Sample Solutions for Practice Problems for Final

Final will cover chapters 1 through 6.4, with special emphasis on material since the midterm (i.e., chs. 5 and sections 1 through 4 of chapter 6). This sheet contains practice problems only for chs. 5 and 6 (except the first problem, which is a followup from the midterm). Refer to earlier practice sheet for problems from chs. 1–4.

1. **(a)** On the midterm we considered the problem of evaluating $f(x) = 1 - \sqrt{1-x}$ and showed that while the problem was well-conditioned for x near 0, one would get an inaccurate result by applying this formula in a straightforward way. For example, if $|x| < 10^{-16}$, then $1 - x$ would be rounded to 1, taking $\sqrt{1}$ would give 1, and then subtracting this from 1 would give an answer of 0, which does not have high relative accuracy. Write down an algorithm that will give high relative accuracy. [Hint: One possible approach: You can evaluate $1 + \sqrt{1-x}$ to high relative accuracy. We have the product formula $(1 - \sqrt{1-x})(1 + \sqrt{1-x}) = x$.]

   The problem with evaluating $1 - \sqrt{1-x}$ when x is near 0 is that we are subtracting two nearby numbers, one of which will likely be off by about $1.e-16$. If the result is on the order of $1.e-16$ or less, then we cannot expect any relative accuracy in the computed result. On the other hand, if we evaluate $1 + \sqrt{1-x}$ when x is near 0, then the true result is close to 2, so if we are off by $1.e-16$ then that does represent a small relative error. We can then use the formula $1 - \sqrt{1-x} = x/(1+\sqrt{x})$ to evaluate $1 - \sqrt{1-x}$ accurately.

   (b) Suppose we wish to solve the quadratic equation

   $$x^2 + bx + c = 0$$

   for x, where $b > 0$ and $|c| << b^2$. The formulas for the two roots are

   $$x_+ = \frac{-b + \sqrt{b^2 - 4c}}{2}, \quad x_- = \frac{-b - \sqrt{b^2 - 4c}}{2}$$

   Write down an algorithm that can be used to evaluate both x_+ and x_- to high relative accuracy.

   Since $-b$ is negative, the two terms in the numerator of x_- have the same sign and so they can be added safely. Therefore, compute x_- using the above formula. Note that $x_+ x_- = c$, and use the formula $x_+ = c/x_-$ to compute x_+.

2. The following fragment of MATLAB code does Gaussian elimination without pivoting on an n by n matrix A:

```
for k=1:n-1,              % Use row k to eliminate entries in column k
                          % of rows k+1 through n.
                          %
                          %   Here you should build in partial pivoting
                          %
  for i=k+1:n,
    mult = A(i,k)/A(k,k);           % Subtract mult times row k from row i
    for j=k:n,                      % in order to zero out A(i,k)
      A(i,j) = A(i,j) - mult*A(k,j);
    end;
  end;
end;
```

(a) Write down the code you would insert to implement partial pivoting. (If you are not sure about the MATLAB commands, you may write your code in C or in some pseudo-MATLAB form, as long as it is clear *exactly* what you are doing.)

Insert this code just before the loop over i:

```
[pivot,index] = max(abs(A(k:n,k)));  % Find largest entry in column.

%  Interchange row k and row index+k-1.  [Remember that "index" is
%  the index past row k of the pivot row.]

temp = A(k,:);             % Temporarily store row k.
A(k,:) = A(index+k-1,:);   % Replace row k by pivot row.
A(index+k-1,:) = temp;     % Move what was row k into the position
                           % from which the pivot row came.
```

(b) Suppose A is *tridiagonal* and pivoting is not required. Show how you could modify the above code to solve this problem efficiently, and count the number of operations performed in your modified code.

```
for k=1:n-1,              % Use row k to eliminate entries in column k
                          % of row k+1.  [Entries in col k of rows k+2
                          % through n are already 0.]
  for i=k+1:k+1,
    mult = A(i,k)/A(k,k);       % Subtract mult times row k from row i
                                % in order to zero out A(i,k).  Only
    for j=k:k+1,                % entries in cols k and k+1 are affected.
      A(i,j) = A(i,j) - mult*A(k,j);
    end;
  end;
end;
```

The amount of work required is:

$$\sum_{k=1}^{n-1}\left(1 \text{ div } + \sum_{j=k}^{k+1}(1 \text{ sub } + 1\text{mult})\right) = 5(n-1) \text{ ops .}$$

3. **(a)** Compute the 2-norm and the ∞-norm of the vector:

$$\begin{pmatrix} 1 \\ -2 \end{pmatrix}.$$

The 2-norm is $\sqrt{1^2 + (-2)^2} = \sqrt{5}$. The ∞-norm is $\max\{|1|, |-2|\} = 2$.

(b) What is the ∞-norm ($\max_{\|v\|_\infty = 1} \|Av\|_\infty$) of the matrix

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 6.1 \end{pmatrix}?$$

The ∞-norm of A is the maximum absolute row sum: $\max\{|1|+|2|, |3|+|6.1|\} = 9.1$.

(c) Determine the *condition number* of A in the ∞-norm?

The condition number of A in the ∞-norm is $\|A\|_\infty \cdot \|A^{-1}\|_\infty$. Since

$$A^{-1} = 10 \begin{pmatrix} 6.1 & -2 \\ -3 & 1 \end{pmatrix},$$

$\|A^{-1}\|_\infty = 10 \max\{|6.1| + |-2|, |-3| + |1|\} = 81$. Hence $\kappa_\infty(A) = 9.1 \times 81 = 737.1$.

4. **(a)** Let x be the exact solution to the linear system $Ax = b$, and let $\hat{x}$ be the exact solution to the linear system $A\hat{x} = \hat{b}$, where A is a nonsingular matrix. Derive a bound on the relative error, $\|\hat{x} - x\|/\|x\|$, in terms of the relative change in b, $\|\hat{b} - b\|/\|b\|$.

Subtracting the two equations, we find that $A(x - \hat{x}) = b - \hat{b}$, or, $x - \hat{x} = A^{-1}(b - \hat{b})$. Taking norms on each side gives $\|x - \hat{x}\| \leq \|A^{-1}\| \, \|b - \hat{b}\|$. Dividing each side by $\|x\|$ (and multiplying and dividing by $\|b\|$ on the right) gives

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \|A^{-1}\| \, \frac{\|b - \hat{b}\|}{\|b\|} \, \frac{\|b\|}{\|x\|}.$$

Since $Ax = b$, we can write $\|b\| \leq \|A\| \cdot \|x\|$, or, $\|b\|/\|x\| \leq \|A\|$, and making this substitution above we find

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \|A^{-1}\| \, \|A\| \frac{\|b - \hat{b}\|}{\|b\|}.$$

Thus the normwise relative change in x is bounded by the condition number of A, $\kappa(A) \equiv \|A\| \cdot \|A^{-1}\|$, times the normwise relative change in b.

(b) What does it mean for an algorithm to be *backward stable*? If a backward stable algorithm is used to solve a linear system $Ax = b$ on a machine with unit roundoff ϵ, approximately how large will the relative error, $\|\hat{x} - x\|/\|x\|$, be?

An algorithm is *backward stable* if the computed solution using that algorithm is the exact solution to a nearby problem. For example, if one has an algorithm for solving linear systems $Ax = b$ and the computed solution $\hat{x}$ always satisfies $(A+E)\hat{x} = \hat{b}$, for some matrix E with $\|E\|/\|A\|$ on the order of machine precision and for some vector $\hat{b}$ with $\|b-\hat{b}\|/\|b\|$ on the order of machine precision, then the algorithm is backward stable. If the matrix A is ill-conditioned, one cannot guarantee a small relative error, even if the algorithm is backward stable; if A or b have been rounded, then the exact solution to the problem with rounded A and b might be quite different from that of the problem with exact A and b. With a good algorithm, however, one should expect

$$\frac{\|x - \hat{x}\|}{\|x\|} \approx \kappa(A)\epsilon,$$

where $\kappa(A) \equiv \|A\| \cdot \|A^{-1}\|$ is the condition number of A and ϵ is the machine precision.

5. Factor the following matrix in the form QR, where Q is a 3 by 2 matrix with orthonormal columns and R is a 2 by 2 upper triangular matrix:

$$A = \begin{pmatrix} 0 & -4 \\ 0 & 0 \\ -5 & -2 \end{pmatrix}.$$

Use your QR factorization to solve the least squares problem $Ax \approx b$, where

$$b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

$$q_1 = A(:,1)/\|A(:,1)\| = A(:,1)/5 = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}.$$

$$\tilde{q}_2 = A(:,2) - \langle A(:,2), q_1\rangle q_1 = \begin{pmatrix} -4 \\ 0 \\ -2 \end{pmatrix} - 2\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} -4 \\ 0 \\ 0 \end{pmatrix}.$$

$$q_2 = \tilde{q}_2/\|\tilde{q}_2\| = \tilde{q}_2/4 = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}.$$

It follows that $A = QR$, where

$$Q = (q_1, q_2) = \begin{pmatrix} 0 & -1 \\ 0 & 0 \\ -1 & 0 \end{pmatrix}, \quad R = \begin{pmatrix} 5 & 2 \\ 0 & 4 \end{pmatrix}.$$

To use this to solve the least squares problem, first compute $Q^T b$ and then solve $Rx = Q^T b$:

$$Q^T b = \begin{pmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} -3 \\ -1 \end{pmatrix},$$

$$\begin{pmatrix} 5 & 2 \\ 0 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -3 \\ -1 \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1/2 \\ -1/4 \end{pmatrix}.$$

6. Consider the following set of data:

x	y
1	1
2	2
3	4

(a) Find the straight line that best fits this data in a least squares sense. Show how you obtained your answer. Also plot the data points and the straight line that you computed.

We want c_0 and c_1 such that $y \approx c_0 + c_1 x$:

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} \approx \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}.$$

Forming the normal equations:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix},$$

$$\begin{pmatrix} 3 & 6 \\ 6 & 14 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 7 \\ 17 \end{pmatrix}.$$

Solving, we find $c_0 = -2/3$, $c_1 = 3/2$.

(b) Write down the Lagrange form of a quadratic polynomial that exactly fits the data.

$$p(x) = 1 \cdot \frac{(x-2)(x-3)}{(1-2)(1-3)} + 2 \cdot \frac{(x-1)(x-3)}{(2-1)(2-3)} + 4 \cdot \frac{(x-1)(x-2)}{(3-1)(3-2)}.$$

(c) Write down the Newton form of a quadratic polynomial that exactly fits the data.

The Newton form is:

$$p(x) = a_0 + a_1(x-1) + a_2(x-1)(x-2).$$

To find the coefficients:

$$\begin{aligned} p(1) = 1 &\Rightarrow a_0 = 1 \\ p(2) = 2 &\Rightarrow a_0 + a_1 = 2 \Rightarrow a_1 = 1 \\ p(3) = 4 &\Rightarrow a_0 + 2a_1 + 2a_2 = 4 \Rightarrow a_2 = 1/2. \end{aligned}$$

(d) Assume that this data comes from a function f whose derivatives are all continuous. Write an expression for the difference between $f(x)$ and your quadratic interpolant at an arbitrary point x. Suppose that all of the derivatives of f are bounded in absolute value by 3; that is, $|f^{(n)}(x)| \leq 3$ for all x and for $n = 1, 2, \ldots$. Give a bound on the difference between $f(1.5)$ and the value of your quadratic at $x = 1.5$.

$$f(x) - p(x) = \frac{f'''(\xi)}{3!}(x-1)(x-2)(x-3), \quad \xi \in [1,3].$$

If $|f'''(x)| \leq 3$ for all $x \in [1,3]$, then

$$|f(1.5) - p(1.5)| \leq \frac{3}{3!}(.5)(.5)(1.5) = \frac{3}{16}.$$

7. Compare the efficiency of the divided difference algorithm for computing the coefficients in the nth degree Newton interpolant of a function f to that of solving a triangular system for these coefficients.

In the divided difference algorithm, one computes a table of divided differences:

$f[x_1] = f_1$			
$f[x_2] = f_2$	$f[x_1, x_2] = (f[x_2] - f[x_1])/(x_2 - x_1)$		
$f[x_3] = f_3$	$f[x_2, x_3] = (f[x_3] - f[x_2])/(x_3 - x_2)$	$f[x_1, x_2, x_3] = (f[x_2, x_3] - f[x_1, x_2])/(x_3 - x$	

There are $(n+1)(n+2)/2$ entries in this table. The first column is given, but each of the other $n(n+1)/2$ entries requires 3 operations to compute. Hence the total work is about $3n(n+1)/2 = (3/2)n^2 + O(n)$ operations.

To solve a triangular system for the coefficients, one forms the $n+1$ by $n+1$ triangular matrix:

$$\begin{pmatrix} 1 & & & & \\ 1 & x_2 - x_1 & & & \\ 1 & x_3 - x_1 & (x_3 - x_1)(x_3 - x_2) & & \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & x_{n+1} - x_1 & (x_{n+1} - x_1)(x_{n+1} - x_2) & \ldots & \prod_{j=1}^{n}(x_{n+1} - x_j) \end{pmatrix}.$$

There is no work in forming the first column. Each of the n nonzero entries in the second column requires 1 subtraction. Successive columns can be formed by multiplying entries of the previous column by one new factor. This requires 1 subtraction and 1 multiplication for each nonzero entry in the column. Therefore the total work to form the matrix is:

$$n + 2[(n-1) + (n-2) + \ldots + 1] = n^2.$$

The time to backsolve an $n+1$ by $n+1$ triangular system is about $(n+1)^2$, so the total work is about $2n^2 + O(n)$.

Both methods are $O(n^2)$, but the constant seems to be a little smaller when forming the divided difference table: 3/2 instead of 2. [There might be some trick that would reduce the constant in the triangular solve, but both methods will be $O(n^2)$. The real advantage of the divided difference table over the triangular system is that you do not run into problems of overflow and underflow that may occur when you form the long products in the bottom right entries of the triangular matrix.]