CS4220 PS8

Due: May 6, 2015

1: Nearest Points For a given point $(a, b) \in \mathbb{R}^2$, we want to find the nearest point (x, y) that lies on the hyperbola $xy = 1$.

1. Write a Lagrangian function $L(x, y, \lambda)$ such that the desired point is a stationary point of L.
2. Write a Newton iteration to find the stationary point of L for $(a, b) = (3, 4)$. Use the starting guess $(x, y, \lambda) = (a, b, 0)$, and demonstrate quadratic convergence.

Note: As the point is to demonstrate a knowledge of Lagrange multipliers, we will not give credit for solutions that eliminate the constraint in advance.

Solution:

1. For (a, b), the nearest point (x, y) on $xy = 1$ can be found by setting up the problem as a constrained minimization problem.

   The Lagrange function $L(x, y, \lambda)$ can be defined as

   $$L(x, y, \lambda) = f(x, y) - \lambda g(x, y), \tag{1}$$

   where $f(x, y) = d^2 = (x - a)^2 + (y - b)^2$ is the distance squared between two points. $g(x, y) = xy - 1 = 0$ is the constraint equation, and λ is the Lagrange multiplier. (We use d^2, instead of d, in the definition of L in order to simplify the its derivatives without changing the nature of the problem.)

   To make the desired (nearest) point a stationary point of L, the gradient of L at the stationary point (x, y) must be zero:

   $$\nabla L(x, y, \lambda) = 0. \tag{2}$$

   The x- and y-component equations of Eq.(2), plus $g(x, y) = 0$ (the constraint equation) are the system we want to solve for x, y, and λ:

   $$\begin{aligned} \frac{\partial f}{\partial x} - \lambda \frac{\partial g}{\partial x} = 2(x - a) - \lambda y &= 0, & (3) \\ \frac{\partial f}{\partial y} - \lambda \frac{\partial g}{\partial y} = 2(y - b) - \lambda x &= 0, & (4) \\ g(x, y) = xy - 1 &= 0. & (5) \end{aligned}$$

2. Newton's iteration: Let s be the solution vector, F the equation vector and J the Jacobian matrix of F, respectively:

$$s = \begin{pmatrix} x \\ y \\ \lambda \end{pmatrix}, \tag{6}$$

$$F = \begin{pmatrix} F_1 \\ F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 2(x-a) - \lambda y \\ 2(y-b) - \lambda x \\ xy - 1 \end{pmatrix} = 0, \tag{7}$$

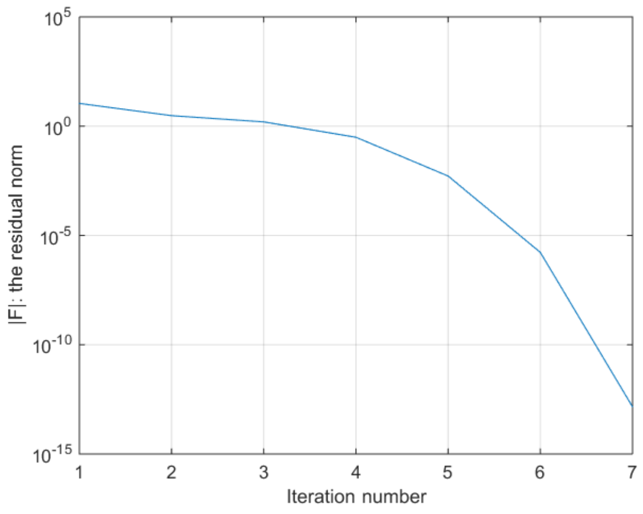
$$J = \begin{pmatrix} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial y} & \frac{\partial F_1}{\partial \lambda} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial y} & \frac{\partial F_2}{\partial \lambda} \\ \frac{\partial F_3}{\partial x} & \frac{\partial F_3}{\partial y} & \frac{\partial F_3}{\partial \lambda} \end{pmatrix} = \begin{pmatrix} 2 & -\lambda & -y \\ -\lambda & 2 & -x \\ y & x & 0 \end{pmatrix}. \tag{8}$$

The solution to the equation system can be obtained by using the Newton's iteration, with the initial guess vector being set at $s_0 = (a, b, 0)^T$.

3. The Newton's iteration converges ($\|F\| < 10^{-10}$) in 7 iterations. The stationary point is:

$$x = 0.26236, \quad y = 3.81156, \quad \lambda = -1.43649.$$

As illustrate by the plot of the residual norm versus iteration number: the reduction rate in residual norms is quadratic: from iteration number 4 to 7, the norms are 10^{-1}, 10^{-3} 10^{-6} and 10^{-13}, respectively. The residual norm versus iteration number is plotted below:



The script of Newton's iteration for Problem 1 is listed here:

```
%
% script for  CS4220 PS8's problem 1
%
% soluton vector s=(x, y, lambda)
% (a,b) is given, find the nearest point (x,y) on xy-1=0.
% by using the Lagrangian function
%
a=3; b=4;
s=zeros(3,1);
rtol=1e-10;
F=zeros(3,1);
J=zeros(3,3);
% initialization  s_0:
s=[a;b;0];
maxiter=10;
rnorms=zeros(maxiter,1);
for iter=1:maxiter
    % s = ( x, y, lambda), so x is s(1), y is s(2), lambda=s(3)
    F(1)=2*(s(1)-a) - s(3)*s(2);
    F(2)=2*(s(2)-b) - s(3)*s(1);
    F(3)=s(1)*s(2)-1;
    %
    rnorms(iter)=norm(F);
    fprintf('\niter=%d,|F|=%g',iter,rnorms(iter));
    fprintf('\nx=%g,  y=%g,  lambda=%g\n', s(1),s(2),s(3));
    if ( rnorms(iter) < rtol )
        rnorms=rnorms(1:iter);
        break;
    end
    % The Jacobian matrix
    J=[2 -s(3) -s(2); -s(3) 2 -s(1); s(2)  s(1)  0];
    p = -J\F;
    s = s+p;
end
nn=1:iter;
semilogy(nn, rnorms);
xlabel('Iteration number');
ylabel('|F|: the residual norm');
```

2: Nonlinear Least Squares In this problem, we consider a nonlinear least squares fitting problem in which we fit the coefficient vector β defining a rational function

$$f(x;\beta) = \frac{\beta_1 + \beta_2 x + \beta_3 x^2 + \beta_4 x^3}{1 + \beta_5 x + \beta_6 x^2 + \beta_7 x^3} \tag{9}$$

by minimizing

$$\phi(\beta) = \sum_j (f(x_j;\beta) - y_j)^2 \tag{10}$$

Your task: Complete the MATLAB script `ps8thuber.m` by filling in the code marked `TODO` with an appropriate solver iteration. You may use Gauss-Newton or Levenberg-Marquardt; I used Gauss-Newton with a line search (necessary to achieve convergence). Terminate when $\|J^T(f-y)\| < 10^{-8}$.

Solution: We employ the nonlinear least-squares method in minimizing $\phi(\beta)$ in Eq.(10):

$$\min_\beta \phi(\beta) = \min_\beta \left(\sum_{i=1}^m (f_i - y_i)^2 \right),$$

where $\beta = (\beta_1, \beta_2, \ldots, \beta_n)^T$ is the solution vector, n is the length of the solution vector ($n = 7$ in this problem) and m is the number of data points (x_i, y_i).

We formulate the Gauss-Newton method for this problem as follows:

1. For brevity, we denote $f(x_i;\beta) = f_i$, then Eq.(10) can be written as

$$\phi(\beta) = (f_1 - y_1)^2 + (f_2 - y_2)^2 + \ldots + (f_m - y_m)^2,$$

where m is the number of data points. We calculate the derivatives of ϕ with respect to β and give them here:

$$\begin{aligned}
\frac{\partial \phi}{\partial \beta_1} &= 2\left[\frac{\partial f_1}{\partial \beta_1}(f_1 - y_1) + \frac{\partial f_2}{\partial \beta_1}(f_2 - y_2) + \ldots + \frac{\partial f_m}{\partial \beta_1}(f_m - y_m)\right], \\
\frac{\partial \phi}{\partial \beta_2} &= 2\left[\frac{\partial f_1}{\partial \beta_2}(f_1 - y_1) + \frac{\partial f_2}{\partial \beta_2}(f_2 - y_2) + \ldots + \frac{\partial f_m}{\partial \beta_2}(f_m - y_m)\right], \\
&\vdots \\
\frac{\partial \phi}{\partial \beta_n} &= 2\left[\frac{\partial f_1}{\partial \beta_n}(f_1 - y_1) + \frac{\partial f_2}{\partial \beta_n}(f_2 - y_2) + \ldots + \frac{\partial f_m}{\partial \beta_n}(f_m - y_m)\right].
\end{aligned}$$

Then the gradient of ϕ is expressed as

$$\nabla\phi(\beta) = \begin{pmatrix} \frac{\partial \phi}{\partial \beta_1} & \frac{\partial \phi}{\partial \beta_2} & \ldots & \frac{\partial \phi}{\partial \beta_n} \end{pmatrix}^T = 2J^T(f-y) = 2J^T r, \tag{11}$$

where $r = f - y$, and

$$J = \begin{pmatrix} \dfrac{\partial f_1}{\partial \beta_1} & \dfrac{\partial f_1}{\partial \beta_2} & \cdots & \dfrac{\partial f_1}{\partial \beta_n} \\ \dfrac{\partial f_2}{\partial \beta_1} & \dfrac{\partial f_2}{\partial \beta_2} & \cdots & \dfrac{\partial f_2}{\partial \beta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial \beta_1} & \dfrac{\partial f_m}{\partial \beta_2} & \cdots & \dfrac{\partial f_m}{\partial \beta_n} \end{pmatrix}$$

is the Jacobian matrix of $f(\beta)$ (a vector) with respect to β. (For $\nabla\phi(\beta)$, J^T is used.)

2. The condition at a critical point ($\nabla\phi = 0$) must be satisfied for the minimization of ϕ. We can directly apply this condition to the Taylor series of $\nabla\phi$ in the neighborhood of the critical point (while dropping the higher-order derivatives):

$$\nabla\phi(\beta^*) = 0 \approx \nabla\phi(\beta) + \nabla^2\phi(\beta)p. \tag{12}$$

From the equation above we have arrived at a system of equations for p, the search direction:

$$\nabla^2\phi(\beta)p = -\nabla\phi(\beta). \tag{13}$$

When solving Eq.(13) for p, instead of using the exact formulation of $\nabla^2\phi(\beta)$, the Gauss-Newton method approximates $\nabla^2\phi(\beta)$ (the Hessian matrix) so that no second derivative is required:

$$\nabla^2\phi(\beta) = 2J^TJ + \sum_{i,j=1}^{m} \frac{\partial^2\phi}{\partial\beta_i\partial\beta_j} \approx 2J^TJ. \tag{14}$$

This approximation significantly simplifies the equation for p.

3. Gauss-Newton starts from an initial guess β_0, then iteratively approaches the optimal solution β^* by solving the following normal equation for p_k (the 2s can be crossed out):

$$2(J^TJ)\,p_k = -2J^T(f_k - y_k) = -2J^Tr_k. \tag{15}$$

Note that we have substituted $2J^TJ$ for $\nabla^2\phi(\beta)$ in Eq.(13) to get Eq.(15) for the Gauss-Newton method. After p_k is determined by Eq.(15) in each iteration, a *line search* along the direction of p_k is conducted when advancing the current-level β_k to the next level β_{k+1}:

$$\beta_{k+1} = \beta_k + \alpha_k p_k,$$

where α_k is an appropriate scalar factor which should minimize $\phi(\beta_{k+1})$ in the search direction. A weak line search strategy in p. 264 from the textbook (Ascher and Greif) is adopted. $\alpha_{\max} = 0.8$, and $\alpha_{\min} = 0.01$ have been found to work relatively well for this problem.

4. The entries in the Jacobian matrix are listed. Note that we can take advantage of the relatonships among elements of J when building the matrix in MATLAB:

$$\frac{\partial f_i}{\partial \beta_1} = \frac{1}{1+\beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3} \tag{16}$$

$$\frac{\partial f_i}{\partial \beta_2} = \frac{x_i}{1+\beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3} = \frac{\partial f_i}{\partial \beta_1} x_i \tag{17}$$

$$\frac{\partial f_i}{\partial \beta_3} = \frac{x_i^2}{1+\beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3} = \frac{\partial f_i}{\partial \beta_2} x_i \tag{18}$$

$$\frac{\partial f_i}{\partial \beta_4} = \frac{x_i^3}{1+\beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3} = \frac{\partial f_i}{\partial \beta_3} x_i \tag{19}$$

$$\frac{\partial f_i}{\partial \beta_5} = -\frac{\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3}{(1+\beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3)^2} x_i \tag{20}$$

$$\frac{\partial f_i}{\partial \beta_6} = -\frac{\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3}{(1+\beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3)^2} x_i^2 = \frac{\partial f_i}{\partial \beta_5} x_i \tag{21}$$

$$\frac{\partial f_i}{\partial \beta_7} = -\frac{\beta_1 + \beta_2 x_i + \beta_3 x_i^2 + \beta_4 x_i^3}{(1+\beta_5 x_i + \beta_6 x_i^2 + \beta_7 x_i^3)^2} x_i^3 = \frac{\partial f_i}{\partial \beta_6} x_i \tag{22}$$

5. In MATLAB, we can take advantage of the vectorization operations in calculating ϕ, $f-y$ and the Jacobian matrix, etc., without using explicit loops. For instance, the following code fragment is used to calculate the Jacobian matrix by using extensively the element-wise operators like "`./`", "`.*`", and "`.^`".

6. A summary of the results from `ps8thuber`: The data below respresent the result after about 50 iterations. Residial norm $\|J^T(f-y)\|$ is about 2.09×10^{-7}. Convergence history of $\|J^T(f-y)\|$ versus iteration number is plotted in Figure (1). We also plot the Thurber data and the nonlinear least squares fitted equation of Eq.(9) in Figure (2). The beta parameters are given here:

$$\begin{aligned}
\beta_1 &= 1.28814 \times 10^{3}, & \beta_5 &= 9.66295 \times 10^{-1},\\
\beta_2 &= 1.49108 \times 10^{3}, & \beta_6 &= 3.97973 \times 10^{-2},\\
\beta_3 &= 5.83238 \times 10^{2}, & \beta_7 &= 4.97273 \times 10^{-2},\\
\beta_4 &= 7.54166 \times 10^{1}.
\end{aligned}$$

As shown below, the relative error versus the reference values of β's are all very small. This is a validation of our successful implementation of the Gauss-Newton method in MATLAB:

```
Relative error vs ref values
1: 3.589732e-11   5: 4.756416e-12
2: 2.829348e-11   6: 9.795597e-12
3: 5.744589e-12   7: 3.096648e-12
4: 9.798426e-14
```

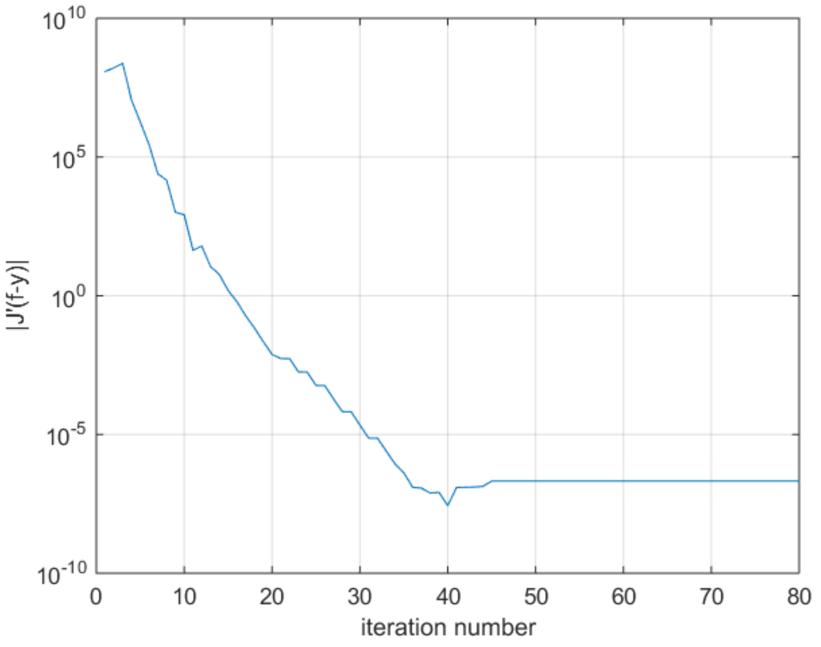


Figure 1: $\|J^T(f-y)\|$ versus iteration number in solving the nonlinear least squares problem.

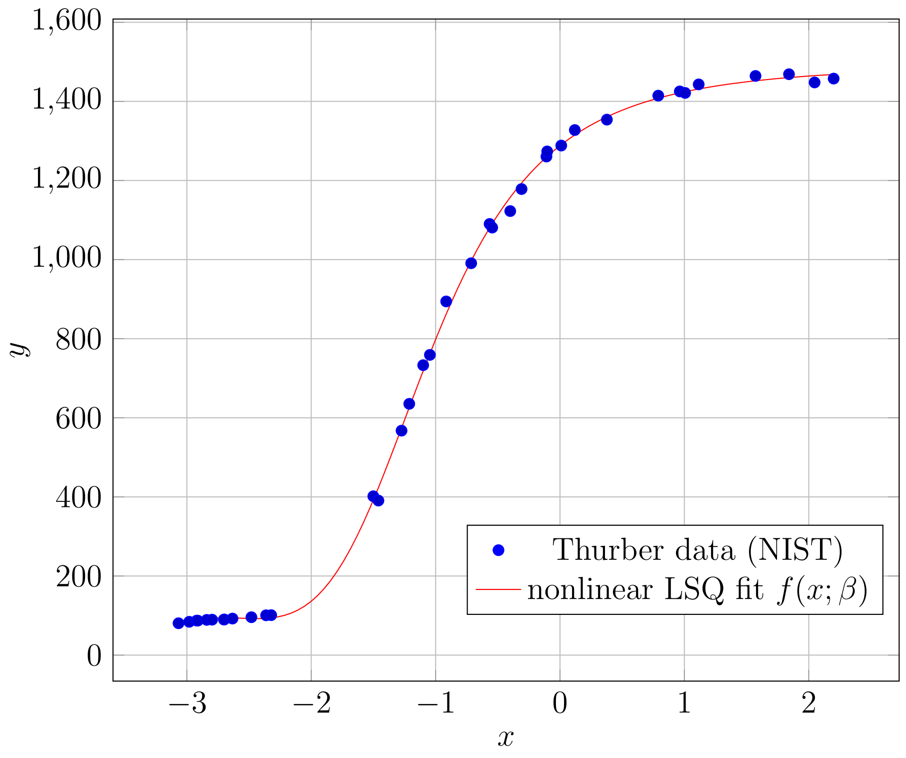


Figure 2: Thurber, R., data (NIST, 1979) and $f(x;\beta)$ of Eq.(9), which is fitted to the data points by the nonlinear least-squares method and solved by Gauss-Newton with a weak line-search strategy in Problem 2.

The code of nonlinear least squares for Problem 2 is listed below:

```
% Source:
%   http://www.itl.nist.gov/div898/strd/nls/data/thurber.shtml
%
xy = [80.574E0      −3.067E0
        84.248E0      −2.981E0
        87.264E0      −2.921E0
        87.195E0      −2.912E0
        89.076E0      −2.840E0
        89.608E0      −2.797E0
        89.868E0      −2.702E0
        90.101E0      −2.699E0
        92.405E0      −2.633E0
        95.854E0      −2.481E0
       100.696E0      −2.363E0
       101.060E0      −2.322E0
       401.672E0      −1.501E0
       390.724E0      −1.460E0
       567.534E0      −1.274E0
       635.316E0      −1.212E0
       733.054E0      −1.100E0
       759.087E0      −1.046E0
       894.206E0      −0.915E0
       990.785E0      −0.714E0
      1090.109E0      −0.566E0
      1080.914E0      −0.545E0
      1122.643E0      −0.400E0
      1178.351E0      −0.309E0
      1260.531E0      −0.109E0
      1273.514E0      −0.103E0
      1288.339E0       0.010E0
      1327.543E0       0.119E0
      1353.863E0       0.377E0
      1414.509E0       0.790E0
      1425.208E0       0.963E0
      1421.384E0       1.006E0
      1442.962E0       1.115E0
      1464.350E0       1.572E0
      1468.705E0       1.841E0
      1447.894E0       2.047E0
      1457.628E0       2.200E0];
```

```
% Data points
y = xy(:,1);
x = xy(:,2);

% Mesh for plotting f
xx =linspace(x(1), x(end));

% Initial values
b = [1000;
     1000;
     400;
     40;
     0.7;
     0.3;
     0.03];

% Certified values
bref = [1.2881396800E+03;
        1.4910792535E+03;
        5.8323836877E+02;
        7.5416644291E+01;
        9.6629502864E-01;
        3.9797285797E-01;
        4.9727297349E-02];

% Tolerance on norm(J'*r)
rtol = 1e-8;

% Record residual norms
resids = [];

% Evaluate initial residual
n = b(1) + (b(2) + (b(3) + b(4)*x).*x).*x;
d =    1 + (b(5) + (b(6) + b(7)*x).*x).*x;
f = n./d;
r = f-y;

% TODO: Iterate to find b s.t. norm(r)^2 is minimal
%       I recommend Gauss-Newton with line search;
%       you may prefer Levenberg-Marquardt or something fancier.
```

```
maxiter=80;
ncol=length(b); % b is the unknown vector of beta coefficient
fp={}; % fp is a cell array: array of vectors in the Jacobian matrix

%evaluate the initial Jacobian matrix
fp{1}= 1 ./ d;
fp{2}= fp{1}.*x;   %fp{2}=x ./ d;
fp{3}= fp{2}.*x;   %fp{3}= x.*x ./ d;
fp{4}= fp{3}.*x;   %x.^3 ./ d;

fp{5}=-n ./ d.^2 .*x;
fp{6}=fp{5}.*x;
fp{7}=fp{6}.*x;

J = [ fp{1} fp{2} fp{3} fp{4} fp{5} fp{6} fp{7} ];
alpha=0.5; % line search will provide the value of  alpha
% main Gauss-Newton iteration
for iter=1:maxiter
    % J and r are either from the intialization or updated after
    % solution (b) is updated at the (k+1) level later:
    %
    % checking for convergence here:
    resids(iter)=norm( J'*r );
    if (resids(iter) < rtol )
        break;
    end

    % Solving Gauss-Newton equation for search direction p:
    p = (J'*J)\(-J'*r);

    [phi]=phi_beta(b, x, y);

    grphi=J'*r;

    % solution b is updated via a line search:
    [bn,alpha]=lineSearch(b, p, x, y, phi, grphi);
    b=bn;
    fprintf('\n iter=%d, alpha=%g, res=%g\n', iter, alpha, resids(iter));
    %b = b + alpha*p;
    % update n, d and r with the new b (beta):
    n = b(1) + (b(2) + (b(3) + b(4)*x).*x).*x;
    d =    1 + (b(5) + (b(6) + b(7)*x).*x).*x;
    r = n./d - y;
    %
    % calculate the updated columns of the Jacobian matrix:
    % fp{i}=\partial f_i / \partial\beta:
    % fp{} is a cell array data:
    %
    % vectorization by the term-by-term operators
    fp{1}= 1 ./ d;
    fp{2}= fp{1}.*x;   %fp{2}=x ./ d;
    fp{3}= fp{2}.*x;   %fp{3}= x.*x ./ d;
    fp{4}= fp{3}.*x;   %x.^3 ./ d;

    fp{5}=-n ./ d.^2 .*x;
    fp{6}=fp{5}.*x;
    fp{7}=fp{6}.*x;

    %J = [ fp{1} fp{2} fp{3} fp{4} fp{5} fp{6} fp{7} ];
    for col=1:ncol
        J (:, col)=fp{col};
    end

end

% Check consistency with reference values
fprintf('Relative error vs ref values\n');
fprintf('%d: %e\n', [1:7; abs((bref-b)./bref )']);

% Plot results
figure(1);
nxx = b(1) + (b(2) + (b(3) + b(4)*xx).*xx).*xx;
dxx =    1 + (b(5) + (b(6) + b(7)*xx).*xx).*xx;
fxx = nxx./dxx;
plot(x, y, '.', xx, fxx);

% Plot convergence
figure(2);
semilogy(resids);
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
function [bn,alpha]=lineSearch(b, p, x, y, phi, grphi)
% [bn]=lineSearch(b, bnew, p, J, f, r)
%  A "weak line search" method for Gauss−Newton in nonlinear LSQ.
%
%  inputs:
%         b: beta at iteration (k)
%         bnew : beta at iteration (k+1)
%         p: Gauss−Newton search direction (descent)
%         phi: phi at k−level
%         grphi:  grad(phi) at k−level
%  output:
%         bn: beta at (k+1) level
%  (Taken from Ascher and Greif: page 264)
%
alphamax=0.8;
alphamin=0.01;
sigma=1E−3;

grpsi=p'*grphi;
alpha=alphamax;
%initially try max alpha
bn=b+alpha*p;
[phi_n]=phi_beta(bn,x,y);

while (phi_n > phi+sigma*alpha*grpsi)*(alpha > alphamin)
    mu=−0.5*grpsi*alpha/(phi_n−phi−alpha*grpsi);
    if mu < 0.1
        mu=0.5;
    end
    %try this new alpha
    alpha=mu*alpha;
    bn=b+alpha*p;
    [phi_n]=phi_beta(bn,x,y);
end

end
```

3: Descent directions In the context of unconstrained minimization of a function $\phi(x)$, suppose that H is symmetric and positive definite, and let $\tilde{p}$ be the solution to the system

$$H\tilde{p} = -\nabla\phi(x) + r$$

where r is a residual vector. Let $\kappa(H) = \lambda_{\max}(H)/\lambda_{\min}(H)$, show that if $\kappa(H)\|r\| < \|\nabla\phi\|$, then $\tilde{p}$ is a descent direction.

Hint: Note that $\lambda_{\min}(H)\|u\|\|v\| \le |u^T H v| \le \lambda_{\max}(H)\|u\|\|v\|$.

Proof.

1. H is symmetric positive definite (SPD), hence all eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of H are positive. Furthermore, there exists a unique decomposition of H such that
$$H = LL^T, \tag{23}$$
where L is a lower triangular matrix. Eq.(23) is the Cholesky decomposition of H. With the Cholesky decomposition, we can easily show H^{-1} is also SPD. First,
$$H^{-1} = (LL^T)^{-1} = (L^{-1})^T L^{-1},$$
hence H^{-1} is also symmetric. The eigenvalues of H^{-1} are $\lambda_1^{-1}, \ldots, \lambda_n^{-1}$, which are all positive as well. Therefore H^{-1} is SPD.

2. By definition, we know if
$$(\nabla\phi)^T\tilde{p} < 0, \tag{24}$$
then $\tilde{p}$ is a descent direction for the unconstrained minimization of ϕ. To solve $\tilde{p}$ in
$$H\tilde{p} = -\nabla\phi(x) + r, \tag{25}$$
we can formally write out the solution as
$$\tilde{p} = -H^{-1}\nabla\phi(x) + H^{-1}r. \tag{26}$$
Multiply (the inner product) Eq.(26) by $(\nabla\phi)^T$ to get
$$(\nabla\phi)^T\tilde{p} = -(\nabla\phi)^T H^{-1}\nabla\phi + (\nabla\phi)^T H^{-1} r. \tag{27}$$
The descent direction condition given in Eq.(24) becomes
$$\begin{aligned}(\nabla\phi)^T\tilde{p} = -(\nabla\phi)^T H^{-1}\nabla\phi \quad &+ \quad (\nabla\phi)^T H^{-1} r < 0,\\ (\nabla\phi)^T H^{-1}\nabla\phi \quad &> \quad (\nabla\phi)^T H^{-1} r. \end{aligned} \tag{28}$$

Because H^{-1} is SPD, $(\nabla\phi)^T H^{-1}\nabla\phi > 0$, Eq.(28) can re-arranged as

$$\frac{(\nabla\phi)^T H^{-1}\nabla\phi}{|(\nabla\phi)^T H^{-1} r|} > 1, \quad \text{so} \quad \left|\frac{(\nabla\phi)^T H^{-1}\nabla\phi}{(\nabla\phi)^T H^{-1} r}\right| > 1. \tag{29}$$

The inequality in Eq.(29) is the condition we need to prove in order that $\tilde{p}$ is a descent direction.

3. From the hint given in the problem statement, we construct the following two inequalities for $|uH^{-1}v|$ (its minimum and maximum eigenvalues are $1/\lambda_{\max}(H)$ and $1/\lambda_{\min}(H)$, respectively) by substituting $u = v = \nabla\phi$ to get:

$$\lambda_{\max}^{-1}\|\nabla\phi\|^2 < |(\nabla\phi)^T H^{-1}\nabla\phi| < \lambda_{\min}^{-1}\|\nabla\phi\|^2, \tag{30}$$

and $u = \nabla\phi$ and $v = r$ to get:

$$\lambda_{\max}^{-1}\|\nabla\phi\|\|r\| < |(\nabla\phi)^T H^{-1} r| < \lambda_{\min}^{-1}\|\nabla\phi\|\|r\|. \tag{31}$$

The bounds of the numerator and the denominator in the descent condition, as expressed in Eq.(29), are given in Eq.(30) and Eq.(31), so the bounds of

$$\left|\frac{(\nabla\phi)^T H^{-1}\nabla\phi}{(\nabla\phi)^T H^{-1} r}\right| \tag{32}$$

can now be carefully established. (The uppper bound of Eq.(30) over the lower bound of Eq.(31) is the upper bound of Eq.(32).) The result is

$$\frac{\lambda_{\min}\|\nabla\phi\|^2}{\lambda_{\max}\|\nabla\phi\|\|r\|} < \left|\frac{(\nabla\phi)^T H^{-1}\nabla\phi}{(\nabla\phi)^T H^{-1} r}\right| < \frac{\lambda_{\max}\|\nabla\phi\|^2}{\lambda_{\min}\|\nabla\phi\|\|r\|}. \tag{33}$$

Comparing the bounds in the above inequality with the descent direction condition in Eq.(29), it is obvious that the lower bound in Eq.(33) is required to be larger than 1 if $\tilde{p}$ is a descent direction, i.e.,

$$\frac{\lambda_{\min}\|\nabla\phi\|^2}{\lambda_{\max}\|\nabla\phi\|\|r\|} > 1. \tag{34}$$

By introducing $\kappa(H) = \lambda_{\max}(H)/\lambda_{\min}(H)$ (the condition number of H), Eq.(34) can be simplified as

$$\frac{\lambda_{\min}\|\nabla\phi\|^2}{\lambda_{\max}\|\nabla\phi\|\|r\|} = \frac{\|\nabla\phi\|}{\kappa(H)\|r\|} > 1, \tag{35}$$

therefore we have got the proof:

$$\kappa(H)\|r\| < \|\nabla\phi\|. \tag{36}$$

□