02610 Optimization and Data Fitting: Homework Assignment 1

Adrian Lopez Pirvu (s232101)
DTU Compute
Technical University of Denmark

November 7, 2023

Contents

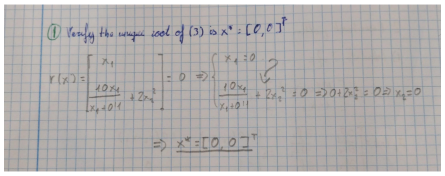
The folder submitted contains:

- The report in the `report.pdf` file
- Two Jupyter files: `exercices9.ipynb` and `assignment`, which contain the main code
- The following Python scripts: `GaussNewton_line.py`, `Levenberg_Marquardt.py`, `Levenberg_Marquardt_yq`, `linearLSQ.py`, and `newton.py`
- A file called `data_exe3_2023`, which contains data required for the exercise 3 of the Assignment

1 Exercises for Week 9 (60%).

$$r(x) = \begin{bmatrix} x_1 \\ \frac{100x_1}{x_1+0.1} + 2x_2^2 \end{bmatrix} = 0 \tag{1}$$

1.1 Verifty the unique root of (3) is $x^* = [0, 0]$



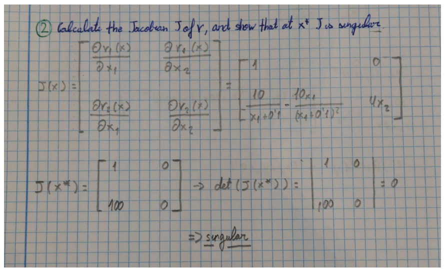
① Verify the unique root of (3) is $x^* = [0,0]^T$

$$r(x) = \begin{bmatrix} x_1 \\ \frac{10x_1}{x_1+0'1} + 2x_2^2 \end{bmatrix} = 0 \Rightarrow \begin{cases} x_1 = 0 \\ \frac{10x_1}{x_1+0'1} + 2x_2^2 = 0 \Rightarrow 0 + 2x_2^2 = 0 \Rightarrow x_2 = 0 \end{cases}$$

$$\Rightarrow \underline{x^* = [0,0]^T}$$

Figure 1: Enter Caption

1.2 Calculate the Jacobian J of r, and show that at x^* J is singular.



② Calculate the Jacobian J of r, and show that at x^* J is singular.

$$J(x) = \begin{bmatrix} \frac{\partial r_1(x)}{\partial x_1} & \frac{\partial r_1(x)}{\partial x_2} \\ \frac{\partial r_2(x)}{\partial x_1} & \frac{\partial r_2(x)}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{10}{x_1+0'1} - \frac{10x_1}{(x_1+0'1)^2} & 4x_2 \end{bmatrix}$$

$$J(x^*) = \begin{bmatrix} 1 & 0 \\ 100 & 0 \end{bmatrix} \rightarrow \det(J(x^*)) = \begin{vmatrix} 1 & 0 \\ 100 & 0 \end{vmatrix} = 0$$

$$\Rightarrow \underline{\text{singular}}$$

Figure 2: Enter Caption

1.3 Implement a Matlab function to calculate r and J with a given x.

```
def fun_rJ_exe(x):
  def r(x):
      return np.array([x[0], 10*x[0]/(x[0]+0.1) + 2*x[1]**2])

  def J(x):
     return np.array([[1, 0], [10/(x[0]+0.1) - 10*x[0]/(x[0]+0.1)**2, 4*x[1]]])

  return r(x), J(x)
```

1.4 Newton'smethod.

$$x_{k+1} = x_k - J(x_k)^{-1} r(x_k) \tag{2}$$

1.4.1 If the Jacobian J is not a square matrix, then can we still apply Newton's method?

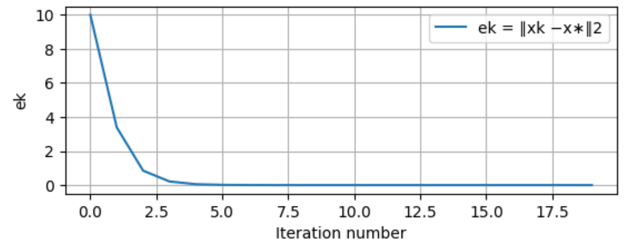
If the Jacobian is not a square matrix, then you cannot directly apply Newton's method as it is formulated. The reason is that Newton's method involves forming the inverse of the Jacobian matrix, which is only defined for square matrices. If the Jacobian is not square, then its inverse is not defined, and the method cannot proceed.

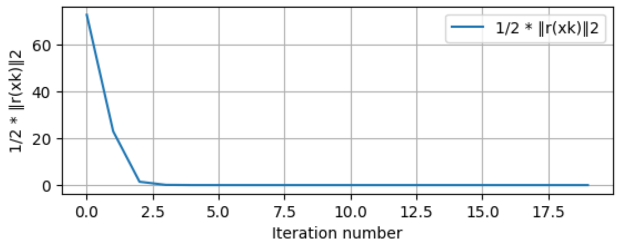
1.4.2 Revise your implementation of Newton's method for solving a minimization problem minxf(x), newton, to solve the nonlinear system (3) with the newton iteration step given in (4). Set the stopping criteria as:

$$\|r_k\|_\infty < 10^{-10} or k \geq 100n \tag{3}$$

1.4.3 Set the starting point $x_0 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$, and apply Newton's method to solve the equation: Plot $e_k = \|x_k - x^*\|^2$ and $\frac{1}{2}\|r(x_k)\|_2^2$ as functions of the iteration number. Which convergence rate can you see? If the method did not converge quadratically, what can be the reason?

The model convergence is linearly due to J is singular in the root. So the best result we can get is a linear convergence.





1.4.4 To further observe the convergence rate, plot (x1)k and (x2)k+1/(x2)k for k = 10,···. How did they change?

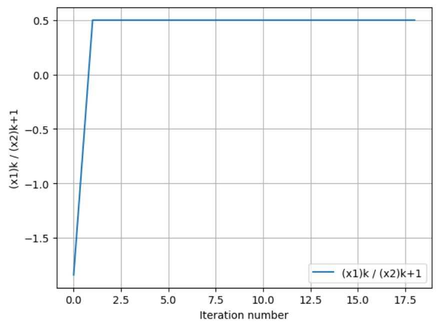


Figure 3: Enter Caption

With this plot we can see how the convergence rate is all time 0.5, for instance, we have a linear convergence.

1.5 Gauss-Newton method

$$x_{k+1} = x_k - (J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k) \tag{4}$$

1.5.1 Show that if J(xk) is a square matrix and nonsingular, the Gauss-Newton iteration step is identical to the Newton step.

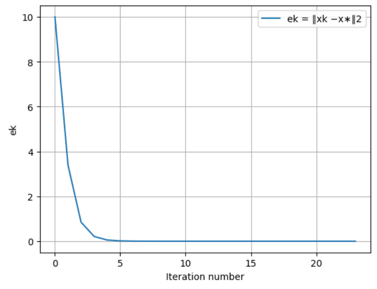
If the Jacobian $J(x_k)$ of the residuals r is square and nonsingular, then the Gauss-Newton iteration step is identical to the Newton step. This is because the Jacobian of the residuals is then equal to the Jacobian of the function, and the gradient of the residuals is equal to the gradient of the function.

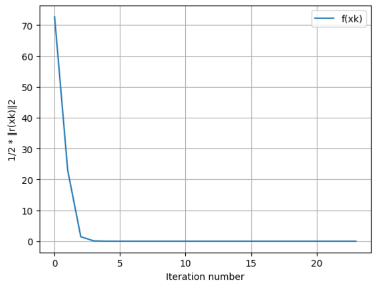
1.5.2 If the Jacobian J is not a square matrix, can we still apply the Gauss-Newton method?

Yes, we can still apply the Gauss-Newton method even if the Jacobian matrix is not square. The Gauss-Newton method is a modification of Newton's method designed to handle non-square systems. Instead of forming the inverse of the Jacobian matrix, the Gauss-Newton method approximates the Jacobian by its square root. This square root is always defined for non-square matrices, allowing the Gauss-Newton method to proceed even when the Jacobian is not square.

1.5.3 Plot ek and f(xk) as functions of the iteration number. Do you get the same plot as in Newton?

Yes, we get the same plots.



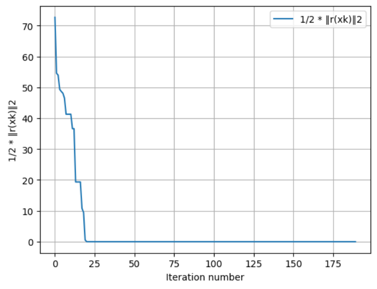


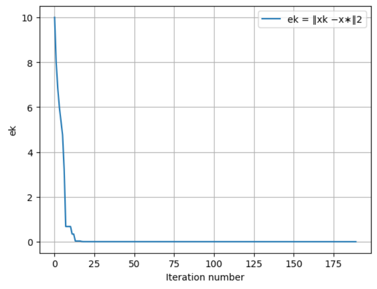
1.6 Levenberg-Marquardt method

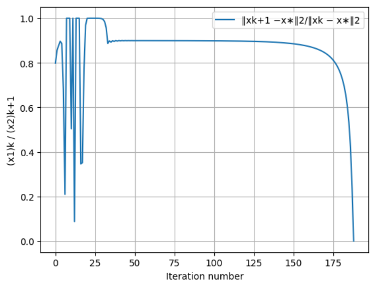
$$x_{k+1} = x_k - (J(x_k)^T J(x_k) + \lambda_k I)^{-1} J(x_k)^T r(x_k) \tag{5}$$

1.6.1 Call your implementation of the L-M method from Week 7, and set the same starting point. Plot ek and f(xk) as functions of the iteration number. To see the convergence rate, plot $||xk + 1 - x^*||_2/||xk - x^*||_2$. Does the method converge linearly?

1.6.2 Download the Matlab function Levenberg Marquardt yq.m, where I used another updating strategy for λ. In this Matlab function, the initial value $\lambda_0 = \tau\|J(z_0)^T J(z_0)\|_2$, and τ is an input. Set $\tau = 1$. Use this Matlab function to solve (1), and plot e_k and $f(x_k)$ as functions of the iteration number. Which convergence rate do you get now? Comparing with the previous result, which one is better?







I can not compare to the previous result because I was not available to do it.

1.6.3 In the figures from Levenberg Marquardt yq.m, you should see that the iteration process seems to stall between the step 20 and 30. What can be the reason?

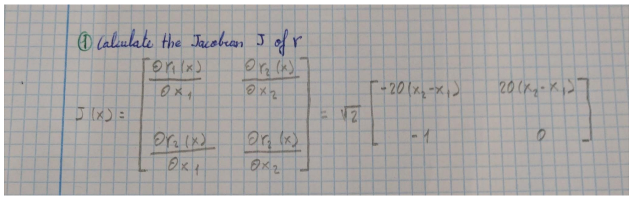
Could be because we are close to have a singular matrix so we get some numerical issues that can cause the algorithm to stall.

2 Rosenbrock problem

$$r(x) = \sqrt{2}\begin{bmatrix} 10(x_2 - x_1^2) \\ 1 - x_1 \end{bmatrix} = 0 \tag{6}$$

It is easy to see that this system has the unique solution $x^* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}^T$, and this is the unique minimizer for $f(x) = \frac{1}{2}\|r(x)\|_2^2$.

2.1 (by hand, 5%) Calculate the Jacobian J of r



① Calculate the Jacobian J of r

$$J(x) = \begin{bmatrix} \frac{\partial r_1(x)}{\partial x_1} & \frac{\partial r_2(x)}{\partial x_2} \\ \frac{\partial r_2(x)}{\partial x_1} & \frac{\partial r_2(x)}{\partial x_2} \end{bmatrix} = \sqrt{2}\begin{bmatrix} -20(x_2-x_1) & 20(x_2-x_1) \\ -1 & 0 \end{bmatrix}$$

Figure 4: Enter Caption

2.2 (5%) Implement a Matlab function to calculate r and J with a given x

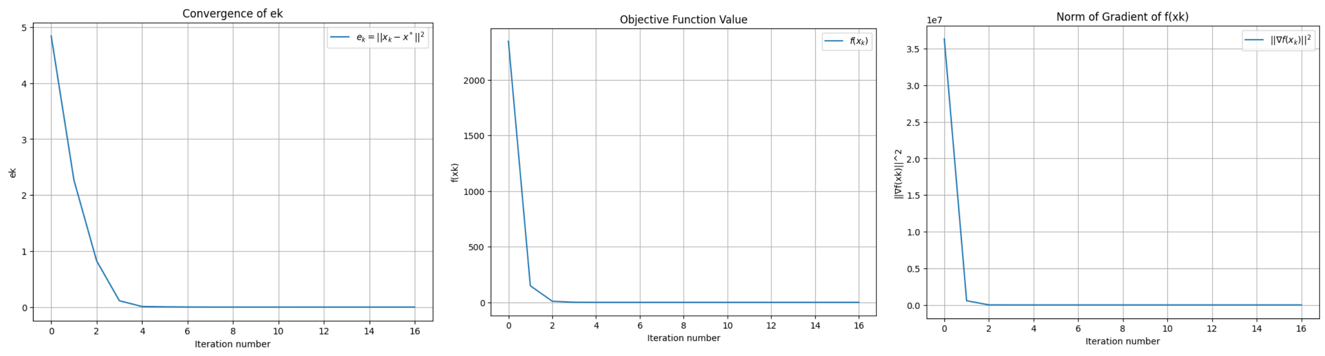
```
def fun_rJ_Rosen(x):
    def r(x):
        return np.array(np.sqrt(2)* np.array([10*(x[1]-x[0])**2, 1-x[0]]))

    def J(x):
       return np.array(np.sqrt(2) * np.array([[-20*(x[1]-x[0]), 20*(x[1]-x[0])], [-1, 0]]))

    return r(x), J(x)
```

2.3 Call the function LevenbergMarquardt_yq, to apply the Levenberg-Marquardt method to solve the minimization problem:

$$\min_x f(x) = \frac{1}{2}\|r(x)\|_2^2.$$

Set the starting point as $x_0 = \begin{bmatrix} -1.2 \\ 1 \end{bmatrix}^T$ and the initial λ as $10^{-3}\|J(x_0)^T J(x_0)\|_2^2$.
Plot $e_k = \|x_k - x^*\|_2^2$, $f(x_k)$, and $\|\nabla f(x_k)\|_2^2$ as functions of the iteration number. Determine the number of iterations the method needed until meeting the default stopping criteria. Calculate the convergence rate, and explain how you found it.

The method has converged in 16 iterations.



For analysing the convergence rate, we plot the convergence rate between two consecutive steps using this expression:

```
df = stats['dF']

# Calculate the Euclidean norms of the gradients
norms = [np.linalg.norm(df[i]) for i in range(len(df))]

# Calculate the convergence rate for each iteration
rates = [np.log(norms[i] / max(1, norms[i-1])) for i in range(1, len(norms))]
```

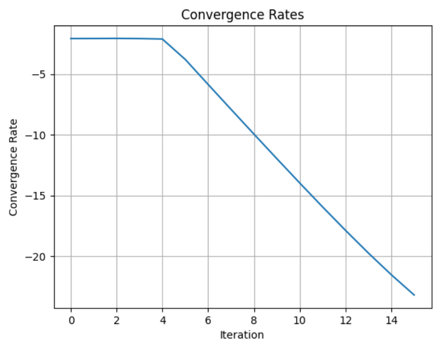


Figure 5: Enter Caption

We got this plot that indicates that the convergence rate is decreasing linearly from the 4th iteration.

3 Linear least squares with weights (12%)

$$\phi(t) = x_1 e^{-27t} + x_2 e^{-8t} + x_3 \tag{7}$$

3.1 (4%) Compute the least squares fit without taking the difference in noise levels into account. What is the solution of all three parameters? What is the 2-norm of the absolute error $\|e\|_2 = \|x - x^*\|_2$?

The solution for the three parameters is: $[0.77158691, 2.68426325, 0.23270323]$
The 2-norm of the absolute error is: 0.81732466

3.2 (6%) According to the standard deviations of the noise, how should we add weights to the problem, i.e, what is the weight matrix? Compute the weighted least squares solution and the 2-norm of the absolute error

We define the weight matrix W based on the noise standard:

```
# The first 10 data points have a standard deviation of 0.5, and the rest have 0.1
noise_std = np.array([0.5] * 10 + [0.1] * 40)

# Define the weight matrix W based on the noise standard deviations
W = np.diag(1 / (noise_std ** 2))
```

3.3 (2%) Compare the solutions without the weights and with the weights. Which one is more accurate?

- **Linear Least Squares (LSQ):**
  - Coefficients (c): $\begin{bmatrix}0.77158691 & 2.68426325 & 0.23270323\end{bmatrix}$
  - 2-Norm of Absolute Error: 0.81732
- **Weighted Least Squares (WLS):**
  - Coefficients (c_WLS): $\begin{bmatrix}1.19093158 & 2.27343584 & 0.27840347\end{bmatrix}$
  - 2-Norm of Absolute Error (WLS): 0.24741

The most accurate solution is the Weighted Least Squares