Week 14: Wednesday, May 4

Logistics

- Project 3 is due Friday.
- Final exam: May 13 at 2:00 in Phillips 203. The format will be similar to the prelim exams. You may have the full 2.5 hours, and you may bring one sheet of notes (writing on both sides if you wish).
- Section this week will cover some of these questions (we didn't get through nearly all of them in class).

Summary

Error analysis and floating point

You should know about relative vs absolute error, forward error, backward error, residual, and condition numbers. You should remember the $1+\delta$ model for rounding, and be able to apply it in simple situations. You should know what underflow, overflow, and cancellation are.

Example questions:

1. What are the forward error and residual error for approximating the larger root of $f(x) = x^2 - 2$ by $\hat{x} = 1.5$? What is the condition number for this problem?
2. Which is more suitable for computation near $x = 0$: $\cos(x) - 1$, or $\sin(x)/(1 + \cos(x))$? Why?
3. Suppose I wanted to sum up numbers $z_1, \ldots, z_n$. A standard approach would be to write a loop to compute successive partial sums.

```
s = 0;
for j = 1:N
  s = s + z(j);
end
```

This loop really runs the recurrence $s_j = s_{j-1} + z_j$ starting at $s_0 = 0$. If I do this in floating point, how could I keep a running error estimate on the partial sums?

Linear algebra, linear systems, and least squares

You should know how to manipulate matrices and vectors in MATLAB, and have some notion of the relative costs of equivalent matrix expressions. You should know about the 1-norm, 2-norm, and infinity norm; about induced operator norms; and about the Frobenius norm. You should remember that the 2-norm (and the operator 2-norm and Frobenius norm) are invariant under orthogonal transformations, and that this can be used to simplify least squares problems. You should know the condition number for solving linear systems, and you should remember the factors that go into the sensitivity analysis for solving least squares problems (it's fine if you don't remember the exact formulas for the condition number in the latter case). You should know how to solve linear systems and least squares problems using MATLAB's backslash operator. You should know something about the normal equations, and about the relation between the solution, right hand side, and residual in a least squares problem. You should know the factorizations $PA = LU$, $A = QR$, and $A = U\Sigma V^T$. You should know what sparsity means.

Example questions:

1. Use norm bounds to show that the iteration $x_{j+1} = Ax_j + f$ converges if $\|A\| < 1$, and bound the magnitude of the limiting value.
2. Given $PA = LU$, write an $O(n^2)$ code fragment to compute $A^{-T}b$.
3. Describe a method to approximately minimize the sum of squared *componentwise relative errors* $d_j = (Ax - b)_j/b_j$.

Nonlinear equations and optimization

You should know about bisection, the Newton idea and its variants, and the concept of fixed point iteration. You should understand what is meant by rates of convergence (linear, superlinear, quadratic, etc). You should be able to reason about the error in iterations by subtracting a fixed point equation from the iteration equation; you should also be able to do Taylor series manipulations needed to understand these methods. For optimization,

you should know what a descent direction is, and what it means to do a line search. At this point, you should have some idea what Gauss-Newton does. I will not ask you about golden section search.

Example questions:

1. Suppose $f(x_*) = 0$ and we know $f'(x_*)$. Argue that the fixed point iteration
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_*)}$$
converges quadratically to x_* when started close enough.

2. Suppose $|f'(x)| \leq \alpha < 1$ for any x. Show that if $x_* = f(x_*)$ is a fixed point, then the iteration $x_{k+1} = f(x_k)$ converges at least linearly to x_*.

3. If we know f is continuously differentiable and $f'(a)f'(b) < 1$, and we have a routine to compute f', describe how to find a minimum or maximum on (a, b).

Interpolation, differentiation, and integration

You should know three different approaches to polynomial interpolation: the Vandermonde approach (writing the polynomial in a power basis); the Lagrange approach; and the Newton approach. You should know how Newton divided differences relate to derivatives, and how this relation can be used to provide error bounds on how well a polynomial interpolant approximates a function (assuming bounds on derivatives of various orders). You should know the basic ideas of piecewise polynomial interpolation, though I did not emphasize this in class and will not emphasize it on the final.

You should know how to derive rules for numerical differentiation and integration by either artfully canceling out terms in Taylor expansions (the method of undetermined coefficients) or by differentiating and integrating an interpolating function. You should understand what is meant by the order of a quadrature rule. You should understand the basic ideas of Gaussian quadrature, and why their order of accuracy $(2n - 1)$ is the greatest possible for any quadrature rule that involves function evaluations at n given points. You should also understand the idea of error estimates based on comparing different quadrature rules.

Example questions:

1. Write the interpolant through $f(0) = 1$, $f(1) = 2$, and $f(2) = 1$ in terms of the power basis, the Lagrange basis, and the Newton basis.

2. Suppose a_n and a_{2n} are two approximations to $\int_{-1}^{1} f(x)\,dx$ computed by the composite midpoint rule with n intervals and $2n$ intervals, respectively. Write an estimate of the error in a_{2n} based on comparing the two methods. Your estimate should be asymptotically exact as $n \to \infty$.

3. Given a quadrature rule

$$I_h[f] = \sum_{j=1}^{n} w_j f(x_j) \approx \int_a^b f(x)\,dx,$$

   give an example of a polynomial for which the quadrature rule cannot compute the true integral.

ODE solvers

You should know how to convert ODEs to standard first-order form for use with MATLAB's ODE solvers. You should be able to write something that makes use of MATLAB's ODE solvers (given a reminder of the basic calling sequence). You should know the formulas for forward and backward Euler, the implicit trapezoidal rule, and the midpoint rule (though I would probably give you a hint if the latter two appeared on the exam). You should know the basic ideas of consistency and zero stability of ODE solvers, and you should understand the basic idea of using pairs of methods for local error control (and how such local error control can still fail to yield good solutions in some cases). You should be able to reason about whether methods applied to the test problem $y' = \lambda y$ converge to zero for different values of $h\lambda$, and you should understand what is meant by the region of absolute stability for a method.

Example questions:

1. Describe how to solve the IVP $mu'' + bu' + ku = g(t)$, $u(0) = u_0$, $u'(0) = v_0$ using the MATLAB solver `ode45`. Remember that `ode45` has the calling sequence

   [tout,yout] = **ode45**(f,tspan,y0);

where `f` is a function that takes arguments `t, y`.

2. Consider the ODE
$$\begin{bmatrix} x \\ y \end{bmatrix}' = \begin{bmatrix} y \\ -x \end{bmatrix}$$
with initial conditions $x(0) = 1$ and $y(0) = 0$. The true solution to this problem is $x(t) = \cos(t)$, $y(t) = \sin(t)$, and so $r(t)^2 \equiv x(t)^2 + y(t)^2$ is equal to 1. Show that if we discretize the problem using forward Euler with fixed step size h, then $r_n^2 \equiv x_n^2 + y_n^2 = (1 + h^2)^n$.

Monte Carlo

You should understand the basic idea of expressing desired quantities as expected values, and you should understand how to compute error bars. You should also understand that $O(N^{-1/2})$ convergence is slow, but may beat the alternatives for high-dimensional problems. Any Monte Carlo questions on the exams will be fairly simple.

Example questions:

1. Write a Monte Carlo code (with error bars) to compute
$$\int_1^2 x^{-1}\,dx = \log(2).$$

2. How should I complete the following Monte Carlo code so that the main loop terminates when the error bars are below δ? Note: your modification should not end up taking $O(n^2)$ time where n is the number of trials.

```
ftotal  = 0;
f2total = 0;
n = 0;
while (n < 1000) || (errbar > delta)
  fX = f(rand(1));
  ftotal  = ftotal  + fX;
  f2total = f2total + fX^2;
  n = n+1;
  fmean = ...
  errbar = ...
end
```