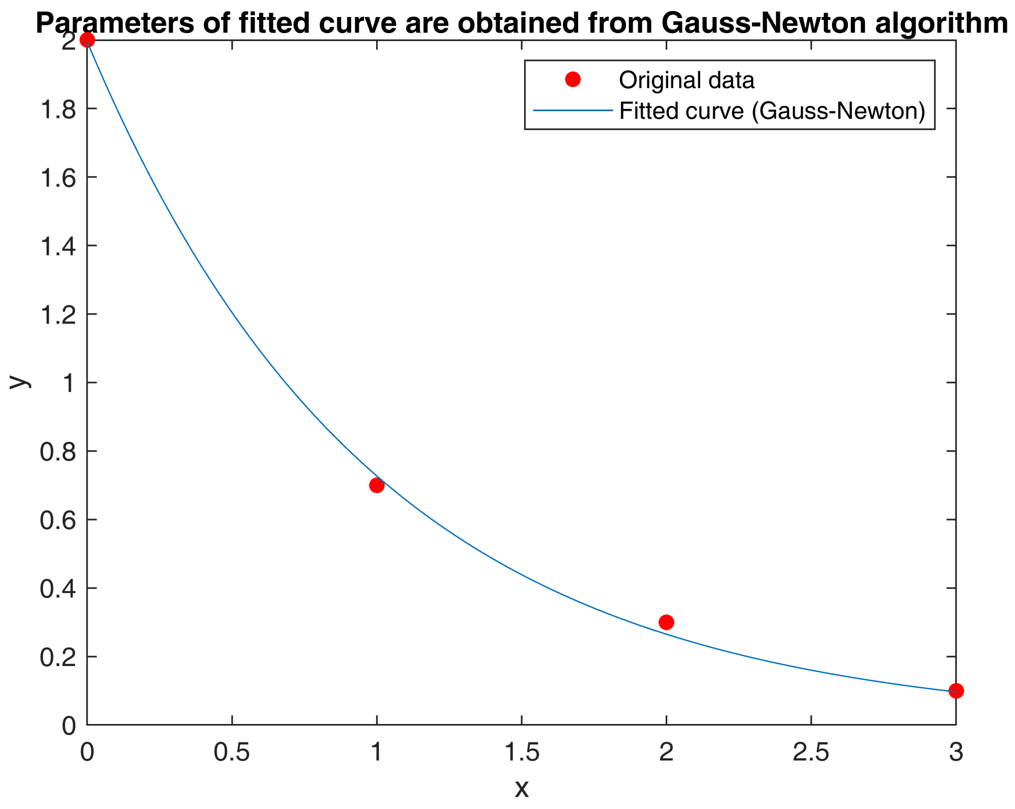
```
norm_squares
```

```
norm_squares = 10×1
    2.3900
    0.2126
    0.0073
    0.0020
    0.0020
    0.0020
    0.0020
    0.0020
    0.0020
    0.0020
```

```
succ_approx_gauss_newton
```

```
succ_approx_gauss_newton = 11×2
    1.0000         0
    1.6900   -0.6100
    1.9751   -0.9305
    1.9941   -1.0036
    1.9950   -1.0093
    1.9950   -1.0095
    1.9950   -1.0095
    1.9950   -1.0095
    1.9950   -1.0095
    1.9950   -1.0095
      ⋮
```

```
figure()
plot(x,y, '.r', 'markersize', 20); hold on
y_gauss_newton = theta_gauss_newton(1) * exp(theta_gauss_newton(2)*x_dat);
plot(x_dat, y_gauss_newton); hold off
xlabel("x")
ylabel("y")
legend("Original data", "Fitted curve (Gauss-Newton)")
title("Parameters of fitted curve are obtained from Gauss-Newton algorithm")
```



Levenberg-Marquardt method

```
initial_theta = [1;0];
```

```
[theta_leven, objectives, residuals, succ_approx_leven] = leven_marq(residual_func,...
    jacobian_func, initial_theta, 1, 100, 1e-6);
theta_leven
```

```
theta_leven = 2×1
    1.9950
   -1.0095
```

```
objectives
```

```
objectives = 14×1
    2.3900
    0.6220
    0.1606
    0.0364
    0.0082
    0.0029
    0.0021
    0.0020
    0.0020
    0.0020
      ⋮
```

```
residuals
```

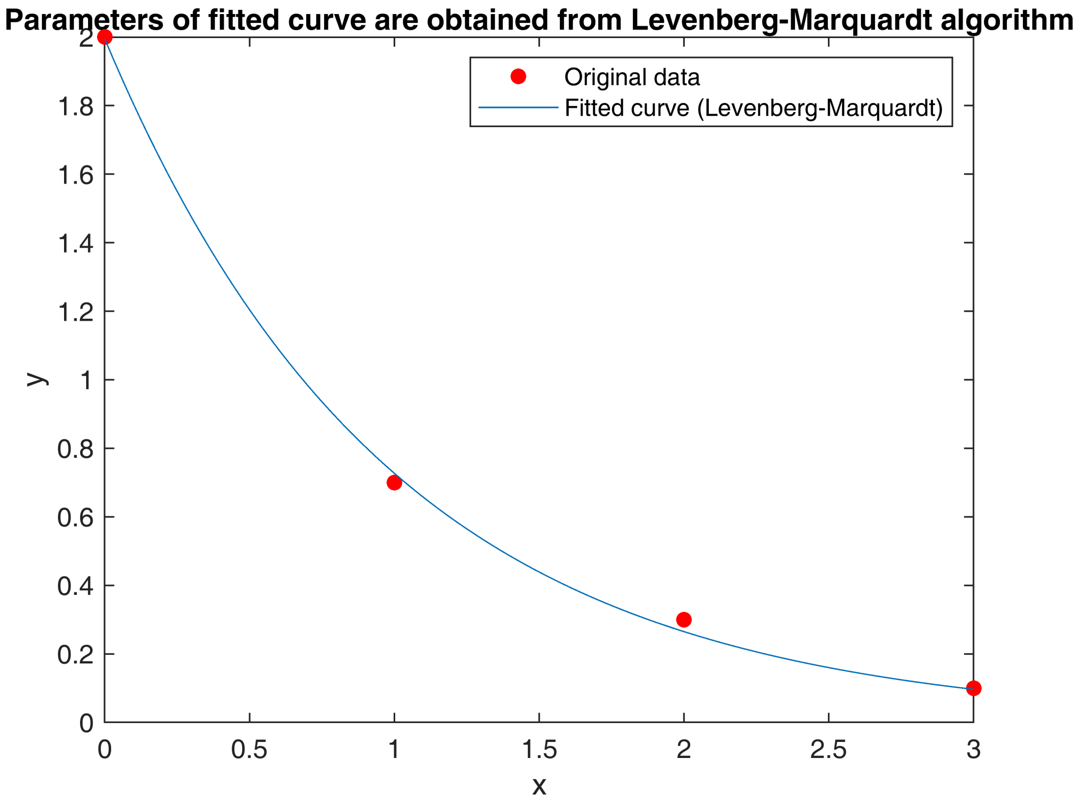
```
residuals = 14×1
    8.9822
    1.6781
    0.7007
    0.3187
    0.1314
    0.0491
    0.0165
    0.0049
    0.0013
    0.0003
      ⋮
```

```
succ_approx_leven
```

```
succ_approx_leven = 14×2
    1.0000         0
    1.3308   -0.4256
    1.6359   -0.7119
    1.8295   -0.8622
    1.9279   -0.9415
    1.9708   -0.9814
    1.9872   -0.9994
    1.9928   -1.0064
    1.9944   -1.0087
    1.9949   -1.0093
      ⋮
```

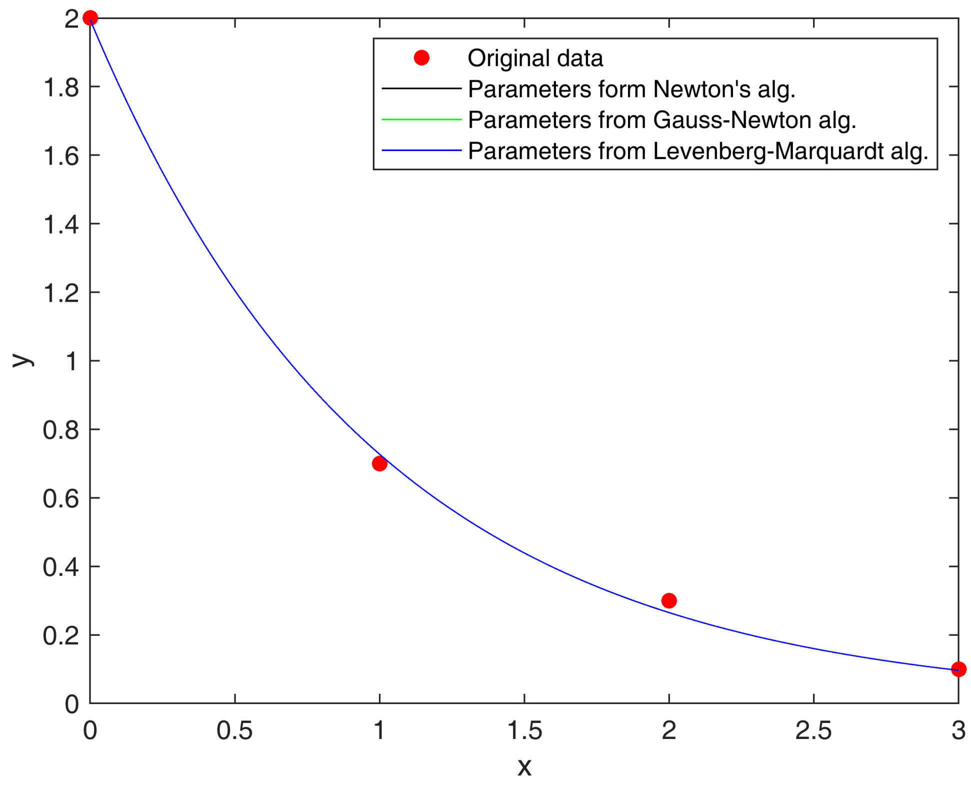
```
figure()
plot(x,y, '.r', 'markersize', 20); hold on
y_leven = theta_leven(1) * exp(theta_leven(2)*x_dat);
plot(x_dat, y_leven); hold off
xlabel("x")
ylabel("y")
```

```
legend("Original data", "Fitted curve (Levenberg-Marquardt)")
title("Parameters of fitted curve are obtained from Levenberg-Marquardt algorithm")
```



Finally, we will compare the curves obtained by using parameters of Gauss-Newton and Levenberg-Marquardt algorithms by plotting both the fits in a single figure along with the original data.

```
figure()
plot(x,y, '.r', 'markersize', 20); hold on
plot(x_dat, y_newton, 'k')
plot(x_dat, y_gauss_newton, 'g')
plot(x_dat, y_leven, 'b'); hold off
legend("Original data", "Parameters form Newton's alg.", ...
    "Parameters from Gauss-Newton alg.", ...
    "Parameters from Levenberg-Marquardt alg.")
xlabel("x"); ylabel("y")
```



As we can see, all the fits are indistinguishable. It can also be checked that parameters obtained from both the methods match till 6 digits after decimal point. We have run Gauss-Newton for a maximum of 10 iterations whereas Levenberg-Marquardt is run for 100 iterations.

```
function [theta, fnorms, succ_approx_newton] = Newton_method(f, grad, hess, ...
    initial_theta, alpha, kmax, tol)
    theta = initial_theta;
    fnorms = nan(kmax,1);
    succ_approx_newton = [theta'; nan(kmax,2)];
    for k = 1:kmax
        fk = feval(f, theta(1), theta(2));
        fnorms(k) = norm(double(fk));
        if fnorms(k) < tol
            break
        end
        theta = theta - alpha * (double(feval(hess, theta(1), theta(2))) \ ...
            double(feval(grad, theta(1), theta(2))));
        succ_approx_newton(k+1,:) = theta';
    end
    fnorms = fnorms(~isnan(fnorms));
    if k ~= kmax
        succ_approx_newton = succ_approx_newton(1:k,:);
    end
end
```

```
function [theta, fnorm_square, succ_approx_gauss_newton] = gauss_newton_method(f,...
    jacobian_func, theta_initial, kmax, tol)
    theta = theta_initial;
    fnorm_square = nan(kmax,1);
    succ_approx_gauss_newton = [theta'; nan(kmax,2)];
    for k = 1:kmax
        fk = feval(f, theta(1), theta(2));
        fnorm_square(k) = norm(double(fk))^2;
        if fnorm_square(k) < tol || (rank(double(feval(jacobian_func, theta(1), theta(2)))) ~=
            break
        end
        theta = theta - double(feval(jacobian_func, theta(1), theta(2))) \ ...
            double(feval(f, theta(1), theta(2)));
        succ_approx_gauss_newton(k+1,:) = theta';
    end
    fnorm_square = fnorm_square(~isnan(fnorm_square));
    if k ~= kmax
        succ_approx_gauss_newton = succ_approx_gauss_newton(1:k,:);
    end
end
```

```
function [theta, objectives, residuals, succ_approx_leven] = leven_marq(f, ...
    jacobian_func, theta_initial, lambda1, kmax, tol)
    n = length(theta_initial);
    theta = theta_initial;
    lambd = lambda1;
    objectives = nan(kmax,1);
    residuals = nan(kmax, 1);
    succ_approx_leven = [theta'; nan(kmax,2)];
    for k = 1:kmax
        fk = double(feval(f, theta(1), theta(2)));
        jacobian_res = double(feval(jacobian_func, theta(1), theta(2)));
        objectives(k) = norm(fk)^2;
        residuals(k) = norm(2*jacobian_res'*fk);
        if residuals(k) < tol
            break
        end
        theta_t = theta - [jacobian_res; sqrt(lambd)*eye(n)] \ [fk; zeros(n,1)];
        if norm(double(feval(f, theta_t(1), theta_t(2)))) < norm(fk)
            lambd = 0.8 * lambd;
            theta = theta_t;
            succ_approx_leven(k+1, :) = theta';
        else
            lambd = 2.0 * lambd;
        end
    end
    objectives = objectives(~isnan(objectives));
    residuals = residuals(~isnan(residuals));
    if k ~= kmax
```

```
        succ_approx_leven = succ_approx_leven(1:k,:);
    end
end
```