CIE 6010/MDS 6118 Midterm Exam: Part 2

Problem Solving in Matlab: CIE 25 points / MDS 35 points

Take-home. Due 5:00pm, November 9, 2018

— Room 511, Dao Yuan Building —

No discussions are allowed except possibly with the instructor or TAs. Please include this page as the cover page for your submission.

Name (print): ____________________________

Student Number: ____________________________

Course Number: ____________________________

Your Matlab function should have the following interface:

```
function [x,hist,iter] = myphase(A,B,d,x0,tol,maxit,itGN)
%
% Solve the phase retrieval least squares problem
%        min f(x) = 0.5*norm(|(A + iB)x|^2 - d)^2
% input:
%      A,B,d = real-valued data
%         x0 = initial guess
%        tol = tolerance
%      maxit = maximum number of iterations in total
%       itGN = min number of Gauss-Newton iterations before
%              possibly switching to Newton iterations
% output:
%          x = computed solution
%       hist = history vector for c_k = norm(g_k)/(1+f_k)
%       iter = total number of iterations taken
```

A copy of my code myphase.m

```
function [ x,hist,iter ] = myphase(A,B,d,x0,tol,maxit,itGN )
% Usage:
% Solve the phase retrieval least squares problem:
%      min f(x) = 0.5 * norm(|(A + iB)x|^2 - d)^2
%
% Input:
%     A, B, d = real-valued data
%          x0 = initial guess
%         tol = tolerance
%       maxit = maximum number of iterations in total
%        itGN = min number of Gauss-Newton iterations before
% Output:
%           x = computed solution
%        hist = history vector for c_k = norm(g_k) / (1 + f_k)
%        iter = total number of iterations taken
x = x0;
hist = [];
itGN = itGN*5;
for iter = 1:maxit
    [J,r] = Jacobian_compute(A,B,d,x);
    nabla = J' * r;
    nabla2_fake = J' * J;
    f = norm(r)^2*0.5;
    gnrm = norm(nabla);
    crit = gnrm / (1 + f);
    fprintf('iter %i: f(x) = %9.2e gnrm = %9.2e crit = %9.2e\n',...
       iter, f, gnrm, crit)
    hist(iter) = crit;
    if crit <= tol, break;end
    if iter <= itGN
        m = nabla2_fake \ nabla;
        x = x - m;
    else
        S = 2*(r'.*A'*A+r'.*B'*B);
        nabla2 = nabla2_fake + S;
        m = nabla2 \ nabla;
        x = x - m;
    end
end
end

function [J,r] = Jacobian_compute(A,B,d,x)
Ax = A*x;
Bx = B*x;
r = Ax.^2 + Bx.^2-d;
J = 2*A.*Ax+2*B.*Bx;
end
```

MATLAB Screen Printout

Command Window

```
>> test_phase
(return for default) n = 1000
Phase: n = 1000, m = 5000

iter:   1  f(x) = 1.503e+10  gnrm = 5.426e+08  crit = 3.610e-02
iter:   2  f(x) = 4.677e+09  gnrm = 1.606e+09  crit = 3.433e-01
iter:   3  f(x) = 3.591e+08  gnrm = 2.010e+08  crit = 5.597e-01
iter:   4  f(x) = 1.588e+08  gnrm = 3.127e+07  crit = 1.970e-01
iter:   5  f(x) = 1.227e+08  gnrm = 3.726e+07  crit = 3.038e-01
iter:  86  f(x) = 8.722e+06  gnrm = 1.634e-04  crit = 1.873e-11
iter:  87  f(x) = 8.722e+06  gnrm = 1.262e-04  crit = 1.447e-11
iter:  88  f(x) = 8.722e+06  gnrm = 9.595e-05  crit = 1.100e-11
iter:  89  f(x) = 8.722e+06  gnrm = 7.376e-05  crit = 8.457e-12
--- yzphase: iter  89 relerr = 1.359e-01  time 11.155 ---

Current plot held
iter:   1  f(x) = 1.503e+10  gnrm = 5.426e+08  crit = 3.610e-02
iter:   2  f(x) = 4.677e+09  gnrm = 1.606e+09  crit = 3.433e-01
iter:   3  f(x) = 3.591e+08  gnrm = 2.010e+08  crit = 5.597e-01
iter:   4  f(x) = 1.588e+08  gnrm = 3.127e+07  crit = 1.970e-01
iter:   5  f(x) = 1.227e+08  gnrm = 3.726e+07  crit = 3.038e-01
iter:   6  f(x) = 1.066e+08  gnrm = 3.733e+07  crit = 3.501e-01
iter:   7  f(x) = 1.008e+08  gnrm = 4.149e+07  crit = 4.116e-01
iter:   8  f(x) = 9.249e+07  gnrm = 4.637e+07  crit = 5.013e-01
iter:   9  f(x) = 8.114e+07  gnrm = 5.287e+07  crit = 6.517e-01
iter:  10  f(x) = 7.360e+07  gnrm = 8.889e+07  crit = 1.208e+00
iter:  11  f(x) = 7.499e+07  gnrm = 1.318e+08  crit = 1.758e+00
iter:  12  f(x) = 4.128e+07  gnrm = 9.818e+07  crit = 2.378e+00
iter:  13  f(x) = 1.312e+07  gnrm = 3.279e+07  crit = 2.499e+00
iter:  14  f(x) = 8.852e+06  gnrm = 2.894e+06  crit = 3.270e-01
iter:  15  f(x) = 8.732e+06  gnrm = 1.600e+05  crit = 1.833e-02
iter:  16  f(x) = 8.724e+06  gnrm = 5.345e+04  crit = 6.127e-03
iter:  17  f(x) = 8.723e+06  gnrm = 3.015e+04  crit = 3.457e-03
iter:  18  f(x) = 8.722e+06  gnrm = 5.360e+02  crit = 6.145e-05
iter:  19  f(x) = 8.722e+06  gnrm = 1.892e-02  crit = 2.169e-09
iter:  20  f(x) = 8.722e+06  gnrm = 1.363e-06  crit = 1.563e-13
--- yzphase: iter  20 relerr = 1.359e-01  time 3.51279 ---

iter 1: f(x) =  1.50e+10 gnrm =  5.43e+08 crit =  3.61e-02
iter 2: f(x) =  4.68e+09 gnrm =  1.61e+09 crit =  3.43e-01
iter 3: f(x) =  3.59e+08 gnrm =  2.01e+08 crit =  5.60e-01
iter 4: f(x) =  1.59e+08 gnrm =  3.13e+07 crit =  1.97e-01
iter 86: f(x) =  8.72e+06 gnrm =  1.63e-04 crit =  1.87e-11
iter 87: f(x) =  8.72e+06 gnrm =  1.26e-04 crit =  1.45e-11
iter 88: f(x) =  8.72e+06 gnrm =  9.59e-05 crit =  1.10e-11
iter 89: f(x) =  8.72e+06 gnrm =  7.38e-05 crit =  8.46e-12
--- myphase: iter  89 relerr = 1.359e-01  time 9.38611 ---

Current plot held
iter 1: f(x) =  1.50e+10 gnrm =  5.43e+08 crit =  3.61e-02
iter 2: f(x) =  4.68e+09 gnrm =  1.61e+09 crit =  3.43e-01
iter 3: f(x) =  3.59e+08 gnrm =  2.01e+08 crit =  5.60e-01
iter 4: f(x) =  1.59e+08 gnrm =  3.13e+07 crit =  1.97e-01
iter 5: f(x) =  1.23e+08 gnrm =  3.73e+07 crit =  3.04e-01
iter 6: f(x) =  1.07e+08 gnrm =  3.73e+07 crit =  3.50e-01
iter 7: f(x) =  1.01e+08 gnrm =  4.15e+07 crit =  4.12e-01
iter 8: f(x) =  9.25e+07 gnrm =  4.64e+07 crit =  5.01e-01
iter 9: f(x) =  8.11e+07 gnrm =  5.29e+07 crit =  6.52e-01
iter 10: f(x) =  7.36e+07 gnrm =  8.89e+07 crit =  1.21e+00
iter 11: f(x) =  7.50e+07 gnrm =  1.32e+08 crit =  1.76e+00
iter 12: f(x) =  4.13e+07 gnrm =  9.82e+07 crit =  2.38e+00
iter 13: f(x) =  1.31e+07 gnrm =  3.28e+07 crit =  2.50e+00
iter 14: f(x) =  8.85e+06 gnrm =  2.89e+06 crit =  3.27e-01
iter 15: f(x) =  8.73e+06 gnrm =  1.60e+05 crit =  1.83e-02
iter 16: f(x) =  8.72e+06 gnrm =  5.35e+04 crit =  6.13e-03
iter 17: f(x) =  8.72e+06 gnrm =  2.40e+03 crit =  2.75e-04
iter 18: f(x) =  8.72e+06 gnrm =  2.09e-01 crit =  2.39e-08
iter 19: f(x) =  8.72e+06 gnrm =  9.81e-06 crit =  1.12e-12
--- myphase: iter  19 relerr = 1.359e-01  time 2.86407 ---
>>
```


Figure 1: Matlab Screen Printout for file test_phase.m

MATLAB Plots

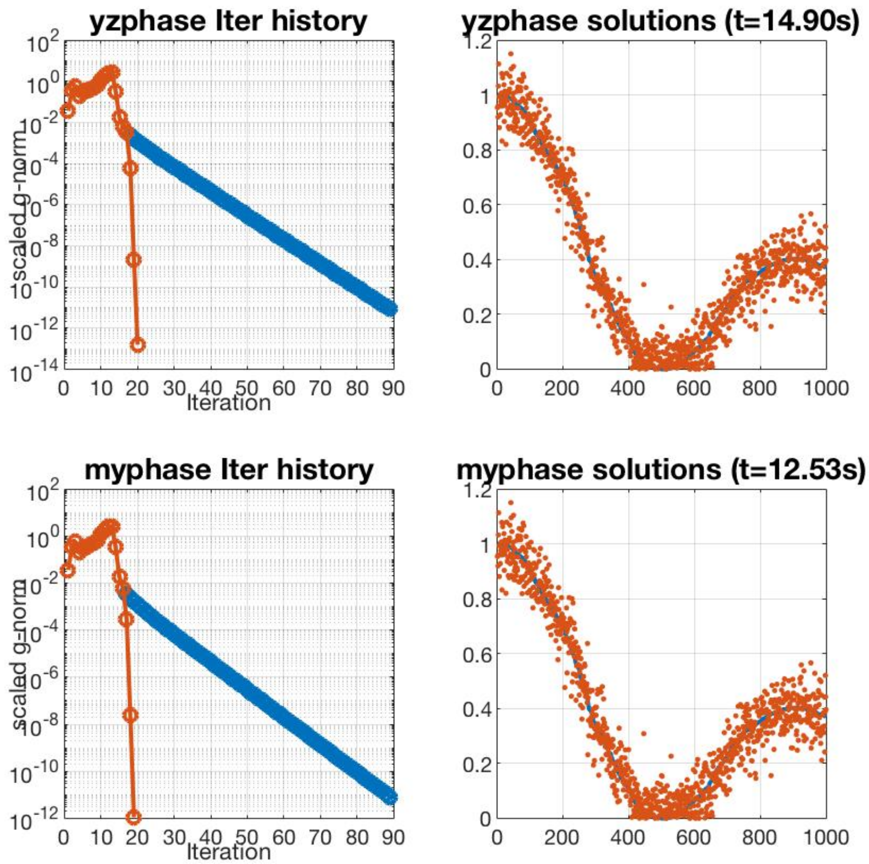

Figure 2: Matlab plots from file test_phase.m

Summary for Phase Retrieval MATLAB Problem

Jie Wang

School of Science and Engineering (pure mathematics major)

The Chinese University of Hongkong, Shenzhen

116010214@link.cuhk.edu.cn

I. Introduction

[1] This project is about solving a *phase retrieval least squares problem* given as follows:

$$\begin{array}{ll} \min_{x\in\mathbb{R}^n} & f(x) \triangleq \frac{1}{2}\|r(x)\|^2 \\ \text{such that} & r_i(x) = (a_i^{\mathrm{T}}x)^2 + (b_i^{\mathrm{T}}x)^2 - d_i, i = 1, 2, \ldots, m, \end{array} \tag{1}$$

where $A + iB$ are known matrix with $a_i^{\mathrm{T}}, b_i^{\mathrm{T}}$ denoting the i-th row of A and B, respectively, d is our observation vector with d_i denoting its i-th entry. Also define the matrices $E_i := a_i a_i^{\mathrm{T}} + b_i b_i^{\mathrm{T}}$, $i = 1, \ldots, m$.

After computations, we obtain that:

$$J^{\mathrm{T}}(x) \cdot J(x) = 2\sum_{i=1}^{m} E_i(xx^{\mathrm{T}})E_i \tag{2a}$$

$$S(x) = 2\sum_{i=1}^{m} E_i \cdot r_i(x) \tag{2b}$$

$$\nabla f(x) = S(x) \cdot x \tag{2c}$$

$$\nabla^2 f(x) = J^{\mathrm{T}}(x) \cdot J(x) + S(x) \tag{2d}$$

The Gauss-Newton direction and Newton direction at a given x is given by:

$$d_{\text{GN}} = -[J^{\mathrm{T}}(x) \cdot J(x)]^{-1} \nabla f(x) \tag{3a}$$

$$d_{\text{Newton}} = -[\nabla^2 f(x)]^{-1} \nabla f(x) \tag{3b}$$

The suggested algprithmatic approach is listed as follows:

II. Decisions & Arrangements on Computation

a) Choice on itGN: As $n = 1000$, adopting Newton's method too soon is not a good idea. We decide to change the minimum iteration taken to swithch to Newton's method as $itGN = 15$.

b) Compute $S(x)$ and $J^{\mathrm{T}}(x) \cdot J(x)$ smartly: We observe that the barrier for computation is due to the Hessian matrix and $J^{\mathrm{T}}(x) \cdot J(x)$. Pre-save the data matrix $\{E_i\}_{i=1,\ldots,m}$ is not a good idea, since it will occupy too much space such that MATLAB cannot handle it. Therefore we should not implement formula (2a) and (2b) directly but re-formulate and implement it in terms of A, B, d, x.

c) Pre-set Function handles: Another barrier is that computing the Hessian and Jacobian matrix requires calling specific structures too many times. Thus we should pre-write the funtion handle for computing these values and therefore can speed up at least 20%.

[1]This MATLAB code has been complied and examined in MATLAB2016b, MATLAB2017a version

Algorithm 1: Combination of Gauss-Newton and Newton's method to solve (1)

Input : Real-valued data A, B, d; initial guess $x0$; switching threshold itGN.
Output: solution to the least squares problem x

```
x ← x0;
for iter = 1,2,...,itGN do
    x ← x + d_GN;
    check termination criteria
end
for iter = itGN+1,...,maxit do
    x ← x + d_Newton;
    check termination criteria
end
return x;
```

III. Observations

a) Local convergence rate: Although our conclusion tells us that the Newton and Gauss-Newton method on Least squares problem both have *locally quadratic convergence rate* in general, during the experiment, we find that Newton's method has much faster convergence rate than Gauss-Newton. From the plot we can see that the Gauss-Newton only allows a *linear convergence rate*, however, with each iteration being *much less expansive* than the Newton's method. Therefore we choose to implement Gauss-Newton method at the beginning, and when close to the optimal point, we change our method into Newton.

b) Efficiency interpretation: As we can see, the convergence for the combination of G-N and Newton's method requires 19 iterations; while the pure G-N method requires 89 iterations. The reason is that the G-N method approximates Hessian with $J^{\mathrm{T}}J$, which sacrifice accuracy for calculation speed.

c) Comments on usage of bsxfun: Although bsxfun speeds up the computation for Jacobian and Hessian matrix greatly, if we re-arrange the computation, we can achieve even faster speed than bsxfun by simply calling the *dot product* and *matrix multiplication*. The reason it that the bsxfun only provides element-wise operation, but the combination of element-wise operation and matrix product operation will perform better in general.