homework-5

January 3, 2025

1 Homework 5 (DL Friday, October 25 at 12:00 PM)

ELEC-E8740 - Basics of sensor fusion - Autumn 2024

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1.1 Question: Implement Gauss–Newton with line search for minimizing the cost function $J(x) = (1.1 - sin(x))^2$. Use grid search with grid $\gamma = [0, 0.1, 0.2, \dots, 1]$. Hint: Beware of the singularity of the derivative at the minimum.

1.1.1 Recall: Check "Gauss–Newton Algorithm with Line Search" in page 15 of houndout 5. Here, we want to use the "Exact Line Search on Grid" method as in page 11 of houndout 5.

1.1.2 Procedure: First, you need to define measurement function $g(x)$, its Jacobian $G(x)$, and the cost function $J(x)$. Consider the function $g(x)$ in this example as $g(x) = sin(x)$ and the measurement as $y = 1.1$.

[2]:
```
def g(x):
    return np.sin(x)

y = 1.1
```

1.1.3 Part a (1 point): In the section below, implement the code for the Jacobian, $G(x)$, and the cost function, $J(x)$.

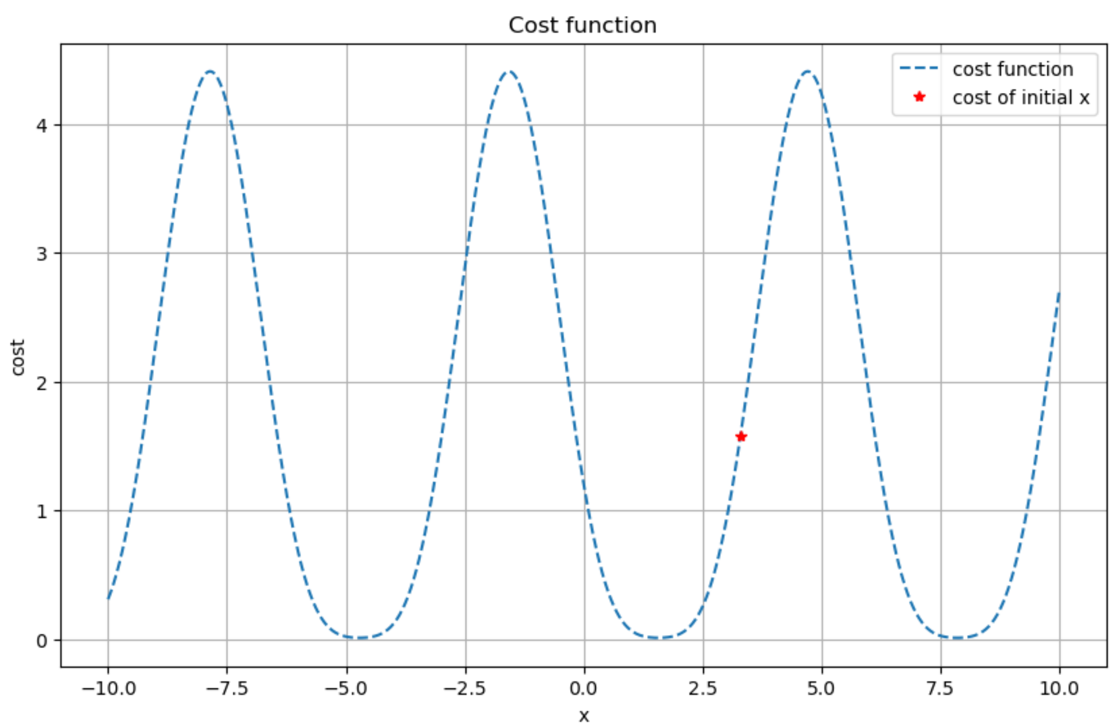
[15]:
```
def G(x):
    """
    Implement the Jacobian of g(x).
    """
    # YOUR CODE HERE
    return np.cos(x)

def J(x):
    """
    Implement the cost function.
    """
```

```
    # YOUR CODE HERE
    return (y - g(x))**2
```

[16]:
```
"""Check the result for several inputs"""
assert np.allclose(G(0.0), 1.0)
assert np.allclose(J(0.0), 1.21)
```

1.1.4 Before implementing Gauss-Newton with line search method, let's visually observe the minimums of the cost function $J(x)$, and the cost of an initial guess e.g. $x_0 = 3.3$.

[17]:
```
x0 = 3.3 # initial guess
xs = np.linspace(-10, 10, 1000)
cost_results = [J(x) for x in xs]

plt.figure(figsize=(10,6))
plt.plot(xs, cost_results, linestyle='--', label='cost function')
plt.plot(x0, J(x0), 'r*', label='cost of initial x')
plt.legend()
plt.grid()
plt.xlabel('x')
plt.ylabel('cost')
plt.title('Cost function')
plt.show()
```



1.1.5 Part b (1 point): Here we aim to implemet the "Exact Line Search on Grid" algorithm according to the Page 11 of houndout 5 (Alg.3).

[18]:
```
def Exact_LS_on_Grid(xi, del_x, Ng):
    '''
    Input:
        xi: result of the previous iteration
        del_x: the update direction
        Ng: the grid size
    Output:
        opt_gamma: optimal step size
    '''
    opt_gamma = 0 # Line 1 of Alg.3 (opt_gamma is gamma* in Alg.3)
    opt_cost = J(xi) # Line 1 of Alg.3 (opt_cost is J* in Alg.3)
    for j in range(0, Ng + 1): # Line 2 of Alg.3 (note that in the question␣
 ↪gamma is starting from zero)
        # Implement exact line search method (line 3 to 8 of Alg.3).
        # You should call the cost function J(x) that you previously coded.
        # YOUR CODE HERE
        gamma = j / Ng  # Compute gamma for the current grid point
        x_new = xi + gamma * del_x  # Calculate the new position
        cost_new = J(x_new)  # Compute the cost at the new position

        # If the new cost is lower than the current optimal cost, update the␣
 ↪optimal gamma and cost
        if cost_new < opt_cost:
            opt_gamma = gamma
            opt_cost = cost_new
    return opt_gamma
```

[19]:
```
"""Check the result for several inputs"""
assert np.allclose(Exact_LS_on_Grid(3.3 , -2, 10), 0.9)
assert np.allclose(Exact_LS_on_Grid(0.0 , 2, 10), 0.8)
```

1.1.6 Part c (2 points): In the section below, implement Gauss Newton with Line Search method (page 15).

[47]:
```
def Gauss_Newton_LS(x_0, y, Ng, number_of_iterations):
    x = np.zeros((number_of_iterations + 1,)) # do not change this line.
    x[0] = x_0
    for i in range(number_of_iterations):

        # Find the update direction (Line 3 Alg.5).
        # for that you need to call functions g and G
        # YOUR CODE HERE
```

```python
        g_xi = g(x[i])  # Evaluate g at the current estimate
        G_xi = G(x[i])  # Evaluate Jacobian at the current estimate

        residual = y - g_xi  # Compute residual

        # Update direction calculation
        if np.isclose(G_xi, 0):
            print("Warning: Jacobian is close to zero; setting del_x to zero.")
            del_x = 0
        else:
            del_x = -residual / G_xi


        # Find the optimal gamma by calling  Exact_LS_on_Grid function.
        # YOUR CODE HERE
        gamma = Exact_LS_on_Grid(x[i], del_x, Ng)

        # Update x (Line 5 Alg.5). you should calculate x[i+1] from x[i]
        # YOUR CODE HERE
        x[i + 1] = x[i] + gamma * del_x
        print(f"Iteration {i}: x[i] = {x[i]}, g(x[i]) = {g_xi}, del_x =␣
↪{del_x}, J(x[i]) = {J(x[i], y)}")
    return x
```

[ ]:

```python
```

1.1.7 In the section below, we check the final result of the "Gauss_Newton_LS" function by considering several initial points, specifically $[-7.0, 0.0, 3.3]$, running for 5 iterations, and employing a grid of size 10.

[48]:

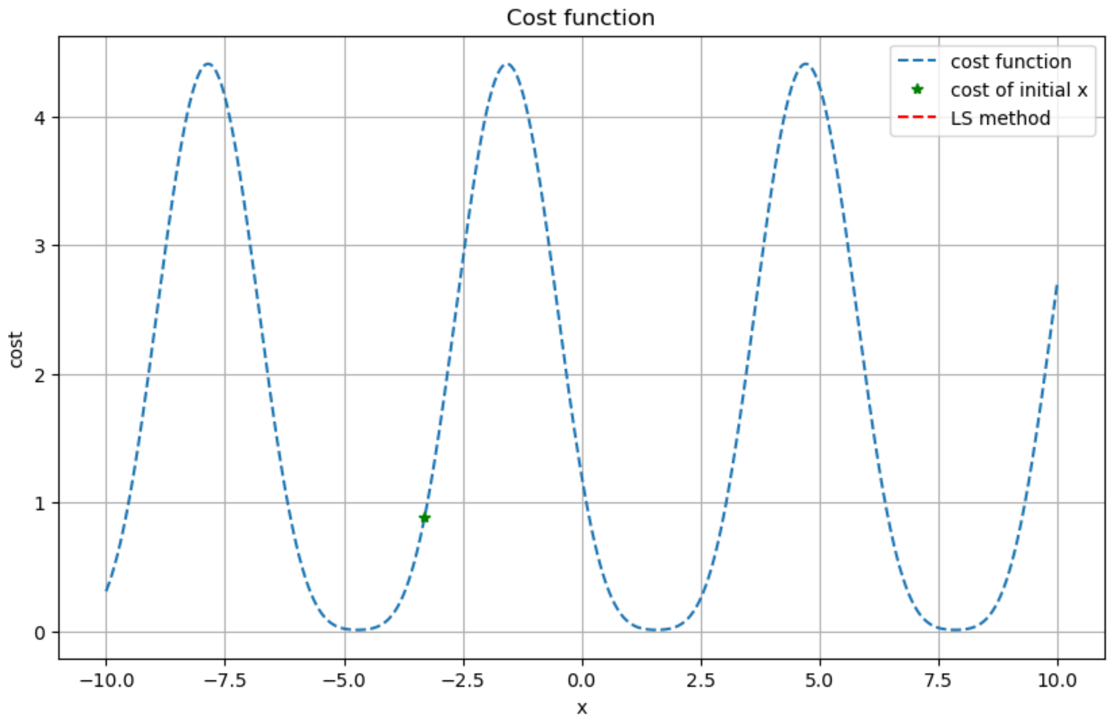
```python
"""Check the result for several inputs"""
assert np.allclose(Gauss_Newton_LS(-7.0, 1.1, 10, 5)[-1], -4.6695, rtol=1e-03,␣
↪atol=1e-04)
assert np.allclose(Gauss_Newton_LS(0.0, 1.1, 10, 5)[-1], 1.5603, rtol=1e-03,␣
↪atol=1e-04)
assert np.allclose(Gauss_Newton_LS(3.3, 1.1, 10, 5)[-1], 1.5672, rtol=1e-03,␣
↪atol=1e-04)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[48], line 2
      1 """Check the result for several inputs"""
----> 2 assert np.allclose(Gauss_Newton_LS(-7.0, 1.1, 10, 5)[-1], -4.6695,␣
↪rtol=1e-03, atol=1e-04)
      3 assert np.allclose(Gauss_Newton_LS(0.0, 1.1, 10, 5)[-1], 1.5603,␣
↪rtol=1e-03, atol=1e-04)
```

```
      4 assert np.allclose(Gauss_Newton_LS(3.3, 1.1, 10, 5)[-1], 1.5672,␣
  ↪rtol=1e-03, atol=1e-04)

Cell In[47], line 29, in Gauss_Newton_LS(x_0, y, Ng, number_of_iterations)
     26     # Update x (Line 5 Alg.5). you should calculate x[i+1] from x[i]
     27     # YOUR CODE HERE
     28     x[i + 1] = x[i] + gamma * del_x
---> 29     print(f"Iteration {i}: x[i] = {x[i]}, g(x[i]) = {g_xi}, del_x =␣
  ↪{del_x}, J(x[i]) = {J(x[i], y)}")
     30 return x

TypeError: J() takes 1 positional argument but 2 were given
```

1.1.8 Feel free to change the initial point in the next section and observe the results through the values of cost function.

[22]:
```
initial_point = -3.3
xx = Gauss_Newton_LS(initial_point, 1.1, 10, 5)

plt.figure(figsize=(10,6))
plt.plot(xs, cost_results, linestyle='--', label='cost function')
plt.plot(initial_point, J(initial_point), 'g*', label='cost of initial x')
plt.plot(xx, J(xx), 'r--', label='LS method')
plt.legend()
plt.grid()
plt.xlabel('x')
plt.ylabel('cost')
plt.title('Cost function')
plt.show()
```


Cost function
cost function
cost of initial x
LS method
cost
x
4
3
2
1
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

[ ]: