WROCLAW UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRONICS

FIELD: Electronics
SPECIALITY: Advanced Applied Electronics

Optimization Methods:
Nonlinear Equations

AUTHOR:

Jaroslaw M. Szumega

SUPERVISOR:

Rafal Zdunek, D.Sc, K-4/W4

GRADE:

Contents

Chapter 1

Solution to the given problems

Problem 1: Show that the following system of nonlinear equations:

$$F_1(\mathbf{x}) = x_1^3 + 3x_1^2 + 3x_1 - x_2 = 0,$$
$$F_2(\mathbf{x}) = x_1^2 + 2x_1 - x_2 + 1 = 0,$$

has the global minimum at $x^* = \begin{bmatrix} 0.46557 & 2.1479 \end{bmatrix}^T$, the local minimum at $x_1 = \begin{bmatrix} -1 & -0.5 \end{bmatrix}^T$, and the saddle point at $x_2 = \begin{bmatrix} -\dfrac{1}{3} & -\dfrac{7}{54} \end{bmatrix}^T$.

In general, the unconstrained nonlinear least squares problem is equivalent to:

$$\min_x \sum_{i=1}^{N} F_i^2(x)$$

with the objective function:

$$f(x) = \frac{1}{2} \sum_{i=1}^{N} F_i^2(x)$$

In the case of the following task, objective function will be

$$
\begin{aligned}
f(x) &= \frac{1}{2}(F_1^2 + F_2^2) \\
&= \frac{1}{2}[(x_1^6 + 6x_1^5 + 15x_1^4 - 2x_1^3 x_2 + 18x_1^3 - 6x_1^2 x_2 + 9x_1^2 - 6x_1 x_2 + x_2^2) \\
&\quad + (x_1^4 + 4x_1^3 - 2x_1^2 x_2 + 6x_1^2 - 4x_1 x_2 + 4x_1 + x_2^2 - 2x_2 + 1)] \\
&= \frac{1}{2}(x_1^6 + 6x_1^5 + 16x_1^4 - 2x_1^3 x_2 + 22x_1^3 - 8x_1^2 x_2 + 15x_1^2 - 10x_1 x_2 + 4x_1 + 2x_2^2 - 2x_2 + 1)
\end{aligned}
$$

And the points to check are:

$$x^* = [0.46557\ 2.1479]^T$$
$$x_1 = [-1\ -0.5]^T$$
$$x_2 = [-\frac{1}{3}\ -\frac{7}{54}]^T$$

Firstly, we can calculate the cost function gradient to verify above–mentioned points:

$$\nabla f(x^*) = 0$$

$$\frac{\delta f(x)}{\delta x_1} = 3x_1^5 + 15x_1^4 + 32x_1^3 - 3x_1^2 x_2 + 33x_1^2 - 8x_1 x_2 + 15x_1 - 5x_2 + 2$$

$$\frac{\delta f(x)}{\delta x_2} = -x_1^3 - 4x_1^2 - 5x_1 + 2x_2 - 1$$

And in fact, the mentioned in task description points are recognized as a solutions of the cost function gradient.

To check their nature, we need to calculate the Hessian and evaluate it according to the stationary points.

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\delta^2 f(x)}{\delta x_1^2} & \frac{\delta^2 f(x)}{\delta x_1 x_2} \\ \frac{\delta^2 f(x)}{\delta x_1 x_2} & \frac{\delta^2 f(x)}{\delta x_2^2} \end{bmatrix}$$

$$\nabla^2 f(x) = \begin{bmatrix} 15x_1^4 + 60x_1^3 + 96x_1^2 - 6x_1 x_2 + 66x_1 - 8x_2 + 15 & -3x_1^2 + 8x_1 + 5 \\ -3x_1^2 + 8x_1 + 5 & 2 \end{bmatrix}$$

$$for\ point\ [0.46557\ 2.1479] = \begin{bmatrix} 50.11 & -9.37 \\ -9.37 & 2 \end{bmatrix}$$

$$det > 0, M1 > 0, f(x) = 5.01e - 11,\ there\ is\ a\ minimum\ (global)$$

$$for\ point\ [-1\ -0.5] = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$det > 0, M1 > 0, f(x) = 0.25,\ there\ is\ a\ minimum$$

$$for\ point\ [-\frac{1}{3}\ -\frac{7}{54}] = \begin{bmatrix} 2.41 & -2.66 \\ -2.66 & 2 \end{bmatrix}$$

$$det < 0, M1 > 0, f(x) = 0.33,\ there\ is\ a\ saddle\ point$$

To make all the calculations above, the following script in Python was written to perform symbolic computations:

```python
1  #!/usr/bin/python
2
3  from sympy import diff, Symbol, latex, Matrix
4
5
6  def optimization(function, x1, x2):
7
8      print latex(function)
9
10     #calculating the gradient's elements
11     firstx1 = diff(function, x1)
12     firstx2 = diff(function, x2)
13
14     print firstx1.expand()
15     print firstx2.expand()
16
17
18     #calculations for Hessian elements
19     secondx1x1 = diff(function, x1, x1)
20     secondx1x2 = diff(function, x1, x2)
21     secondx2x1 = diff(function, x2, x1)
22     secondx2x2 = diff(function, x2, x2)
23
24     print "\n Hessian elements:\n\n"
25     print latex(secondx1x1) +"\t" + latex(secondx1x2)
26     print latex(secondx2x1) + "\t"+latex(secondx2x2)
27
28
29     point1 = [0.46557, 2.1479]
30     point2 = [-1.0, -0.5]
31     point3 = [-1.0/3, -7.0/54]
32
33     point = point3
34     print secondx1x1.subs(x1, point[0]).subs(x2, point[1]);
35     print secondx1x2.subs(x1, point[0]).subs(x2, point[1]);
36     print secondx2x1.subs(x1, point[0]).subs(x2, point[1]);
37     print secondx2x2.subs(x1, point[0]).subs(x2, point[1]);
38     print "\n\n"
39     print function.subs(x1, point[0]).subs(x2, point[1]);
40
41
42  def main():
43     x1 = Symbol('x1')
44     x2 = Symbol('x2')
45
46     f1 = x1**3 + 3*x1**2 + 3*x1 - x2
47     f2 = x1**2+2*x1 -x2 + 1
48     f = (f1**2 + f2**2)/2
49
50     function = f.expand()
51     optimization(function, x1,x2)
52
53  main()
```

Problem 2: Solve the following system of nonlinear equations with the selected nonlinear least squares methods.

$$F_1(\mathbf{x}) = x_1^3 - 3x_1^2 + 3x_1 - x_2 - 3 = 0,$$
$$F_2(\mathbf{x}) = x_1^2 - 2x_1 - x_2 = 0.$$

Draw the contour lines of the cost function in the nonlinear least square problem. Check the optimality points.

To resolve this nonlinear LS problem, the Newton and Broyden algorithm were used.

```
1  Newton's Method
2  iteration =   5
3
4  x =
5  2.4656   1.1479
6
7  Elapsed time is 0.000832081 seconds.
8
9
10 Broyden Method
11 iteration =   424
12
13 x =
14 2.4656  1.1479
15
16 Elapsed time is 0.055002 seconds.
```
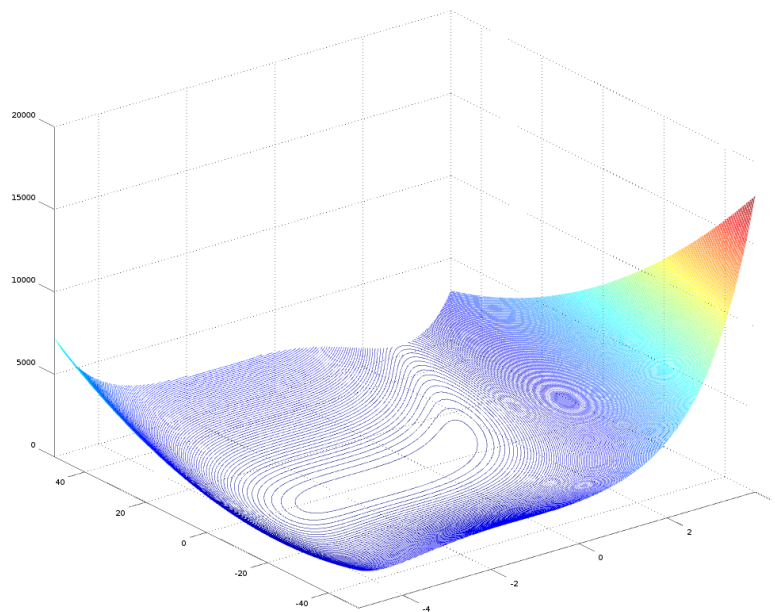

Figure 1.1 Contour plot of cost function.

Problem 3: Solve the following system of nonlinear equations:

$$2x_1 - x_2 = \exp(-x_1),$$
$$-x_1 + 2x_2 = \exp(-x_2),$$

starting from $\mathbf{x}_0 = \begin{bmatrix} -5 & -5 \end{bmatrix}^T$.

```
1  Newton Method
2  x =
3  0.56714
4  0.56714
5
6  iter_newton =   8
7  Elapsed time is 0.00742817 seconds.
8
9
10 Broyden Method
11 x =
12 0.56714
13 0.56714
14
15 iter_broyden =   8
16 Elapsed time is 0.00764704 seconds.
```
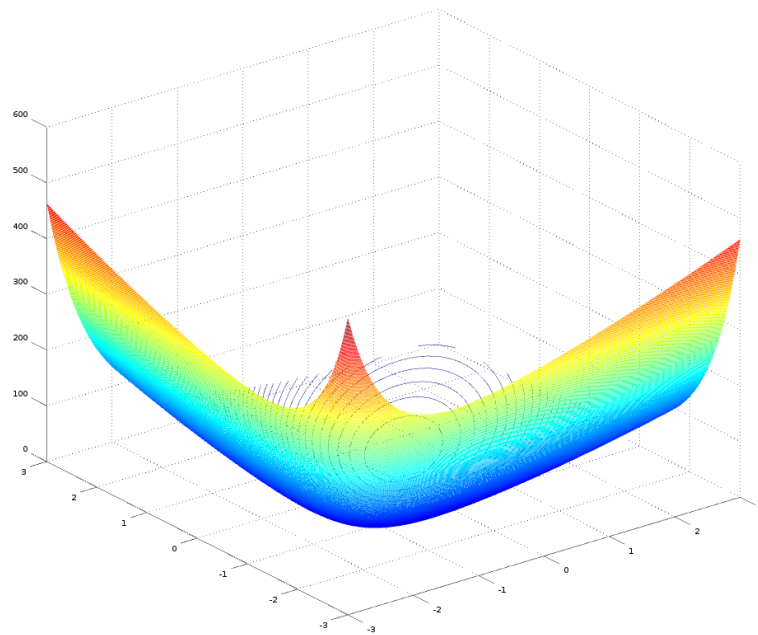

Figure 1.2 Contour plot of cost function

Chapter 2

Listings of algorithms

2.1 Coded selected algorithms

Algorithm 1 - Newton algorithm

```
1  function [x,k] = newton(fun,x,err)
2      stop = 2*err;
3      iter = 0;
4      while stop > err
5          [F,J] = fun(x);
6          dx = J\(-F);
7          x = x+dx;
8          [F,J] = fun(x);
9          stop = max(abs(F));
10         iter = iter + 1;
11     endwhile
12
13     iteration = iter
14 endfunction
```

Algorithm 2 - Broyden method

```
 1 function [xv,it]=broyden(f,x,tol,n)
 2
 3     fr=zeros(n,1); it=0; xv=x;
 4
 5     Br=eye(n);
 6     fr=f(xv);
 7
 8     while norm(fr)>tol
 9         it=it+1;
10         pr=-Br*fr;
11         tau=1;
12
13         xv1=xv+tau*pr;
14         xv=xv1;
15
16         oldfr=fr;
17         fr=f(xv);
18
19         %Update approximation to Jacobian using Broydens formula
20         y=fr-oldfr;
21         oldBr=Br;
22         oyp=oldBr*y-pr;
23         pB=pr'*oldBr;
24
25         for i=1:n
26             for j=1:n
27                 M(i,j)=oyp(i)*pB(j);
28             end;
29         end;
30         Br=oldBr-M./(pr'*oldBr*y);
31     end;
32 endfunction
```

Bibliography

[1] J. Nocedal, S. J. Wright, Numerical Optimization, Springer, 1999,

[2] Zdunek R., Optimization Methods - lecture slides.

[3] Mathworks webpage, "Unconstrained Optimization Algorithms", https://www.mathworks.com/help/optim/ug/unconstrained-nonlinear-optimization-algorithms.html

[4] Nonlinear Programming course webpage, North Carolina State University, http://www4.ncsu.edu/ kksivara/ma706/