Supplementary Notes to Math 551-Numerical Methods, Spring 2013

Tom DeLillo, WSU Math Dept.

May 9, 2013

1 Introductory remarks-Lecture 1/22/13

I will try to maintain here a record of my lectures, with varying levels of detail, as a supplement to our text [CM]. This is not meant to replace or completely reproduce the lectures. Please let me know if you spot any typos or errors. I will also post pdf files of older notes when necessary and type them up here in this latex file as frequently as I can manage. (Students who want to learn latex, can volunteer to help! I will send you the latex file as a sample to help you get started.) A good supplement to our text is the book [CVL]. It contains some more detailed derivations and some analysis (one theorem per chapter) of the many of the methods we will be discussing. Some of my notes here and in class will follow [CVL]. Another text to be aware of in [TB] which we use frequently in our Math 751, Numerical Linear Algebra course offered each Fall. I will adopt the idea there of introducing the singular value decomposition almost immediately, since it sheds so much light on the properties of matrices and linear systems $Ax = b$ which will be an ongoing concern to us.

A glance at the table of contents of our text [CM] will show you that this course will probably draw from every undergraduate math course that you have taken from calculus to differential equations to linear algebra. The point of this course is to develop efficient, accurate, and reliable methods for computing numerical solutions to many of the problems you have discussed in your core mathematics courses. You will need access to MATLAB and are advised to get the *Student Edition of MATLAB*.

Table 1: Approximate syllabus

Week	Tuesday lecture	Thursday lecture
1	Sec. 1.7	difference quotient error
2	lin. alg. rev.: Sec. 2.9 norms	..., matrix norms, Sec. 10.1 SVD
3	SVD cont.	SVD, oper. counts
4	oper. counts, start Chap. 2	Chap. 2...
5	lutx, bslashtx, oper. counts	$\det(A), A^{-1}$
6	tridisolve	sec. 3.1-poly. interp.
7	3.2,3.3 pw cubic, spline	3.4, 3.5 spline, periodic
8	finish Chap 3	4.1,4.2, Newton, Picard iter.
9	4.3,4.4 secant	Exam I thru 4.4
10		
11		
12		
13		
14		
15		

I'll include here some short, unpolished pieces of MATLAB code to illustrate the discussion. I'll try to make these available on my web page eventually, but many of them are short enough that you can just type them in yourself.

Numerical methods, along with theory and experiment, are fundamental to modern applied science and engineering. For an incisive overview of the field of numerical analysis, read Nick Trefethen's essay in the Appendix of [TB] on *The Definition of Numerical Analysis* or on his web site [LNT]—read it now and at the conclusion of this course. For those of you who consider yourselves to be pure mathematicians, you might take the attitude that you don't fully understand a topic unless you know how to compute effectively!

1.1 Floating point arithmetic-Lecture 1/22/13

We will not go through Chapter 1 of the text in detail. We reviewed section 1.7 of the text and the fact that a double precision floating point number is

stored as a 64 bit word with 52+1 bits used to store the mantissa. (The sign and the exponent base 2 are stored in the remaining 12 bits. 8 bits =1 byte, so a real floating point number is 8 bytes and 8 MB = 10^6 floating point numbers.) Since

$$2^{10} = 1024 \approx 10^3 \text{ we have } 2^{-52} = \left(2^{-10}\right)^{5.2} \approx \left(10^{-3}\right)^{5.2} \approx 10^{-16},$$

double precision gives us 16 digits accuracy, i.e., at most 16 significant digits. We call $\epsilon_{machine} \approx 10^{-16}$ the machine epsilon or roundoff (`eps` in MATLAB; see p. 35–36 for a more precise definition). When a floating point calculation is done the answer must be, in effect, rounded to 16 digits when it is stored. A good model for a floating point operation such as addition, that suppresses the details of the particular computer is

$$fl(x+y) = (x+y)(1+\epsilon) \text{ for some } |\epsilon| \leq \epsilon_{machine};$$

simlarly for multiplication and other operations. Note that this replaces the unknown features of a particular computer's adder with exact addition of the original numbers perturbed by a very small but unknown amount. This will facilate our occasional analysis of rounding error. To see the effect of rounding error, run the mfile

```
%machepstd.m
x=1;
y=1;
z=x+y;
n=0;
data=[0 1 2];
while z>x
   n+1;
   y=y/2;
   z=x+y;
   data=[data;n y z];
end
```

When does this program stop?

Accumulation of rounding error in the 16th digit is very slow when adding even very many numbers, so not much loss of accuracy can be expected. However, a great deal of accuracy can be lost if two nearly equal numbers are subtracted and many leading digits cancel. For instance, .12345-.12344=.00001

results in a loss of four significant digits, since the leading 0's in .00001 are not significant. This is an example of *catastrophic cancellation*. To see the effects of this in finite precison floating point arithmetic, run the following code for approximating $\exp(x)$ by a truncated Taylor series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

and compare the answer with the built-in $\exp(x)$. For $x = 1$ you can get an accurate answer with a few terms. For $x = 10, 20, \ldots$, you can eventually get good accuracy by taking enough terms. However, for $x = -10, -20, \ldots$ the summation of the series produces worse and worse results. This is due to the fact that $\exp(-20)$ is a very small number and we are trying to compute it by summing very large terms in the series with alternating signs. There MUST be a large amount of cancellation to produce a small number. However, we only have 16 digits available to cancel, so there is no way to get the correct value. An easy fix in this case the to note that $\exp(-x) = 1/\exp(x)$. See also problem 1.39 in the text.

```
%ExpTaytd
x=input('x value = ');
nterms=input('number of terms in series = ');
s=1;
term=1;
data=[0 s];
for k=1:nterms
   term=x*term/k;
   s=s+term;
   data=[data;k s];
end
```

Note that the code computes the terms in the series *recursively*.

1.2 Approximating derivatives numerically-Lecture 1/24/13

We will analyze the rounding error in divided difference approximations to the derivative. This will illustrate the limitations to doing calculus on a computer and give a simple example of how one might analyze the effects of roundoff error. First, we recall the following useful ideas:

Definition 1. *(Big-Oh notation)* $g(h) = O(h^k)$ *if there exists* $C, \epsilon > 0$ *such that* $|g(h)| \leq C|h|^k$ *for all* $|h| \leq \epsilon$.

Recall the Taylor series for f sufficiently differentiable at x,

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f^{(3)}(x)}{3!}h^3 + O(h^4). \tag{1}$$

Also recall the definition of the derivative of f,

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{2}$$

What happens if we try to compute $f'(x)$ numerically by letting $h \to 0$ on a computer using (2)? Let's call the *one-sided difference quotient*

$$D_h f(x) := \frac{f(x+h) - f(x)}{h}.$$

Note that, using (1),

$$\begin{aligned} D_h f(x) &= \frac{f(x+h) - f(x)}{h} \\ &= \frac{f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + O(h^3) - f(x)}{h} \\ &= f'(x) + \frac{f''(x)}{2}h + O(h^2). \end{aligned}$$

That is, $D_h f(x)$ gives an $O(h)$ approximation to $f'x)$. This is called *first order accuracy*, i.e., the error is $O(h^p)$ where $p = 1$ is the *order of accuracy*. This approximation could be made as accurately as we please by letting h get small, if we had exact (infinite precision) arithmetic. The trouble is that for most functions $f(x)$ the computer will give the value $f(x) + C\epsilon_{machine}$ for some small constant C, not the exact value $f(x)$, and similarly for $f(x+h)$. Therefore, the computer gives

$$\begin{aligned} D_h f(x) &= fl\left(\frac{f(x+h) - f(x)}{h}\right) = \frac{f(x+h) - f(x) + C\epsilon_{mach}}{h} \\ &= f'(x) + \frac{f''(x)}{2}h + O(h^2) + \frac{C\epsilon_{mach}}{h}, \end{aligned}$$

and so the error using $D_h f(x)$ can be modeled as

$$err_D(h) = |f'(x) - D_h f(x)| \approx C'h + \frac{C\epsilon_{mach}}{h}.$$

Note: When h is small $f(x+h) \approx f(x)$, so there will be catastrophic cancellation of leading digits in $f(x+h) - f(x)$.

If we run the following code for, say, $a = 1$ and $n = 20$, we will get approximations of $d\sin(x)/dx|_{x=1} = \cos(1)$ for $h = 10^{-1}, 10^{-2}, \ldots, 10^{-n}$. A plot of the log of the error is given in Figure 1. Note that the error decreases to about 10^{-8} for $h = 10^{-8}$ as h decreases and then starts to increase. We see that letting h go to zero does not improve the accuracy. What is happening and can we improve the results?

```
%derivtd
a=input('enter a =');
n=input('enter n =');
for k=1:n
   h(k)=10^(-k);
   Dh=( sin(a+h(k))-sin(a) )/h(k);
   err(k)=abs(Dh-cos(a));
end
```

Our simple model, in fact, roughly predicts the outcome of this calculation. To find the minimum of $err_D(h)$, solve

$$\frac{derr_D(h)}{dh} = C' - \frac{C\epsilon_{mach}}{h^2} = 0$$

to find the optimal h,

$$h_{opt} = \sqrt{(C/C')\epsilon_{mach}} \approx \sqrt{\epsilon_{mach}} \approx 10^{-8},$$

(assuming $C'/C = O(1)$), just as we see in Figure 1. We also see that the optimal error is roughly predicted,

$$err_{opt} = err_D(h_{opt}) = C'h_{opt} + \frac{C\epsilon_{mach}}{h_{opt}} \approx h_{opt} \approx 10^{-8}.$$

To improve these results requires a difference approximation of higher order which we now discuss and which will be the basis of your first assignment.

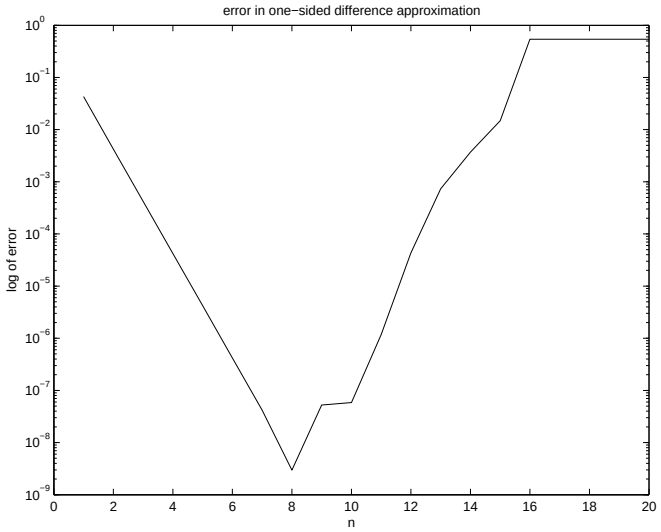

Figure 1: Error in the one-sided difference approximation to $d\sin(x)/dx|_{x=1} = \cos(1)$using $h = 10^{-n}, n = 1, 2, \ldots, 20$.

We develop the centered difference by averaging one sided differences,

$$\begin{aligned} C_h f(x) &:= \frac{1}{2}\left(D_h f(x) + D_{-h} f(x)\right) \\ &= \frac{1}{2}\left(\frac{f(x+h) - f(x)}{h} + \frac{f(x-h) - f(x)}{-h}\right) \\ &= \frac{f(x+h) - f(x-h)}{2h}. \end{aligned}$$

Using the Taylor series (1), we find that

$$\begin{aligned} f(x+h) - f(x-h) &= f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f^{(3)}(x)}{3!}h^3 + O(h^4) \\ &\quad -f(x) + f'(x)h - \frac{f''(x)}{2!}h^2 + \frac{f^{(3)}(x)}{3!}h^3 + O(h^4) \\ &= f'(x)2h + \frac{f^{(3)}(x)}{3}h^3 + O(h^4). \end{aligned}$$

Therefore,

$$C_h f(x) = \frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{f^{(3)}(x)}{3!}h^2 + O(h^3),$$

so the truncation error error (in exact arithmetic) is

$$f'(x) - C_h f(x) = \frac{f^{(3)}(x)}{3!}h^2 + O(h^3) = O(h^2).$$

That is, the centered difference is *second order accurate*. (Systematic derivations of higher order finite difference schemes are given in texts or courses on the numerical solution of differential equations.) Similar to the error using $D_h f(x)$, the error using $C_h f(x)$ in floating point arithmetic can be modelled as

$$err_C(h) = |f'(x) - C_h f(x)| \approx C'h^2 + \frac{C\epsilon_{mach}}{h}.$$

Homework 1 due Th 2/7/13.

a) Revise the code `derivtd.m` above to compute $err_C(h)$ and find h_{opt} and $err_C(h_{opt})$ computationally. Turn in a copy of your code and a plot like Figure 1.

b) Find h_{opt} by minimizing $err_C(h)$ above, as we did for $err_D(h)$ and compare your estimate to the computed value in a). Also, find $err_C(h_{opt})$ and compare it to the value in a).

Remark 1. How would you find h_{opt} if you did not know the exact derivative, did not have a good model of $err_D(h)$, and could only compute $D_h f(x)$ for various h's? One possible strategy might be to look at the differences between successive $D_h f(x)$'s: $|D_{h(k)}f(x) - D_{h(k-1)}f(x)|$ for $k = 1, \ldots, n$ and see where they cease to improve. This is shown in Figure 2 for $f(x) = \sin(x), x = 1$.

This is a simple example of the type of results that occur in the numerical solution of *inverse problems*, an active area of research the WSU Math Department. An inverse problem is roughly a problem where you can measure the "effect" and want to find the "cause", as contrasted with the more common *direct problem* where you have a mathematical model of the "cause" and want to compute the outcome or "effect". In practice, measurements always have errors or "noise". Often for inverse problems, the high frequency components in the (small) measurement noise can be greatly amplified in the calculation of the "cause" and completely overwhelm the answer. You must filter out the amplified noise without filtering out all the useful information.

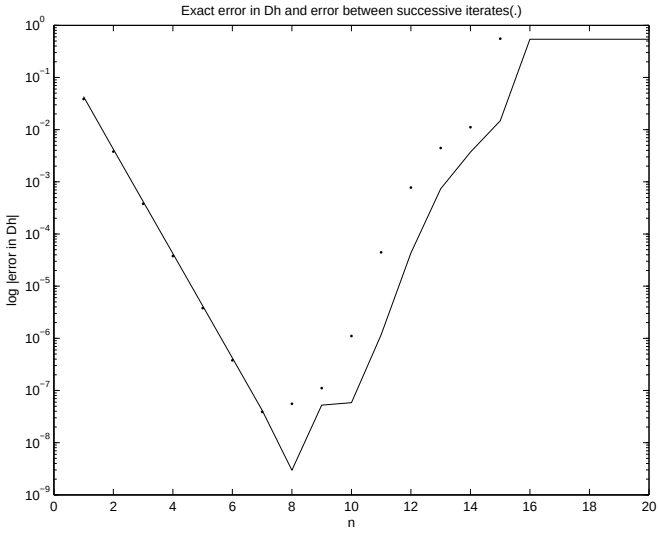

Figure 2: Successive differences in one-sided difference approximation to $d\sin(x)/dx|_{x=1} = \cos(1)$using $h = 10^{-n}, n = 1, 2, \ldots, 20$.

Such a procedure is called *regularization*. (The direct problem often damps out errors in high frequency terms and leads to easier calculations.) A typical error for an inverse problem calculation will start to converge to 0 and then diverge as in our Figures above. This is called *semiconvergence*. The problem is to select the optimal solution with no knowledge of the actual error. You should generally have some knowledge of the noise level δ if you hope to make any progress.

Extra credit homework 1, due date to be determined.

Reproduce Figure 2. Try the same procedure for the centered difference. Do you get a good estimate for h_{opt}? For floating point arithmetic the noise level $\delta = \epsilon_{mach}$. Suppose you only know $f(x)$ to accuracy $f(x)(1 + \texttt{rand} * \delta)$ where, say, $\delta = 10^{-6}$ or $10^{-3}, \ldots$ and `rand` is the MATLAB random number generator. See if you can find h_{opt} in this case. Write up your results in a clear way and turn in your writeup with codes and plots. You may work in teams of two or three people. Try to write a report in latex.

2 Vector and matrix norms and the SVD-Lect. 1/29/13, 1/31/13, 2/5/13

Recommended reading: text sec. 2.9 and 10.1, and [TB], Lectures 1–5 (especially for the case of complex vectors and matrices).

As background for Chapter 2 on Linear equations—solving $Ax = b$—we will review some facts from linear algebra and discuss matrix and vector norms. We will also discuss the singular value decomposition of a matrix (square or rectangular), `svd(A)`, since it gives so much information about A.

2.1 Linear algebra review-Lect. 1/29/13

See Lectures 1 and 2 of [TB] or your favorite linear algebra text. For convenience, we'll often denote an $m \times n$ matrix A as

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = [a_1|a_2|\cdots|a_n]$$

where a_j is the jth column of A. Then multiplication of A times an n (column) vector x gives this

$$y := Ax = \sum_{j=1}^{n} x_j a_j \in \text{range}(A) = \text{column space of} A.$$

Therefore we can solve $Ax = b$ if and only if b is in the vector subspace spanned by the columns of A, i.e. $< a_1, a_2, \ldots, a_n >$, the space of all linear combinations of the columns a_j of A.

Some facts and definitions from linear algebra:

Matrix-vector multiplication is *linear*, that is, $A(\alpha x + \beta y) = \alpha Ax + \beta Ay$ for any vectors x, y and scalars α, β.

The vectors $a_1, a_2, \ldots, a_n$ are *linearly independent*, if $x_1 a_1 + x_2 a_2 + \cdots + x_n a_n = 0$ implies the scalars $x_1 = x_2 = \cdots = x_n = 0$. In this case, the vectors a_j form a *basis* for the vector space $V(= R^n)$ and the *dimension* of V is $\dim(V)$.

The *null space* of A is $\text{null}(A) = \{x|Ax = 0\}$.

$\text{rank}(A)$ = number of linearly independent columns of A = number of linearly independent rows of $A = \text{rank}(A^T) \leq \min(m, n)$.

Theorem 1. *For $A \in R^{n\times n}$ the following conditions are equivalent:*

(a) A^{-1} *exists.*

(b) rank*(A)*=*n.*

(c) range*(A)*=R^n.

(d) null*(A)*=$\{0\}$.

(e) 0 is not an eigenvalue of A.

(f) $\det(A) \neq 0$.

(g) $Ax = b$ *has a unique solution.*

Note, a solution x to $Ax = b$ *exists* if $b \in$ range(A), and so if $y \in$null(A), then $A(x+y) = Ax = b$. Therefore, the solution x is *unique* if null$(A) = \{0\}$.

2.2 Vector and matrix norms-1/29/13, 2/5/13

Let our vectors be column vectors, $x, y \in R^{n\times 1}$. The *inner* or *dot product* is

$$x \cdot y := x^T y = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i.$$

The 2-*norm* of x is $\|x\|_2 := \sqrt{x \cdot x} = \sqrt{\sum_{i=1}^{n} x_i^2}$. Recall $x \cdot y = \|x\|_2 \|y\|_2 \cos\theta$ for the angle θ between x and y with $0 \leq \theta \leq \pi$. Thus the dot product give *geometric* information in Euclidean space R^n, i.e., lengths (or distances) and angles. Recall for $x, y \neq 0$, $x^T y = 0$ implies $\theta = \pi/2$ and so x and y are perpendicular or *orthogonal*.

The *Kronecker delta* is defined by

$$\delta_{ij} = \begin{cases} 1 & \text{when } i = j, \\ 0 & \text{when } i \neq j. \end{cases}$$

Definition 2. *A set of n linearly independent vectors $u_1, u_2, \ldots, u_n$ such that $u_i^T u_j = \delta_{ij}$ is an* orthonormal basis *for R^n, i.e., the u_i's are mutually orthogonal unit vectors.*

Definition 3. *An $n \times n$ matrix $Q = [q_1|q_2|\cdots|q_n]$ is* orthogonal *if $Q^TQ = [q_i^T q_j] = QQ^T = I$. Note that in this case the columns q_i of Q form an orthonormal basis for R^n and $Q^{-1} = Q^T$. (For complex matrices, replace the transpose of Q by the Hermitian transpose $Q^H = \overline{Q}^T$.)*

The general definition of a *norm* is

Definition 4. *A* norm *is a function $\|\cdot\| : R^n \to R$ such that for any $x, y \in R^n$ and any $\alpha \in R$*

(i) $\|x\| \geq 0$, *and* $\|x\| = 0$ *iff* $x = 0$

(ii) $\|\alpha x\| = |\alpha|\|x\|$

(iii) $\|x + y\| \leq \|x\| + \|y\|$, *the triangle inequality.*

Note that $\|x\|_2$ satisfies the definition. We will also occasionally use two other norms,

$$\|x\|_1 := \sum_{i=1}^{n} |x_i|$$

and

$$\|x\|_\infty := \max_{i=1,\ldots,n} |x_i|.$$

Recommended exercise: Prove that all of these norms satisfy the definition. These norms are built in to MATLAB as `norm(x)=norm(x,2)`, `norm(x,1)`, and `norm(x,inf)`. The unit circle in R^2 for each of these norms is shown in Figure (to be included).

Recommended problem (assigned below): Given a nonsingular matrix A and a norm $\|\cdot\|$, define $\|x\|_A := \|Ax\|$. Show that $\|\cdot\|_A$ is a norm.

Norms will be useful to us for computing errors, e.g. between an "exact" x and a computed approximation x_{comp} to x,

$$\textit{absolute error} := \|x - x_{comp}\| \text{ and } \textit{relative error} := \frac{\|x - x_{comp}\|}{\|x\|}.$$

In general, relative errors can be at best $\approx \epsilon_{mach}$ (or exactly 0). Errors using $\|\cdot\|_1$ give an upper bound on the *componentwise* error.

We will also use norms on matrices.

Definition 5. *The matrix norm* $\|A\|$ *induced by a vector norm* $\|x\|$ *is*

$$\|A\| := \max_{\|x\|=1} \|Ax\|.$$

Note that the definition of the matrix norm is in terms of (given) vector norms. Recall that $\|Ax\|$ is a continuous function of (the components of) x and so (by a Theorem from Advanced Calculus) must achieve its maximum value for some x on the compact(=closed and bounded) set $\|x\| = 1$. $\|Ax\|$ also achieves its mininmum. These facts are illustrated for the norm $\|\cdot\|_2$ for a 2×2 matrix in Figure 3. Note that for $x \neq 0$, $\|\frac{x}{\|x\|}\| = 1$. Therefore

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \text{the maximum magnification of } x \text{ by } A.$$

Example 1. In any induced norm, the identity I has norm 1,

$$\|I\| = \max_{\|x\|=1} \|Ix\| = \max_{\|x\|=1} \|x\| = 1.$$

Example 2. Note that for Q orthogonal,

$$\|Qx\|_2^2 = (Qx)^T Qx = x^T Q^T Qx = x^T x = \|x\|_2^2.$$

Therefore

$$\|Q\|_2 = \max_{\|x\|_2=1} \|Qx\|_2 = \max_{\|x\|_2=1} \|x\|_2 = 1.$$

Example 3. For $\|\cdot\| = \|\cdot\|_2, \|\cdot\|_1$, or $\|\cdot\|_\infty$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$,

$$\|\Lambda\| = \max_{i=1,\ldots,n} |\lambda_i|.$$

Can you prove this?

Matrix norms satisfy the definition of a norm. There are other matrix norms, but the induced norms satisfy the nice properties $\|Ax\| \leq \|A\|\|x\|$ and $\|AB\| \leq \|A\|\|B\|$.

Proof. The first inequality is obvious from the definition of $\|A\|$ and immediately gives the second inequality, since

$$\|ABx\| \leq \|A\|\|Bx\| \leq \|A\|\|B\|\|x\|.$$

□

Definition 6. *Two norms* $\|\cdot\|$ *and* $|||\cdot|||$ *are* equivalent *if there exist constants* $C \geq c > 0$ *such for any* x

$$c\|x\| \leq |||x||| \leq C\|x\|.$$

For instance, $\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty$, where $c = 1$ and $C = \sqrt{n}$. Note that for $x = [1, 0, 0, \cdots, 0]^T$ equality is achieved for the left inequality and for $x = [1, 1, \cdots, 1]^T$ equality is achieved for the right inequality, so no larger c or smaller C can be found. In such cases, the bounds are called "sharp". What are the best constants for other combinations or our norms? Similar results hold for our matrix norms.

Theorem 2. *All vector norms are equivalent.*

As a conseqence, if a sequence of vectors converges $x_i \to x$ in one norm, i.e., if $\|x_i - x\| \to 0$, then the sequence converges in any other norm $|||x_i - x||| \to 0$, so we may use any convenient norm to monitor convergence. (This equivalence is not true in general for norms for infinite dimensional spaces of functions, which makes functional analysis more complicated.)

2.3 Eigenvalues and eigenvectors - 2/5/13

Recall that for $A \in R^{n\times n}$, if $Ax = \lambda x$, then x is an *eigenvector* of A and λ is the associated *eigenvalue*. $p(\lambda) := \det(\lambda I - A)$ is the nth degree *characteristic polynomial* of A. The eigenvalues of A are the zeros of $p(\lambda)$, that is, the solutions of $p(\lambda) = 0$. For A real, the eigenvalues λ are real or occur in complex conjugate pairs, $\lambda = \mu + i\eta$ and $\overline{\lambda} = \mu - i\eta$ (Why?). For $n = 2$, we can solve for λ using the quadratic formula. This is the first "formula" we all learn. It solves a nonlinear equation. It is misleading, since there are many, many nonlinear equations in the world and few of them have formulas for their solution. In fact, for $n > 4$, Galois theory tells us that there are no "formulas" (=expressions in terms of the elementary operations of addition, subtraction, multiplication, division, or finding roots using the coefficients of $p(\lambda)$) for solving $p(\lambda) = 0$. This is a nonlinear problem and so, to solve the eigenvalue problem, we must use an *iterative method* such as *Newton's method* to produce a sequence converging to an eigenvalue. (Some standard numerical methods for finding eigenvalues, such as the QR algorithm, are discussed in Chapter 8 and built in to MATLAB. We may not get to this material. These methods are treated more fully in [TB] and our Math 751 course.)

Recall that there are exactly n eigenvalues $\lambda_i, i = 1, \ldots, n$ of our $n \times n$ matrix A, and so $p(\lambda) = \prod_{i=1}^{n}(\lambda - \lambda_i)$. A root λ_i that repeats exactly k times is called an eigenvalue of *(algebraic) multiplicity k*. To find the associated eigenvector, we must solve the singular linear problem $(A - \lambda_i I)x_i = 0$ for $x_i \neq 0$. Then cx_i is also a solution, so we may normalize x_i such that, e.g., $\|x_i\|_2 = 1$. There is always at least one such x_i. However, an eigenvalue of (algebraic) multiplicity $k \geq 2$ may have only j linearly independent eigenvectors with $1 \geq j < k$. j is the *geometric multiplicity* of λ_i. If $j < k$, the eigenvalue (and matrix) is *defective*. A canonical example is $A = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}$ where $\lambda_1 = 2$ with algebraic multiplicity 2, but $x_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is the only eigenvector. (You should be able to calculate eigenvalues and eigenvectors of 2 and selected 3×3 matrices by hand, if necessary, on an exam.) Therefore, the geometric multiplicity of $\lambda_1 = 2$ is 1 and A is *defective*, i.e., A does not have a complete set of n linearly independent eigenvectors; cf. the Jordan canonical form in [CM, Section 10.8] or in your linear algebra text.

If $A \in R^{n \times n}$ is not defective, with a complete set of n linearly independent eigenvectors x_i and corresponding eigenvalues λ_i, then the matrix $X := [x_1|x_2|\cdots|x_n]$ has rank n and so X^{-1} exists. Further,

$$\begin{aligned} AX &= [Ax_1|Ax_2|\cdots|Ax_n] \\ &= [\lambda_1 x_1|\lambda_2 x_2|\cdots|\lambda_n x_n] \\ &= [x_1|x_2|\cdots|x_n] \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & & & 0 \\ 0 & 0 & \cdots & 0 & \lambda_n \end{bmatrix} \\ &= X\Lambda, \end{aligned}$$

where $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$. That is, $A = X\Lambda X^{-1}$, is the *eigendecomposition* of A, or in other words, X *diagonalizes* A, i.e., $X^{-1}AX = \Lambda$. If $A = [a_{ij}] = [a_{ji}] = A^T$, then A is *symmetric*. In this case, A has real eigenvalues (Why?) and a complete set of orthonormal eigenvectors. That is X is orthogonal and so $A = X\Lambda X^T$. Such *factorizations* of matrices are a recurring theme in numerical linear algebra. The next topics, the svd and the LU factorizations, are examples.

Definition 7. *A is* positive definite *if $x^TAx > 0$ for all vectors $x \neq 0$.*

Note that if A is symmetric (to insure λ and x are real), positive definite and $Ax = \lambda x$ with $\|x\|_2 = 1$, then

$$0 < x^TAx = x^T(\lambda x) = \lambda x^Tx = \lambda\|x\|_2^2 = \lambda.$$

That is the eigenvalues of a positive definite matrix are positive.

2.4 The svd - 2/7/13

The svd is defined for general (complex) rectangular matrices. (In the complex case replace orthogonal matrices by unitary matrices U, i.e., $U^HU = I$, where $U^H = \overline{U}^T$ is the Hermitian transpose of U. Note that $U^H = U'$ and $U^T = U.'$ in MATLAB.) We'll just consider real square matrices for the moment. Since A^TA is symmetric positive (semi)definite (semi if $x^TA^TAx = 0$ for some $x \neq 0$), it has eigenvalues $\lambda_i \geq 0$ and a complete set of orthonormal eigenvectors $v_i, i = 1, \ldots, n$. Let $V := [v_1|v_2|\cdots|v_n]$. Then V is orthogonal and $A^TAV = V\Lambda$ where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ where we assume $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_n \geq 0$, wlog. That is $Av_i = \lambda_i v_i$. Define $\sigma_i := \sqrt{\lambda_i}$, called the ith *singular value* of A, and $u_i = \frac{1}{\sigma_i}Av_i$. Note that $u_i^Tu_j = \frac{1}{\sigma_i\sigma_j}v_i^TA^TAv_j = \delta_{ij}$, so the u_i's are orthonormal and $U := [u_1|u_2|\cdots|u_n]$ is orthogonal. This gives us the *singular value decomposition* or *svd* of A,

$$A = U\Sigma V^T \text{ where } \Sigma := \text{diag}(\sigma_1, \sigma_2, \cdots, \sigma_n) \text{ or } Av_i = \sigma_i u_i$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \sigma_n \geq 0$. The u_i's and v_i's are the left and right *singular vectors* of A.

We compute by hand the svd of $A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. First, $A^TA = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$. The eigenvalues of A^TA are found by solving

$$\det(\lambda I - A^TA) = \begin{vmatrix} \lambda - 2 & -2 \\ -2 & \lambda - 2 \end{vmatrix} = (\lambda - 2)^2 - 4 = 0.$$

Therefore, $\lambda_1 = 4 \geq \lambda_2 = 0$ with associated orthonormal eigenvectors, $v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ and $v_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$. Therefore, $\sigma_1 = 2 \geq \sigma_2 = 0$ and,

in this case $u_1 = v_1, u_2 = v_2$. (Since $\sigma_2 = 0$ we must just choose u_2 orthogonal to u_1.) Summarizing, the svd is

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = U\Sigma V^T = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}.$$

(In this case, the svd is just the eigendecompostion of A since A is symmetric postitive semidefinite. Try this matrix in MATLAB, `[U,S,V]=svd(A)`. Note that here rank(A)=1, which is the number of nonzero singular values of A, In general, the rank of a matrix is the number of nonzero singular values. However, in practice singular values may not be exactly 0. Instead, if an $n \times n$ matrix A has singular values,

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r >> \epsilon_{tol.} \geq \sigma_{r+1} \geq \cdots \geq \sigma_n \geq 0,$$

we can say the rank of A is r to a tolerance of $\epsilon_{tol.}$.

(Below there is a small sample code to try, illustrating the meaning of the svd for a 2×2 matrix.)

Our next main concern is to solve $Ax = b$, for A nonsingular using Gaussian elimination. We could use the svd instead, but it is more expensive. However, it is very illuminating to see how this is done. Since the u_i's and v_i's form orthonormal bases for R^n, for the given b we have $b = \sum_{j=1}^n b_j u_j$. Note that $b_j = u_j^T b = \|b\|_2 \cos\theta$, the component of b in the u_j direction. Similarly, the unknown $x = \sum_{j=1}^n x_j v_j$ with $x_j = v_j^T x$. We need to find the x_j's. Note that

$$Ax = U\Sigma V^T x = \sum_{j=1}^n \sigma_j (v_j^T x) u_j = \sum_{j=1}^n \sigma_j x_j u_j = b = \sum_{j=1}^n b_j u_j.$$

Therefore, the solution is $x_j = \frac{b_j}{\sigma_j}, j = 1, \ldots, n$, i.e,

$$x = \sum_{j=1}^n \frac{b_j}{\sigma_j} v_j = \sum_{j=1}^n \frac{u_j^T b}{\sigma_j} v_j.$$

Another way to write the svd is as an outer product expansion,

$$A = \sum_{i=1}^n \sigma_i u_i v_i^T.$$

The $n \times n$ matrices $u_i v_i^T$'s are *outer products* of u_i and v_i (cf. the inner product $u_i^T v_i$). They are rank one. (Why?) The inverse A^{-1} can be written as

$$A^{-1} = \sum_{i=1}^{n} \frac{v_i u_i^T}{\sigma_i}.$$

Is this the svd of A^{-1}?

Remark 2. For A nearly singular the $\sigma_j/\sigma_1 \approx 0$ for $j \approx n$. Therefore, if those b_j's are $O(1)$, i.e. not very small, $x_j = b_j/\sigma_j$ will be very large. If there are errors in the b_j's associated with limits in the accuracy of measurements of b ("noise" in the data), these errors are greatly amplified by the small singular values and can overwhelm the computed solution x making it useless. The u_i's and v_i for large $i \approx n$ are usually associated with high frequency or highly oscillatory components of b and x. One way to get a useful approximate solution x is to filter or damp out inaccurate high frequency components x_j. This is another example of *regularzation*, referred to above. Regularization techniques, as we said, are frequently needed to solve *inverse problems*, where the matrices are often nearly singular (*ill-conditioned*; see below) and the data may have only a few percent accuracy. Such techniques were use by some of at WSU to solve inverse problems in acoustics [DIVW1, DIVW2, DH]. These problems arose in attempts to locate sources of noise in the cabins of Cessna business jets by taking pressure measurements (the data b or "effect") near the fuselage and trying to reconstruct the boundary vibrations (the solution x or the "cause"). This is sometimes called *nearfield acoustic holography* and has been used by the U. S. Naval Research Laboratory in Washington, D.C. to understand noise sources in submarines in order to make them quieter.

Question: If you run the given MATLAB code for plotting $y = Ax$, the lines in Figure 3 indicating min $\|y\|_2$ and max $\|y\|_2$ are not always orthogonal, especially for elongated ellipses. Why?

```
% plot y=Ax, A 2x2 matrix, ||x||=1
n=128;
%A=rand(2,2);
A=randn(2,2);
x=[cos(2*pi*[0:n]/n);sin(2*pi*[0:n]/n)];
y=A*x;
for j=1:n
```

```
 y2(j)orm(y(:,j),2);
end
[mmax,kmax]=max(y2);
[mmin,kmin]=min(y2);
subplot(1,2,1)
plot(x(1,:),x(2,:));
hold on;
plot([0 x(1,kmax)],[0,x(2,kmax)]);
hold on;
plot([0 x(1,kmin)],[0,x(2,kmin)]);
axis equal
hold on;
subplot(1,2,2)
plot(y(1,:),y(2,:))
hold on;
plot([0 y(1,kmax)],[0,y(2,kmax)]);
hold on;
plot([0 y(1,kmin)],[0,y(2,kmin)]);
axis equal
hold on;
```

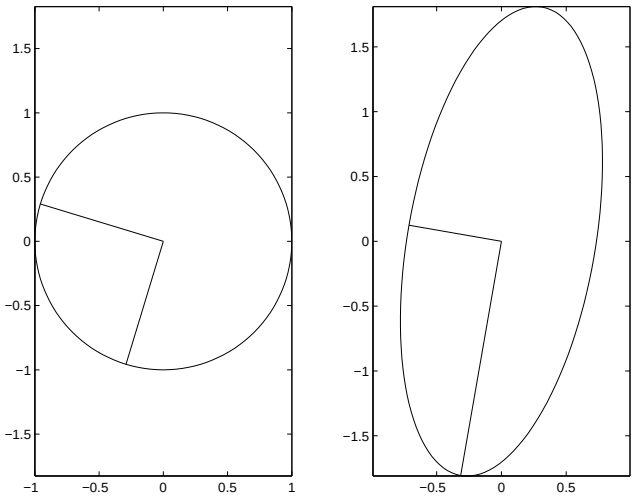

Figure 3: Illustration of svd for a 2×2 matrix.

2.5 Operation counts - 2/7/13

It is useful to understand the amount of computational work and memory is required by a computer in order to run a given algorithm on a specified problem involving, say, the solution of n equations or n numbers. Since computers now do many things in parallel and, e.g, MATLAB tries to optimize operations like matrix-vector multiplication, counting the number of operations may often overestimate the work required, but it can indicate where a "bottleneck" is in your computations. Our "Big-Oh" notation is handy here.

Definition 8. *(Big Oh notation)* $f(n) = O(g(n))$ *if there exists* $C, N > 0$ *such that* $|f(n)| \leq C|g(n)|$ *for all* $|n| \geq N$.

For instance, $-2n^3 - 10n - 100 = O(n^3)$ where you could take $C = 2.001$ for N sufficiently large. If we are discussing computational work or "cost", then we want to find some simple function $g(n)$ of n such that, say, the number of floating point operations $f(n)$ needed to solve a problem for n equations or pieces of data is $O(g(n))$. Usually n is quite large or the computation has to be done again and again and should done efficiently.

Here are some simple examples.

(1) Searching a phone book with $n = 2^m$ names by successively subdividing the phone book according to which half the (alphabetically listed) name is in requires, in the worst case, $m = \log_2 n$ subdivisions, i.e., $O(\log n)$. (Operations counts or *complexity* results usually differ betwen computer science and numerical analysis. Computer science algorithms often involve searching and sorting n things, rather than doing arithmetic operations on n numbers.)

(2) Computing the inner and outer products of two (real, floating-point) $n \times 1$ vectors, u and v from the definitions are, for the inner product,

$$u^T v = \sum_{i=1}^{n} u_i v_i,$$

which requires n floating point multiplications and $n - 1$ floating ponts additions, or $O(n)$ "flops". The outer product is the $n \times n$ matrix, uv^T whose entries are

$$[uv^T]_{i,j} = u_i v_j.$$

How many computations does this require? If x is an $n \times 1$ vector, how would you best compute the matirx-vector product, $uv^T x$, as $(uv^T)x$ or as $u(v^T x)$? What is the operation count each way, $O(n), O(n^2), O(n \log n)$, or what?

(3) How many flops are required to compute Ax where A is an $n \times n$ matrix and x is an $n \times 1$ vector? Count from the definition,

$$[Ax]_i = \sum_{j=1}^{n} A_{ij} x_j, \quad \text{for} \quad i = 1, \ldots, n.$$

(4) How many flops to form AB for two $n \times n$ matrices? How would you compute ABx for an $n \times 1$ vector x?

Suggested short problem: This problem will give you some feeling for the speed of your computer. Try the following MATLAB operations to compare times for matrix-vector and matrix-matrix multiplication. Are they $O(n^2)$ and $O(n^3)$? Make a table listing times for $n = 10, 100, 1000$. Can you estimate the speed of your computer in $flops/second$?

```
>> n=1000;
>> A=ones(n,n);
>> x=ones(n,1);
>> tic; A*x; toc
Elapsed time is 0.005466 seconds.
>> tic; A*A; toc
Elapsed time is 1.817824 seconds.
```

2.6 LU factorization - 2/14/13, 2/19/13

MATLAB solves $Ax = b$ with $x = A\backslash b$ by GEPP. See Sections 2.1–2.6 and handout on $PA = LU$. Operation counts: GEPP (= Gaussian Elimination with Partial Pivoting) costs $O(n^3)$ flops. Forward and backsubstitution cost $O(n^2)$ flops. See `lutx, bslashtx, lugui`. Here are some simple examples.

First, let's use Gaussian elimination to solve

$$\begin{aligned} x_1 + x_2 &= 3 \\ 2x_1 - x_2 &= 1. \end{aligned}$$

We write this as

$$Ax = b,$$

where

$$A = \begin{bmatrix} 1 & 1 \\ 2 & -1 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

We want to eliminate x_1 from one of the equations by taking linear combinations of the equations. We'll see that it will be numerically expedient to "pivot" and make the equation equation with the largest coefficient of x_1 to top, here by just exchanging the order of the two equations,

$$\begin{aligned} 2x_1 - x_2 &= 1 \\ x_1 + x_2 &= 3. \end{aligned}$$

Throughout this course it will often be useful to think of procedures in terms of matrix operations and matrix factorizations (like the svd). We can represent pivoting in term of multplication by a matrix P which is just the indentity matrix with its rows (or columns) permuted,

$$\begin{aligned} PAx &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ &= Pb = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}. \end{aligned}$$

P just exchanges the first and second rows of the matrix A and the vector b giving system

$$PAx = \begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} = Pb.$$

Next, we eliminate x_1 from the second equation by adding -1/2 times the first equation to the second equation. This can be represented in matrix form by multiplication by a matrix M_1,

$$M_1PAx = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{5}{2} \end{bmatrix},$$

or, with the *upper triangular matrix* $U := M_1PA$,

$$Ux = \begin{bmatrix} 2 & -1 \\ 0 & \frac{5}{2} \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{5}{2} \end{bmatrix} = M_1Pb.$$

This upper triangular system is easily "backsolved" as

$$\begin{aligned} x_2 &= (5/2)/(5/2) = 1, \\ x_1 &= (1 + x_2)/2 = (1 + 1)/2 = 1. \end{aligned}$$

Next, note that the inverse of M_1 is easy to find. All you have to do is change the sign of the multiplier -1/2,

$$L_1 := M_1^{-1} = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{bmatrix}.$$

$L = L_1$ is a *unit lower triangular* matrix. We may therefore rewrite what we have done as a matrix factorization of A known as the LU-factorization,

$$PA = LU.$$

These steps generalize to the $n \times n$-dimensional case as follows. We indicate the computational cost for each step. (We will derive these costs shortly.)

To solve $Ax = b$ with GEPP, we perform the following steps:

1. Find $PA = LU$. Cost: $O(n^3)$.
2. Forward solve $Ly = Pb$. (That is $y = L^{-1}Pb = MPb$) Cost: $O(n^2)$.
3. Back solve $Ux = y$. (Note that then $PAx = LUx = Ly = Pb$.) Cost: $O(n^2)$.

To illustrate why we should pivot, consider the following simple system; see also the other example in the text. We want to solve

$$\begin{aligned} .001x_1 + x_2 &= 1.001 \\ x_1 + x_2 &= 2. \end{aligned}$$

The solution is obviously $x = [1; 1]$. If we don't pivot, we multiply the first equation by 1000 and subtract it from the second equation,

$$\begin{aligned} x_1 + x_2 &= 2 \\ -x_1 - 1000x_2 &= -1001 \end{aligned}$$

to get

$$-999x_2 = -999.$$

This gives $x_2 = 1$. Substituting into the first equation gives

$$.001x_1 + x_2 = 001x_1 + 1 = 1.001.$$

Solving for x_1 gives

$$x_1 = \frac{1.001 - 1.000}{.001} = 1.$$

However, we have formed x_1 by subtracting $1.001 - 1.000$ by (catastrophically) cancelling leading digits and then dividing by a small number .001. Suppose we only had a few digit arithmetic and there had been a small rounding error in x_2, e.g. $x_2 = .999$. Then we'd have instead

$$x_1 = \frac{1.001 - .999}{.001} = \frac{.002}{.001} = 2,$$

which is way off!

Here is a an example of $PA = LU$ for a 3×3 matrix which will illustrate the general case; see [TB] for a similar discussion. You are asked to imitate this in written Homework 5, below. Let

$$A = \begin{bmatrix} 3 & 17 & 10 \\ 2 & 4 & -2 \\ 6 & 18 & -12 \end{bmatrix}.$$

the first pivot is

$$P_1 A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} A = \begin{bmatrix} 6 & 18 & -12 \\ 2 & 4 & -2 \\ 3 & 17 & 10 \end{bmatrix}.$$

Next, we eliminate x_1 from the last two equations by zeroing the column below $PA(1,1) = 6$.

$$M_1 P_1 A = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{3} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{bmatrix} P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & -2 & 2 \\ 0 & 8 & 16 \end{bmatrix}.$$

(The computer does not have to calculate the 0's.) The second pivot is

$$P_2 M_1 P_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} M_1 P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & -2 & 2 \end{bmatrix}.$$

Finally, we zero the (3,2) entry -2 to get

$$M_2 P_2 M_1 P_1 A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{4} & 1 \end{bmatrix} P_2 M_1 P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & 0 & 6 \end{bmatrix} =: U,$$

where U is an upper triangular matrix. Next, note that $P_2^{-1} = P_2^T = P_2$ and use the trick of multiplying by $I = P_2^{-1}P_2$ (there are only two basic computational tricks: multiplying by 1 or adding 0, in clever ways),

$$U = M_2P_2M_1P_1A = M_2P_2M_1(P_2^{-1}P_2)P_1A = M_2(P_2M_1P_2^{-1})P_2P_1A.$$

Note that

$$\begin{aligned}
\hat{M}_1 := P_2M_1P_2^{-1} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{3} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{3} & 0 & 1 \\ -\frac{1}{2} & 1 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{3} & 0 & 1 \end{bmatrix}.
\end{aligned}$$

(Note that multiplying on the right by P_2 permutes columns and multiplying on the left permutes rows.) The inverses of the multiplier matrices just change the signs of the entires below the diagonal,

$$L_1 := \hat{M}_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & 0 & 1 \end{bmatrix} \quad \text{and} \quad L_2 := M_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{4} & 1 \end{bmatrix}.$$

The product L_2L_1 is very simple,

$$L := L_1L_2 = (M_2\hat{M}_1)^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & -\frac{1}{4} & 1 \end{bmatrix}.$$

In addition, we define

$$P = P_2P_1 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

(products of permutation matrices are permutation matrices.) Putting this

all together, we have the LU–factorization of A,

$$\begin{aligned} PA &= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 3 & 17 & 10 \\ 2 & 4 & -2 \\ 6 & 18 & -12 \end{bmatrix} \\ &= \begin{bmatrix} 6 & 18 & -12 \\ 3 & 17 & 10 \\ 2 & 4 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & -\frac{1}{4} & 1 \end{bmatrix} \begin{bmatrix} 6 & 18 & -12 \\ 0 & 8 & 16 \\ 0 & 0 & 6 \end{bmatrix} = LU. \end{aligned}$$

Note that we can store P as the vector [3 1 2], meaning that the first row of P is the third row of I, the second row is the first row of I, and the third row is the second row of I. All of this now generalizes to the $n \times n$ case.

An advantage of this organization of Gaussian elimination is that solving triangular systems is easy an costs $O(n^2)$. If, for instance, you had to solve $Ax = b$ for multiple right-hand sides b and the same A, you would only have to perform $PA = LU$ once. We rarely want to find A^{-1}, but if we did, how could you do it? (Hint: What is the solution to $AX = I$? See Prob 2.11 in the text.) Also, suppose you really wanted to know $\det(A)$. How could you use the LU–factorization? (See Prob. 2.7 of the text. Compare it with the implementation of the linear algebra definition in `dettd` below.)

Let's check the operation counts for forward and back substitution and LU by looking at the text demo codes in Sec. 2.7 and counting operations. The forward substition `x=forward(L,b)` solves $Lx = b$ by implementing

$$x_k = (b_k - L_{k,1}x_1 - L_{k,2}x_2 - \cdots - L_{k,k-1}x_{k-1})/L_{k,k} \quad k = 1, \ldots, n.$$

(Note that the code sets $x = b$ and overwrites x to use memory efficiently.) The count is for $k = 1$ one division, for $k = 2$, one subtraction, one mult., and one divide, for $k = 3$, two subtractions, two mults. and a divide, etc. That is $2k - 1$ flops for each k, for a total of

$$\sum_{k=1}^{n}(2k-1) = 2\left(\sum_{k=1}^{n} k\right) - n = 2\frac{n(n+1)}{2} - n = n^2 + n - n = n^2 \quad \text{flops},$$

that is, $O(n^2)$ as we said. The count for the back substitution `x=backsubs(U,b)` is roughly the same.

If you forgot the forget for things like $\sum_{k=1}^{n} k$ you can get an asymptotic (in n) estimate by integrating,

$$\sum_{k=1}^{n} k \sim \int_1^n k dk = \frac{1}{2}k^2]_1^n \sim \frac{1}{2}n^2.$$

Here we've used the handy symbol $\sim$ where

$$f(n) \sim g(n) \quad \text{if} \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

For the LU factorization `[L,U,p]=lutx(A)` we have (ignoring the "bookkeeping" of the pivoting using `p([k m])`) the main part of the loop is

```
for k = 1:n-1
 i = k+1:n;
  A(i,k) = A(i,k)/A(k,k) % compute multipliers and
                         % overwrite col k of A below (k,k)
  j = k+1:n;
  A(i,j) = A(i,j) - A(i,k)*A(k,j); % update remainder of A
 end
end
```

The most work is in the update of the remaider of A. You can visulaize this as filling a square array of dimensions $(n-k)\times(n-k)$ with a mult. and a subtraction for each entry. The total cost ignoring the lower order computation of the multipliers is

$$2\sum_{k=1}^{n-1}(n-k)^2 = 2\sum_{k=1}^{n-1}k^2 = ???$$

If you forgot the formula, look in your calculus book. We can get the asymptotic behavior, which is all we're really interested in, from

$$\sum_{k=1}^{n-1}k^2 \sim \int^{n} k^2 dk \sim \frac{1}{3}n^3.$$

That is, the LU factorization cost $O(n^3)$ for large n, as claimed.

2.7 Condition number - 2/16/10

Sections 2.6–2.9.

Some facts about condition number $\kappa(A) = \|A\|\|A^{-1}\|$.

$\kappa(I) = 1$.

$\kappa(A) = \|A\|\|A^{-1}\| \geq \|AA^{-1}\| = \|I\| = 1$.

$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n}$.

Theorem 3. *If A is nonsingular and $\frac{\|E\|}{\|A\|} < \frac{1}{\kappa(A)}$, then $A+E$ is nonsingular (i.e., $\frac{1}{\kappa(A)}$ is the relative distance from A to the nearest singular matrix. This implies that the set of all nonsingular matrices is open.)*

Proof. Suppose $A + E$ is singular. The there exists and $x \neq 0$ such that $(A + E)x = 0$. Therefore

$$\begin{aligned} Ax &= -Ex \\ x &= A^{-1}Ex \\ \|x\| &\leq \|A^{-1}\|\|E\|\|x\| \\ \text{and so} \frac{1}{\kappa(A)} &= \frac{1}{\|A\|\|A^{-1}\|} \leq \frac{\|E\|}{\|A\|}. \end{aligned}$$

□

According to these estimates, if $cond(A) \approx 10^p$, you can expect lose about the last p digits in the computed solution. To get some feeling for the error estimates and conditioning, try using `xc=A\b` to solve $Ax = b$ where

$$Ax = \begin{bmatrix} 1 & 1.001 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.001 \\ 2 \end{bmatrix} = b.$$

Be sure to use `format long` to see all the digits. Change 1.001 to 1.0001 1.00001, etc., and the corresponding b while leaving $x = [1, 1]^T$, and watch $xc - x$, the residual $b - Axc$, and `cond(A)` change.

2.8 How NOT to compute determinants, inverses, etc. - 2/18/20

See text Probs. 2.7 and 2.11 for effective methods to compute $\det(A)$ and A^{-1}.

You should NOT use the recursive defintion from linear algebra where an $n \times n$ determinant is calculated in terms of n $n-1 \times n-1$ determinants. The cost would therefors be $O(n!)$. Here is an implementation of this. Note that it use the recursive properties of MATLAB. That is the code calls itself:

```
function y = dettd(A)
% Computes the determinant of A recursively
% from the linear algebra definition.
```

```
% T. DeLillo, Math 451, Fall 2001.
[m,n]=size(A);
if m ~= n
    disp('A is not a square matrix');
    break
elseif n == 1
    y=A(1,1);
elseif n > 1
    y=0;
    for j=1:n
        B=A;
        B(:,j)=[];
        B(1,:)=[];
        y=y + (-1)^(1+j)*A(1,j)*dettd(B);
    end
end
```

Try this on some small matrices and use `tic;\dettd(A);toc;` to convince yourself of the $O(n!)$ cost. If you try a matrix with n greater than 8 or 9, you should be prepared to kill the program with `ctrl^`.

2.9 Sparse matrices - 2/23/10

Section 2.10. Thomas algorithm `tridisolve` for solving a tridiagonal system costs $O(n)$ flops. Whenever a matrix A has "structure" or sparsity the cost of solving $Ax = b$ can often be reduced.

3 Interpolation

3.1 The interpolating polynomial - 2/25/10

Section 3.7, `interpgui` gives an overview and comparison of polynomial, piecewise linear, piecewise Hermite cubic, and cubic spline interpolation which we will study in detail. Discuss Section 3.1, Lagrangian form `polyinterp`, power form and Vandermonde matrix `vander`.

3.2 Lecture - 3/2/10

To illustrate the products multiplying the y_k's in the Lagrange form of the interpolating polynomial in section 3.1, try this:

```
>> x=[0 1 2 3];
>> y=[0 0 1 0];
>> u=(0:0.1:3);
>> v=polyinterp(x,y,u);
>> plot(x,y,'o',u,v)
```

Polynomial interpolation at equidistant points and the Runge phenomenon; run `rungeinterp` and see Prob. 3.9; remark on Weierstrass approximation theorem.

Section 3.2, pw linear interpolation, Section 3.3 and 3.4, Piecewise linear Hermite cubic and shape preserving pw cubic interpolation...; see Hŕner's algorithm, `polyval` (built into MATLAB), `piecelin`, `pchip`,...

Figure 4 graphically illustrates the difference between a C^1 pw Hermite cubic and a C^2 pw cubic spline. Figure 4 (top) is a C^1 function generated by the following MATLAB commandes.

```
>> x=linspace(-1,1,1000);
>> y=zeros(1:500);
>> y=zeros(1,500);
>> y=zeros(501,1000)=x(501:1000).^2;
>> y(501:1000)=x(501:1000).^2;
>> plot(x,y);
>> axis([-1 1 -1 1])
```

Figure 4 (bottom) is a C^1 function generated by the following MATLAB commandes.

```
>> x=linspace(-1,1,1000);
>> y=zeros(1:500);
>> y=zeros(1,500);
>> y=zeros(501,1000)=x(501:1000).^3;
>> y(501:1000)=x(501:1000).^3;2>> plot(x,y);
>> axis([-1 1 -1 1])
```

3.3 Lecture - 3/4/10

Sections 3.5, 3.6 Cubic spline interpolation. We went through the derivation of the tridiagonal system for the $d_k = P'(x_k), k = 1, \ldots, n$ such that $P''(x) \in C^2[x_1, x_n]$. This piecewise cubic interpolationg polynomial is known as a cubic spline. The derivation is given in the text and in my posted handwritten notes. The slopes d_1, d_n at the endpoints can be chosen in three ways: (1) the *not-a-knot* strategy given in the text, (2) the *draftsman's spline*[1] where $P''(x_1) = P''(x_n) = 0$, so that slopes $P'(x)$ is constant to the left and right of $[x_1, x_n]$ like a draftsman's spline, and (3) the periodic cubic spline, which we discuss here in more detail. Here the period is $\Delta = x_n - x_1$ and so $P^j(x + \Delta) = P^j(x), j = 0, 1, 2$. Therefore, $P'(x_1) = d_1 = P'(x_n) = d_n$. This reduces the number of unknown d_k's by 1. In the case of equidistant x_k's where $h_k = h = x_2 - x_1$, using the basic relation

$$d_{k-1} + 4d_k + d_{k+1} = 3(\delta_{k-1} + \delta_k)$$

for $k = 1$ and $k = n - 1$ and $d_0 = d_{n-1}$ and $d_1 = d_n$ by periodicity, we get the near-tridiagonal system for the d_k's,

$$Ad = \begin{bmatrix} 4 & 1 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 4 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 4 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 4 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 4 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-3} \\ d_{n-2} \\ d_{n-1} \end{bmatrix} = 3 \begin{bmatrix} \delta_{n-1} + \delta_1 \\ \delta_1 + \delta_2 \\ \delta_2 + \delta_3 \\ \vdots \\ \delta_{n-4} + \delta_{n-3} \\ \delta_{n-3} + \delta_{n-2} \\ \delta_{n-2} + \delta_{n-1} \end{bmatrix} =: r.$$

The periodicity introduces 1's in th upper right and lower left hand corners of A. You are asked to fill in some entries of A for case of general h_k and modify `splinetx` in Computer Homework III (Problem 3.13).

[1] A draftsman's spline is a flexible rod used in the old, pre-computer-graphics days which was placed against the pins (knots) on draft paper in order to help draw a smooth curve through the points. At the overhangs, the rod is straight. Between the endpoints, the total bending (integral of the $|P''(x)|^2$) is minimized; see [Hen, Section 5.8] to see that this condition leads to our spline equations.

4 Zeros and Roots-solving nonlinear equations-Lectures 3/11/10, 3/13/10, and 3/23/10.

(No class 3/16/10 and 3/18/10 for spring break.)

In general, we can't expect "formulas" for solutions to nonlinear problems (The first formula we all learn, the quadratic formula, is misleading, in that sense, and not even useful numerically if the roots are close together.) Therefore, we usually need some type of iterative method which produces a sequence converging to the solution, $x_n \to x, n \to \infty$. The following definitions are useful.

Definition 9. *If x_n converges to x and with $e_n := |x - x_n|$ and there is a constant $c \geq 0$ such that*

$$e_{n+1} \leq ce_n^p, \qquad for\ all\ n\ sufficiently\ large,$$

then p is the order of convergence. *$p = 1$ is called* linear convergence *and $p = 2$ is called* quadratic convergence. *If $p = 1$ and $\lim_{n\to\infty} e_{n+1}/e_n = r$ and $0 < r < 1$, then r is called the (linear)* convergence ratio *and $e_n \leq Cr^n$ for some $C > 0$.*

Let's recall some methods for solving a scalar equation $f(x) = 0$, where f is a nonlinear function of x. If $f'(x)$ exists we can try Newton's method, of course. Instead, suppose f has the form $f(x) = x - g(x)$. The solution x is therefore a *fixed point* of $g(x)$ since $x = g(x)$. Then a *method of successive approximation* suggests itself: Start with an initial guess x_0 and iterate,

$$x_{n+1} = g(x_n), n = 0, 1, 2 \ldots.$$

If $x = x_{exact}$ is the solution, then convergence can be shown, if $r = |g'(x)| < 1$ with convergence ratio r and x_0 is sufficiently close to x_{exact}. If g is contnuously differentiable, this follows from

$$x - x_{n+1} = g(x) - g(x_n) = g'(\xi_n)(x - x_n)$$

for some ξ_n between x and x_n. Therefore,

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n} = |g'(x)|$$

and so x_n converges to x, if $r|g'(x)| < 1$.

Possible problem or demo. As an example, try $x = g(x) = \cos x$, if you've never done so. You don't have to do any coding in MATLAB. Note that there is a solution between 0 and $\pi/2$. Then just type, say, `x=1` and, repeatedly, `x=cos(x)` and watch the digits slowly line up. Be sure to use `format long`. (Try Newton's method using `x=x-f(x)/f'(x)` to see faster quadratic convergence.)

A method of successive approximation or *Picard's method* can also be used in other contexts to establish existence of functions solving particular equations. A familiar example, similar to Theordorsen's method, is the proof of existence and uniqueness of solutions to the initial value problem for a general first order differential equation,

$$y' = f(x, y), \quad y(0) = y_0,$$

as discussed in, e.g., [**?**, Sec. 2.11]. One starts with and initial guess $\phi_0(x)$ for the solution and iterates using the integral form of the equation,

$$\phi_{n+1}(x) = y_0 + \int_0^x f(t, \phi_n(t))dt.$$

Convergence of the iterates to a unique solution can then be shown in a small interval near 0 if f and f_y are continuous.

Th 3/25/10 - Exam I on material through Section 4.4 of the text and the notes to that point. I will base most questions on simple arguments or calculations in these notes or the text. The exam is mainly meant to see that you have been following the basic mathematical steps in the notes and the text. The exam will consist of about five or six problems similar the the following sample problems.

1. Derive the first few terms of the Taylor series for a given function.
2. Find the order of accuracy of the one-sided or the centered difference approximation to the derivative.
3. Show that the 1-norm $\|x\|_1$ satisfies the definition of a norm.

4. If Q is an orthogonal matrix, show that

   a) $\|Qx\|_2 = \|x\|_2$.

   b) $\text{cond}(Q) := \kappa_2(Q) = 1$.

   c) If θ is the smallest angle between x and y and ϕ is the smallest angle between Qx and Qy, show that $\theta = \phi$.

5. Find the SVD or eigenvalues and eigenvectors of a given matrix A.

6. Find $PA = LU$ using GEPP.

7. Solve $Ax = b$ using $PA = LU$ and forward and backward substitution.

8. Count the number of floating point operations (flops) for forward or backward substitution, GEPP, matrix-vector or matrix-matrix multiplication.

9. If $Ax = b$, $A(x+\delta x) = b+\delta b$, and A^{-1} exists, show that $\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\delta b\|}{\|b\|}$

10. Set up the Vandermonde system for polynomial interpolation and solve it for a simple example.

11. Dervive and perform a step or two of Picard, Newton, or the secant method on a given function.

12. Explain a simple segment of MATLAB code.

13. Explain some basic fact about finite precision floating point arithmetic.

5 Homework

(Mainly) Written Homework

Do not use Blackboard for written homework. Hand in hard copies unless otherwise indicated.

Homework 1 due Th 2/7/13.

a) (2 pts) Revise the code `derivtd.m` above to compute $err_C(h)$ and find h_{opt} and $err_C(h_{opt})$ computationally. Turn in a copy of your code and a plot like Figure 1. (Do not upload to Blackboard.)

b) (2 pts) Find h_{opt} by minimizing $err_C(h)$ above, as we did for $err_D(h)$ and compare your estimate to the computed value in a). Also, find $err_C(h_{opt})$ and compare it to the value in a).

Homework 2 (4 pts.) due Th 2/14/13. Let $\|x\|$ be a vector norm for $x \in R^n$ and let $A \in R^{n\times n}$ be a nonsingular matrix. Show that $\|x\|_A := \|Ax\|$ is a vector norm.

Homework 3 (4 pts.) due Th 2/21/13. Find the svd of

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

by a hand calculation and compare with the MATLAB result.

Homework 4 (4 pts.) due Th 2/21/13. Show that a real 2×2 matrix A maps the unit circle $\|x\|_2 = 1$ to an ellipse $y = Ax$ as illustrated in Figure 3. (Hint: Verify the components of y satisfy the standard equation of the ellipse $\frac{y_1^2}{a^2} + \frac{y_2^2}{b^2} = 1$, if you choose the coordinate system and a and b properly.)

Homework 5 (4 pts.) due Th 2/28/13. Find $PA = LU$ by a hand calculation using G.E.P.P. following the class example for

$$A = \begin{bmatrix} 2 & 5 & 5 \\ 6 & 12 & 6 \\ 3 & 8 & 7 \end{bmatrix}.$$

Homework 6 due Th 3/14/13.

a) (4 pts) Show that for $n = 3$ and $x_j \neq x_k, j \neq k$, the Vandermonde matrix V is nonsingular by showing that

$$\det V = \begin{vmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{vmatrix} \neq 0.$$

b) (Bonus: 4 pts—due date open) Show $\det V \neq 0$ for the general $n \times n$ Vandermonde matrix V with $x_j \neq x_k, k \neq j$. Hint: Use mathematical induction.

Homework 7 (4 pts) due Th 3/14/13. Prob. 3.7 from text.

Homework 8 (8 pts) due 5/2/13. Prob. 5.4 from text.

Homework 9 (4 pts) due 5/2/13. Prob. 5.5 from text.

Computer Homework

For most of these problems you should upload your mfiles to Blackboard. You may wish to use the `publish` command in MATLAB to create html files of your reports.

Computer Homework I (4 pts.) due Th 2/14/13. Problem 1.39 from Moler's text.

Computer Homework II (4 pts.) due Th 3/14/13. Problem 2.3 from Moler's text.

Computer Homework III (4 pts.) due Th 4/4/13. Prob. 3.9 from text on the Runge phenomenon.

...not assigned Spring 2013:

Computer Homework I-extra (6 pts.). Prob. 3.13 from the text; see also old Com puter HW III below for some possible extensions and hints.

Computer Homework II-extra (6 pts.). Program Simpson's rule. You may revise my short trapezoidal rule code `traptd2(f,a,b,n)` posted on my webpage. Compare your program with `traptd2` and `quadtx` for some selected know integrals, $\int_0^1 f(x)dx$ for $f(x) = \sqrt{x}, x^2, x^3, x^4$ and various n, e.g., $n = 10, 100, 1000, \ldots$. Tabulate and comment on your results in light of the error estimates in the posted notes on Newton-Cotes methods. Also try the integral for π in text problems 6.3 and 6.4.

Computer Homework III-extra (4 pts.). Prob. 2.19 from the text. (You may just hand in an orderly copy of the MATLAB commands. You might want to use the MATLAB `diary on` command. Type `help diary`. I do not want to see all the entries of the $n \times n$ matrix where $n = 100$ or the $n \times 1$ solution vector x!)

old Computer Homework IV-extra.

a) (2 pts) Fill in the ?'s in the matrix equation for the periodic cubic spline,

$$\begin{bmatrix} 2(h_{n-1}+h_1) & h_{n-1} & 0 & \cdots & 0 & 0 & h_1 \\ h_2 & 2(h_1+h_2) & h_1 & \cdots & 0 & 0 & 0 \\ 0 & h_3 & 2(h_2+h_3) & h_2 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & h_{n-2} & 2(h_{n-3}+h_{n-2}) & h_{n-3} \\ ? & 0 & 0 & \cdots & 0 & ? & ? \end{bmatrix}$$

$$\times \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-2} \\ d_{n-1} \end{bmatrix} = \begin{bmatrix} h_1\delta_{n-1}+h_{n-1}\delta_1 \\ h_2\delta_1+h_1\delta_2 \\ h_3\delta_2+h_2\delta_3 \\ \vdots \\ h_{n-2}\delta_{n-3}+h_{n-3}\delta_{n-2} \\ h_{n-1}\delta_{n-2}+h_{n-2}\delta_{n-1} \end{bmatrix}.$$

b) (4 pts) Problem 3.13 in text.

c) (bonus problem) Revise your code in part b) to interpolate points in the x, y-plane forming a closed curve using chordal arclength $h_k = \sqrt{(x_{k+1}-x_k)^2+(y_{k+1}-y_k)^2}$ as the spline parameter.

d) (bonus problem) Design, code, and test an efficient solver for the matrices above for periodic cubic splines. The operation count should be $O(n)$.

Computer Homework V-extra (2 pts). Reproduce Figure 5.4 from the text.

Final Exam and Final Computer Homework all due 10:00 AM, Thursday, 5/16/13.

Final Grades–drop lowest of four grades below

Written Homework	100 points
Computer Homework (including final HW (below)	100 points
Exam I	100 points
Final exam	100 points
Total	300 points

Final exam–Take-home exam worth 100 points. Do your own work. Hand in a hard copy of the Final exam on May 16, 2013.

1. Let $\omega_1 f(x_1)+\omega_2 f(x_2)$ be an integration rule approximating $\int_{-1}^{1} f(x)dx$.

   a.) (10 points) If we demand that the rule integrates $f(x) = 1$ and $f(x) = x$ exactly, what two equations do we get for $x_1, x_2, \omega_1, \omega_2$?

   b.) (10 points) If we set $x_1 = -1$ and $x_2 = 1$, what are ω_1 and ω_2? What is this simple rule called?

2. (20 points) Prob. 6.2 from text.

3. (20 points) Prob. 6.20 from text.

4. (20 points) Prob. 7.1 from text.

5. (20 points) Prob. 7.8 from text:

   a) Find J by a hand calculation.

   b) (bonus) Find λ using the symbolic toolbox.

Final Computer Homework

You may work individually or in teams of two or three people and turn in one copy for the team along with a print-out of your MATLAB code. (I will also let you upload your code and writeup to Blackboard.) Select one of the following (sets of) problems from Chapter 7 and do as much as you can with them: Problems are 7.9–13 (inclusive), Problems 7.15–16 (inclusive), or any one of Problems 7.21, 7,22, or 7.23. (Extra credit is possible for exceptional work or for completing more than one problem.)

If you are working alone, here are two an alternative problems:

1.) Program some of the Gauss quadrature rules (say for $m = 2, 3, 4$) from my posted notes and show that they work as advertized. (You could also compare with the trapezoidal and Simpson's rules.)

2.) Program the simple Runge-Kutta scheme from my posted notes and test it on a simple 1×1 equation, say $y' = ky$ and a 2×2 system, such as the predator-prey equation. (You could code classical R-K and compare the two methods.)

References

[DH] T. DeLillo and T. Hrycak, *A stopping rule for the conjugate gradient regularization method for inverse problems in acoustics,* J. Comput. Acoustics, 14 (2006), pp. 397–414.

[DIVW1] T. DeLillo, V. Isakov, N. Valdivia, and L. Wang, *The detection of the source of acoustical noise in two dimensions*, SIAM Journal of Applied Math., 61 (2001), pp. 2104–2121.

[DIVW2] T. DeLillo, V. Isakov, N. Valdivia, L. Wang, *The detection of surface vibrations from interior acoustical pressure*, Inverse Problems, 19 (2003), pp. 507–524.

[Hen] P. Henrici, *Essentials of Numerical Analysis*, John Wiley, New York, 1982.

[CM] Cleve Moler, *Numerical Computing with MATLAB*, SIAM, 2004.

[CVL] C. F. Van Loan, *Introduction to Scientific Computing*, Second edition, Prentice-Hall, 2000.

[LNT] L. N. Trefethen, *The definition of numerical analysis*, SIAM News, Nov. 1992
http://www.comlab.ox.ac.uk/nick.trefethen/home.html

[TB] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM, 1997.

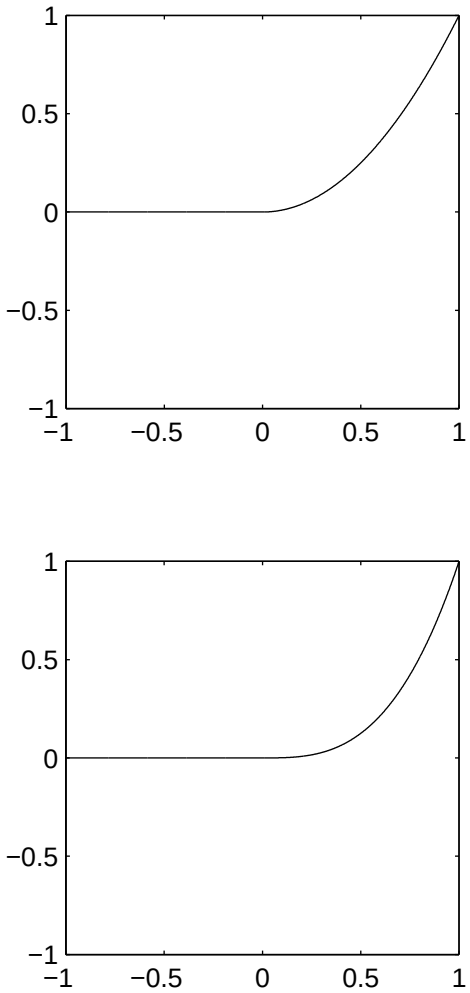

Figure 4: Illustration of C^1 function (top) and C^2 function (bottom).