

PROJECT 9 SOLUTIONS

Math 441

All “by hand” calculations require that you SHOW YOUR WORK. All MATLAB code and output must be turned in.

1. Sensitivity of the Eigenvalue Problem:

- (b) Using the Bauer-Fike bound, we get $|\delta\lambda| \leq \kappa_2(W)\|\delta A\|_2 = 81.610^{-8} = 8.16 \times 10^{-7}$.
- (d) Plugging in the known factors of the Bauer-Fike bound $|\delta\lambda| \leq \kappa_2(W)\|\delta A\|_2$ we get that $\|\delta A\|_2 \geq \frac{2.09 \times 10^{-4}}{81.6} = 2.57 \times 10^{-7}$. Here's the relevant MATLAB code:

```
>> eighat=QRalg(A)
>> [W eigtrue]=eig(A)
>> [eighat eigtrue]

ans =

    -1.8637    -1.8637
    -1.6372    -1.6372
    -0.7897    -0.7899
    -0.6343    -0.6343
    -0.4383    -0.4383
     0.3386     0.3386
     0.5668     0.5668
     0.5692     0.5692
     0.7405     0.7407
     1.4778     1.4778

>> norm(eighat-eigtrue,inf)
    2.0930e-004
>> cond(W)
    81.5945

>> type QRalg
function evals=QRalg(A,maxiter)

% This program implements the QR Algorithm for finding eigenvalues

[m n]=size(A);

if m~=n
    error('A must be square')
end

if ~exist('maxiter','var')
    maxiter=10*n;
end

tic
A=hess(A);

for i=1:maxiter
    [Q R]=qr(A);
    A=R*Q;          % Note that this means that Anew = Q'QRQ=Q'AQ
```

```

if ~mod(i,n) | i==maxiter
    %imagesc(A)
    %colorbar
    spy(abs(A)>sqrt(eps))
    title(sprintf('Check for convergence at iter=%d: Is this matrix either upper-triangu
    fprintf('\nCheck for convergence in figure(1); Hit any key to continue\n')
    pause
end
end

evals=diag(A);
toc

```

2. Lanczos Eigensolving:

```

(a) >> [T Q]=Lanczos(A,1,12);
>> full(T)
Columns 1 through 6

```

4.3098	14.0370	0	0	0	0
14.0370	-4.9776	15.4731	0	0	0
0	15.4731	4.3100	29.4868	0	0
0	0	29.4868	-4.9763	30.9310	0
0	0	0	30.9310	4.3089	44.9305
0	0	0	0	44.9305	-4.9751
0	0	0	0	0	46.3791
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

```

Columns 7 through 12

```

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
46.3791	0	0	0	0	0
4.3078	60.3680	0	0	0	0
60.3680	-4.9740	61.8195	0	0	0
0	61.8195	4.3068	75.8002	0	0
0	0	75.8002	-4.9731	77.2539	0
0	0	0	77.2539	4.3059	91.2275
0	0	0	0	91.2275	-4.9722

- (d) The 11 largest eigenvalues of A are the ones being estimated because they are the ones that are well separated. The other 88 eigenvalues of A are close to .001, and T has one eigenvalue near .001. Some MATLAB code:

```

>> eigA=eig(A);
>> [eigA(89:end) eig(T)]
0.0048    0.0010
0.0169    0.0204
0.0614    0.0616

```

0.2115	0.2115
0.6671	0.6671
1.9003	1.9003
4.8371	4.8371
10.8922	10.8922
21.4861	21.4861
36.8020	36.8020
54.3375	54.3375
68.7940	68.7940

- (e) After 12 Lanczos iterations, the instability of the algorithm is clear: Lanczos vectors q_i lose orthogonality with the previous vectors (you can check $Q^T T_{15} Q$ for the case where T_{15} is 15×15). Two of the eigenvalues of T_{15} are repeated estimates of the largest eigenvalues $\lambda_{99} = 54.3375$ and $\lambda_{100} = 68.7940$ of A , instead of estimating the clustered eigenvalues of A near .001. This is called the “ghost eigenvalue problem.”

3. Use the Lanczos Eigensolver (**eigs**) instead of the QR algorithm (**eig**) to estimate the eigenvalues of A when A is large (and possibly sparse) and symmetric, or if only a small number of eigenvalues are desired.

4. **Natural Logs:** Consider estimating $\int_1^R \frac{1}{x} dx$.

- (a) These results can be checked in any calculus text book and by running **[integ weights nodes]=numint('recipfunc',1,2,3,method)** where method is either 1,2,3 or 4 (see **numint.m**. The code for **recipfunc.m** is given in 4(d) below. For right-hand Riemann sums, the weight of $\Delta x = \frac{1}{3}$ is the same at each of the nodes $\{1\frac{1}{3} \ 1\frac{2}{3} \ 2\}$. The Trapezoid Rule and Simpsons Rule have nodes at $\{1 \ 1\frac{1}{2} \ 2\}$. The weights for Trapezoid Rule are $\frac{\Delta x}{2}[1 \ 2 \ 1] = [\frac{1}{4} \ \frac{1}{2} \ \frac{1}{4}]$. The weights for Simpsons Rule are $\frac{\Delta x}{6}[1 \ 4 \ 1] = [\frac{1}{6} \ \frac{2}{3} \ \frac{1}{6}]$.
- (b) The Lanczos tridiagonal matrix for $R = 2$ and 3 nodes is

$$T = \begin{pmatrix} 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{3}} & 0 & \frac{2}{\sqrt{15}} \\ 0 & \frac{2}{\sqrt{15}} & 0 \end{pmatrix} \approx \begin{pmatrix} 0 & 0.5774 & 0 \\ 0.5774 & 0 & 0.5164 \\ 0 & 0.5164 & 0 \end{pmatrix}.$$

This was found by changing the line **T=spdiags([[dt 0]' [0 dt]'],[-1 1],n,n);** in **numint.m** to **T=full(spdiags([[dt 0]' [0 dt]'],[-1 1],n,n))**

- (c) For Gaussian Quadrature, the nodes are the (linear functions) of the eigenvalues of the tridiagonal matrix T in 4b. The weights are (linear functions) **2*W(1,:).^2**, two times the square of the first component of each eigenvector. For $R = 2$ and 3 nodes, the weights are [0.2778 0.4444 0.2778] respectively at each of the nodes {1.1127 1.5 1.8873}.
- (d) An m-file, called **recipfunc.m**, which evaluates $\frac{1}{x}$ for any given x is
- ```
function f=recipfunc(x);
f=1./x;
```
- (e)–(g) Gaussian Quadrature is most accurate with an absolute difference from  $\ln 2$  on the order of  $10^{-16}$  (Riemann sums  $10^{-3}$ , Trapezoid  $10^{-6}$ , Simpsons  $10^{-10}$  - recall the theory: Gauss Quadrature is guaranteed to find  $\min_{w_i, x_i} |\int_a^b f(x) - \sum_{i=1}^n w_i f(x_i)|$  if  $f$  is (estimated well by) a polynomial of order  $2n - 1$  or less). Gauss Quadrature is also most expensive since it is solving an eigenvalue problem for a matrix of size  $n$ , which costs about  $\frac{10}{3}n^3$  flops, whereas the other three methods cost only about  $2n^2$  flops to evaluate.

5. **Square Roots:** (known by Heron ca. 250-150BC, about 200 years after Pythagorus) Use Newton's Method to find the zeros of  $f(\beta) = \beta^2 - R$ .

- (a) The mfile **sqrtfunc.m** is

```
function [f g]=sqrtfunc(x,r);

if ~exist('r','var')
 r=2;
end

f=x.^2 - r;
g=2*x;
```

- (b) The Newton step is  $\beta_{k+1} = \beta_k - \frac{1}{2\beta_k} (\beta_k^2 - R) = \frac{\beta_k^2 + R}{2\beta_k}$ .
- (c) Initiating Newton's Method at  $\beta_0 = 2$ , we estimate that the root  $\sqrt{2}$  of  $f(\beta)$  is approximately 1.4142 after 6 iterations. In MATLAB, `newton('sqrtfunc',2)` yields:

```
>> newton('sqrtfunc',2)
iter=0 |f(x_0)|=2.000e+000 |e|=0.000e+000
iter=1 ||x_k||=1.500000 ||f(x_k)||=2.500e-001 ||step||=5.000e-001
iter=2 ||x_k||=1.416667 ||f(x_k)||=6.944e-003 ||step||=8.333e-002
iter=3 ||x_k||=1.414216 ||f(x_k)||=6.007e-006 ||step||=2.451e-003
iter=4 ||x_k||=1.414214 ||f(x_k)||=4.511e-012 ||step||=2.124e-006
iter=5 ||x_k||=1.414214 ||f(x_k)||=4.441e-016 ||step||=1.595e-012
iter=6 ||x_k||=1.414214 ||f(x_k)||=4.441e-016 ||step||=1.570e-016
 error stopping tolerance met.

ans =

 1.4142
```

- (d) Since  $f'(\beta_0) < 0$  for all  $\beta_0 < 0$ , then the roots of each linear approximation to  $f$  will be negative. Thus, when initiated with negative  $\beta_0$ , Newton's method will never find the positive root of  $f$ .
- (e)  $\beta = 0$  is the minimum of  $f$ , and so  $f'(0) = 0$ . Thus, the Newton step in 5b is undefined.
- (f) The absolute error  $|\hat{\beta} - \sqrt{2}|$  was calculated in MATLAB via

```
>> betahat=newton('sqrtfunc',2)
 1.4142

>> abs(betahat-sqrt(2))
 2.2204e-016
```

Since this value is on the order of machine precision (for double precision arithmetic), then applying Newton's Method to `sqrtfunc.m` is comparable to using MATLAB's `sqrt`.

6. **Optimization:** Use Newton's method to minimize the scalar function  $f \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \beta_1 \ln(\beta_2)$ .

- (a) The  $2 \times 1$  gradient is  $\nabla_{\beta} f = \begin{pmatrix} \ln(\beta_2) \\ \frac{\beta_1}{\beta_2} \end{pmatrix}$ . The  $2 \times 2$  Hessian matrix of second derivatives is

$$d_{\beta}^2 f = \begin{pmatrix} 0 & \frac{1}{\beta_2} \\ \frac{1}{\beta_2} & -\frac{\beta_1}{\beta_2^2} \end{pmatrix}.$$

- (b) The code which performs the function, gradient and Hessian evaluations is `quadlnfunc.m`:

```
function [f g H]=quadlnfunc(x);

f=x(1)*log(x(2));
g=[log(x(2)); x(1)/x(2)];
H=[0 1/x(2); 1/x(2) -x(1)/x(2)^2];
```

(c) The Newton step is

$$\begin{aligned}
 \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}_{k+1} &= \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}_k - \begin{pmatrix} 0 & \frac{1}{\beta_2} \\ \frac{1}{\beta_2} & -\frac{\beta_1}{\beta_2^2} \end{pmatrix}_k^{-1} \begin{pmatrix} \ln(\beta_2) \\ \frac{\beta_1}{\beta_2} \end{pmatrix}_k \\
 &= \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}_k - \beta_2^2 \begin{pmatrix} \frac{\beta_1}{\beta_2} & -\frac{1}{\beta_2} \\ -\frac{1}{\beta_2} & -\frac{\beta_1}{\beta_2^2} \end{pmatrix}_k \begin{pmatrix} \ln(\beta_2) \\ \frac{\beta_1}{\beta_2} \end{pmatrix}_k \\
 &= \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}_k - \begin{pmatrix} \beta_1 & \beta_2 \\ \beta_2 & 0 \end{pmatrix}_k \begin{pmatrix} \ln(\beta_2) \\ \frac{\beta_1}{\beta_2} \end{pmatrix}_k \\
 &= \begin{pmatrix} \beta_1 \ln \beta_2 \\ \beta_2 (1 - \ln \beta_2) \end{pmatrix}_k.
 \end{aligned}$$

(d) `NewtonForRosenbrock.m` was modified (and saved as `NewtonForQuadLog.m`) so that the top two lines are

```
cost_func='quadlnfunc';
x=[1.5,1.5]';
```

(e) After 6 iterations, Newton's method converges to  $\hat{\beta} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ . Here's the MATLAB output:

```
>> NewtonForQuadLog
iter=2 f=6.96452e-002 |g|=6.91534e-001 x=[-0.608198 0.891802]
iter=3 f=4.24504e-004 |g|=7.03356e-002 x=[-0.069645 0.993923]
iter=4 f=7.85365e-009 |g|=4.24914e-004 x=[-0.000425 0.999981]
iter=5 f=1.34405e-018 |g|=7.85551e-009 x=[-0.000000 1.000000]
iter=6 f=0.00000e+000 |g|=1.34405e-018 x=[-0.000000 1.000000]
Gradient tolerance met. Stop iterations
```

(f) From 6a,  $d_{\beta}^2 f \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$  which has eigenvalues -1 and 1. Since  $d_{\beta}^2 f \begin{pmatrix} 0 \\ 1 \end{pmatrix}$  is not positive definite, then it is not a minimum (rather, it's a saddle).

(g) When starting Newton's Method from  $\beta_0 = \begin{pmatrix} 1.5 \\ 5 \end{pmatrix}$ , the first few iterations are

```
iter=2 f=-2.68990e+000 |g|=3.42619e+000 x=[-2.414157 -3.047190]
iter=3 f=-5.55964e+000 |g|=2.85764e+000 x=[2.689901 0.348049]
iter=4 f=9.85791e-001 |g|=3.24571e+000 x=[5.559636 14.250976]
iter=5 f=-1.41053e+002 |g|=4.54017e+000 x=[-0.985791 -18.610195]
iter=6 f=5.87302e+002 |g|=5.34406e+000 x=[141.053113 121.562756]
iter=7 f=-6.93386e+003 |g|=7.27714e+000 x=[-587.301580 -450.040539]
iter=8 f=5.52181e+004 |g|=8.58878e+000 x=[6933.861295 3056.991249]
```

and eventually the program crashes. The oscillating signs of  $f(\beta_k)$  and of  $\beta_k$  show that Newton's Method fails to converge in this case. This is due to the fact that  $f$  has no local minima (the only stationary point is at  $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ ), is only defined for  $\beta_2 > 0$ , and decreases without bound.

7. **Non-linear least squares:** Use the Gauss-Newton Method to fit a logistic model to the number of cases of AIDS in the US from 1981 to 1992. The data is available from the course web site, where the first column of the data matrix is in years,  $t = 1$  corresponds to 1981, and  $t = 12$  corresponds to 1992. The second column is the number of new AIDS cases each year, in the thousands. The least squares problem you will solve is  $\min_{\beta} f(\beta) = \min_{\beta} \|y - \phi(\beta)\|_2^2$  where

$$\phi(\beta) = \frac{\beta_1 \beta_2}{\beta_1 + (\beta_2 - \beta_1) e^{-\beta_3 t}}.$$

Note that  $\beta_3$  is the exponential rate of change of  $f$  from  $\beta_1$  to  $\beta_2$ .

- (a) By definition,  $t = 0$  corresponds to the year 1980, and  $\phi(0) = \beta_1$ , so  $\beta_1$  is interpreted as the predicted number of AIDS cases in 1980. And,

$$\lim_{t \rightarrow \infty} \phi(\beta, t) = \frac{\beta_1 \beta_2}{\beta_1 + (\beta_2 - \beta_1) \lim_{t \rightarrow \infty} e^{-\beta_3 t}} = \frac{\beta_1 \beta_2}{\beta_1} = \beta_2$$

shows that  $\beta_2$  is the asymptotic number of AIDS cases.

- (b) The m-file **GNlogistic.m** accepts a  $3 \times 1$  vector of  $\beta$  parameters and a  $n \times 1$  vector of times  $t$ , and outputs the value of  $\phi(\beta, t)$ :

```
function f=GNlogisticfunc(beta,t)

% logistic function

P=beta(1); % Intial value
M=beta(2); % Asymptotic
k=beta(3); % rate

% If P<M and k>0, then increasing logistic function from P to M
% If P<M and k<0, then decreasing logistic function from M to P
% If P>M, then there is an asymptote, and the function looks like 1/x

f=M*P./(P+(M-P)*(exp(-k*t)));
%f=M*P./(M+P*(exp(-k*t))-1); % unstable
```

- (c) In general, the Gauss-Newton step is  $\beta_{k+1} = \beta_k - (J_k^T J_k)^{-1} J_k^T (y - \phi(\beta_k))$ , where  $J_k$  is the Jacobian evaluated at  $\beta_k$ ,  $J_k = -d_\beta \phi(\beta_k)$ . In the case of the AIDS data,  $J_k$  is  $12 \times 3$  since

$$d_\beta \phi = \begin{pmatrix} \frac{\partial \phi}{\partial \beta_1} & \frac{\partial \phi}{\partial \beta_2} & \frac{\partial \phi}{\partial \beta_3} \end{pmatrix}$$

where each  $\frac{\partial \phi}{\partial \beta_i}$  is  $12 \times 1$  (since  $t$  is a  $12 \times 1$  vector). If any of you computed any of these, they are

$$\frac{\partial \phi}{\partial \beta_1} = \frac{\beta_2(D - \beta_1(1 - e^{-\beta_3 t}))}{D^2}$$

$$\frac{\partial \phi}{\partial \beta_2} = \frac{\beta_1(D - \beta_2 e^{-\beta_3 t})}{D^2}$$

$$\frac{\partial \phi}{\partial \beta_3} = \frac{t \beta_1 \beta_2 (\beta_2 - \beta_1) e^{-\beta_3 t}}{D^2}$$

where  $D = \beta_1 + (\beta_2 - \beta_1) e^{-\beta_3 t}$  is the denominator of  $\phi$ .

- (d) The first few lines of **GaussNewtonExample.m** were changed and saved off as **GaussNewtonAids.m**:

```
% Get Data
load aids.txt;
t=aids(:,1); % t=0 is 1980
y=aids(:,2); % y is in thousands

% Function to Fit
phi='GNlogisticfunc';
```

(e) The MATLAB output:

```
>> GaussNewtonAids
Estimates:
beta_1=0.326538 SE=0.090929 95percent CI: [0.120842,0.532234]
beta_2=49.513085 SE=1.259392 95percent CI: [46.664143,52.362027]
beta_3=0.681534 SE=0.042122 95percent CI: [0.586248,0.776819]
```

Thus, the Gauss-Newton estimate for the logistic function fit to the AIDS data is

$$\phi(t) = \frac{16.18}{.3265 + 49.19e^{-.6815t}}$$

- (f) The statistics do suggest that the number of AIDS cases in 1980 was larger than 100 since the 95% confidence interval for  $\beta_1$  predicts that the number of cases of AIDS in 1980 is between 120 and 532 cases, both of which are more than 100.
- (g) The model predicts that the number of cases of AIDS in 1993 was less than 100,000. This is because the 95% prediction interval band depicted in the figure for  $t = 13$  (1993) suggests that the number of AIDS cases in 1993 is between 46 and 51 thousand, both of which are less than 100,000.
- (h) Extrapolation is always a dicey proposition. The tight fit of the model to the data throughout the years 1981 to 1992 suggest that this model is very good for these years. Outside the range of data to which a model is fit, no model should be trusted. This model IS a bad model of the number of AIDS cases in 1993.