Review Sheet for Final Exam Selected Solutions APPM/MATH 4650 Fall '20 Numerical Analysis

Instructor: Prof. Becker
solutions version 12/6/2020

Note: the final exam is cumulative, but will focus a bit more on recent material (ch 5 and 6) not tested on midterm 2. This review sheet does not cover the entire class (it only covers ch 5 and 6), so please review the midterm 1 and midterm 2 review sheets.

Chapter 5: IVPs and ODEs

1. Why don't we see higher-order Taylor methods in software much?

   Solution:

   Because they require the user specifying derivatives of f, and you can achieve similar results via Runge-Kutta methods which don't require derivatives of f

2. How many "steps" is RK4?

   Solution:

   Trick question: via the usual meaning of "steps" in ODE solvers, we'd say that RK4 (and all Runge-Kutta methods) are one-step methods. They do have several stages, so we can call them a *multi-stage* method if we want.

3. What's the big deal about an "embedded" RK formula?

   Solution:

   It allows us to estimate the error using the combination of the lower- and higher-order formula, and by the embedded property, these two formulae share many of the same calculations, so this is an efficient use of computation

4. What are the pros/cons of one-step methods vs multi-step methods?

   Solution:

   Multi-step methods are all "embedded" so error estimates are easy, and they are more efficient with computation (no intermediate steps to throw away). However, they work best with constant step-sizes; they can work with adaptive step-sizes but its more complicated and requires a bit of extra computation. One-step methods like RK4 work easily with adaptive time steps, but don't always work well with error estimates (they may require more work) unless you use special embedded formulas like Dormand-Prince or Runge-Kutta-Fehlberg. Another benefit of one-step methods is that under very mild conditions they are stable, which is not true for all multi-step methods.

5. How are Adams methods derived?

   Solution:

   Via interpolation. We didn't cover the exact details

6. What's the main difference between Adams-Bashforth and Adams-Moulton?

   Solution:

   AB are explicit, AM are implicit

7. What does it mean to say a method is implicit or explicit?

Solution:

An explicit method methods the update w_{i+1} is just a straightforward computation, like $w_{i+1} = w_i + hf(t_i, w_i)$ (that's forward Euler). An implicit method means we need to solve a root-finding problem to solve for w_{i+1} because it appears on both sides of the equation, as in $w_{i+1} = w_i + hf(t_{i+1}, w_{i+1})$ (that's backward Euler).

8. What's the idea of a predictor-corrector method?

   Solution:

   To use an explicit method to predict the value of w_{i+1} and then use that predicted value inside an implicit method and avoid the root-finding procedure

9. How might one solve for the update step of an implicit method?

   Solution:

   If you have the derivative of f, then Newton's method is an option. You could do the bisection method, but it's probably more common to do a fixed point iteration (especially if h is very small), or the secant method. We didn't talk about it much, but all these things apply to systems of equations, where we'd need to solve a vector root-finding problem (to be discussed in the 2nd semester of this class). Newton's method, the secant method (now one of many quasi-Newton methods), and fixed-point iterations all generalize to higher dimensions. The bisection method does not generalize to higher dimensions (at least not in a practical way; the "ellipsoid" method from the 1970s is kind of a generalization, but not practical).

10. How are backward differentiation (BD) methods derived?

    Solution:

    Via quadrature, as opposed to interpolation

11. What's so special about Adams and BD methods? Can't we create more?

    Solution:

    Yes, you can create some more (e.g., midpoint is not of these forms). But while it is easy to make many more methods that are consistent, most of them are not zero-stable. Our homework had an example of doing this with a backward differentiation method: we used quadrature to derive two different rules, but only one of them was zero-stable.

12. What types of higher-order ODEs can be recast as a system of first-order ODEs?

    Solution:

    All of them; any higher-order ODE (linear, nonlinear, etc.) can be recast as a system of first-order ODEs.

13. How does the book define "stability" of a numerical IVP method?

    Solution:

    That a "stable" method is one for which the output depends continuously on the input. It's basically the same as the notion of "zero-stability" which means that as the stepsize $h \to 0$, the numerical solution stays bounded.

14. What does it mean to say a method is "consistent"?

    Solution:

    It means that the local truncation error goes to 0 as $h \to 0$. That is, the numerical scheme is consistent with the ODE we're solving; i.e., it agrees on a local level.

15. What's the local truncation error (LTE)?

Solution:

There are a few different ways to say this, and different formula depending on whether its a one-step or multi-step method. But either way, the idea is the following: if $w_i = y_i$ is exact, then the LTE is $(y_{i+1} - W_{i+1})/h$.

16. If the LTE is $O(h^3)$, what kind of error would we expect the global error to be?

Solution:

We'd expect it to be also $O(h^3)$. The extra $1/h$ builtin to the definition of the LTE accounts for the fact that we're going to add up $1/h$ of these local errors.

17. For finite difference methods for derivatives, we worried a lot about roundoff error. ODEs involve derivatives. Are we really worried about roundoff error for ODE solvers?

Solution:

Yes and no. In short, we're not worried as much. First, the main update formulas are really better thought of as integration, $y_{t+1} = y_t + \int_{t_i}^{t_{i+1}} f(t, y(t))\,dt$, and quadrature is stable. We can have issues (recall Theorem 5.10 which gives a formal statement), but usually our stepsizes h are not small enough to make it a big deal. For reasonable problems, if we took stepsizes h small enough to see major roundoff error, then we'd probably have a bigger issue that the computation would take a very long time. For finite difference methods, there was no computational slowdown associated with taking h extremely tiny, but for IVP solvers there's a big computational slowdown.

However, we do worry somewhat. For example, if a method is not stable (zero-stable), then even an infinitesimal amount of error will lead to an unusably bad solution.

18. What does "convergence" mean?

Solution:

This is the property we want: that we can approximate the true solution $y(t)$ as accurately as we like by just reducing the stepsize h sufficiently small (neglecting roundoff error).

19. How are stability, convergence and consistency related?

Solution:

The equivalence theorem says that a consistent method is convergent iff it is stable. Requiring a method to be consistent makes sense, otherwise its not even approximating the right equation. We don't talk about inconsistent methods.

20. How do we check for the stability of a one-step method?

Solution:

We don't check usually, since most reasonable methods are stable (under appropriate Lipschitz assumptions)

21. How do we check for the stability of a multi-step method?

Solution:

Check the root condition. This is completely independent of the hf_i terms.

22. Is "absolute stability" just "stability" with absolute values? What is it?

Solution:

No, it's not "stability" with absolute values at all. Our older notion of stability (basically zero-stability) was about what happens as $h \to 0$. Absolute stability is about, if you're solving the Dahlquist test equation $y' = \lambda y$, whether your numerical solution goes to 0 as $T \to \infty$ when you solve in the interval $t \in [0, T]$.

23. What does it mean to have a "stiff" equation? Why is absolute stability discussed in this chapter?

Solution:

A stiff equation has timescales of different orders. The slow timescale means we need to solve for $t \in [a,b]$ with $b-a$ very large, hence we need a large stepsize h otherwise the computation will take too long. The fast timescale may operate on scales faster than h, so we miss the details of the fast solution, but we'd at least like it not to cause our solution to blowup. That is, we may not be able to "resolve" the fastest time scale, but often this solution goes away (e.g., for $e^{\lambda t}$ solutions with $\lambda < 0$) so we want to make sure it doesn't ruin everything even after its gone. Hence we worry about a fixed stepsize (not $h \to 0$) and taking $b \to \infty$, as in absolute stability.

24. Are some methods absolutely stable while others are not?

Solution:

No, it's not that simple. For plan "stability", some methods are stable and others are not, and we don't often see the unstable methods. But for absolute stability, it all depends on context. We talk about whether a method is absolutely stable at a given $h\lambda$, where $h > 0$ is the stepsize, and $\lambda \in \mathbb{C}$ represents a component of the form $y = e^{\lambda t}$ that is present in the true ODE solution.

There are cases where our ODE doesn't have any terms like $y = e^{\lambda t}$; in this case, we can fall back to heuristics, like preferring methods that have overall larger regions of stability.

25. What is the region of stability? How do we calculate it?

Solution:

For a given method, it is the set of $h\lambda$ values for which the numerical method is absolutely stable. To calculate it, we form the appropriate characteristic polynomial, and then for the value of $h\lambda$, calculate all the roots (for a one-step method, there's only a single root), and check if all the roots are < 1 in absolute value (or complex modulus).

Chapter 5: IVPs and ODEs

1. What kind of linear equations did we mostly discuss solving (over-determined? under-determined? square? consistent? singular?

Solution:

We talked about square systems (n equations, n variables). They could be consistent or inconsistent (that is, there may or may not be a solution – Gaussian elimination may fail), and they could be singular (meaning A^{-1} doesn't exist) or not.

2. True or false: if A is a square matrix, then we can solve $Ax = b$ iff A is invertible

Solution:

False; it's true that if A is invertible, then we can always solve $Ax = b$; but the "only if" part isn't true, as this depends on b and whether its a consistent equation. If A is invertible, then we can solve $Ax = b$ for *all* b. But if A is singular, we can solve for some b. For example, no matter what A is, if $b = 0$, there's always a solution ($x = 0$).

3. Let

$$A = \begin{bmatrix} | & | & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \\ | & | & | \end{bmatrix}, \quad B = \begin{bmatrix} — & \mathbf{b}_1^T & — \\ — & \mathbf{b}_2^T & — \\ — & \mathbf{b}_3^T & — \end{bmatrix},$$

i.e., A has columns $\mathbf{a}_i$ and B has rows $\mathbf{b}_j^T$. Write down an expression for AB in terms of $\mathbf{a}_i$ and $\mathbf{b}_j$.

Solution:

It is $AB = \sum_{i=1}^{3} \mathbf{a}_i \mathbf{b}_i^T$.

4. For A and B defined as above, what is the $(i,j)^{\text{th}}$ entry of BA?

Solution:

It is $(BA)_{i,j} = \mathbf{b}_i^T \mathbf{a}_i$.

5. What's the complexity of matrix multiplication of a $n \times n$ matrix times a vector?

   Solution:

   It's $O(n^2)$

6. What's the complexity of matrix multiplication of a $n \times n$ matrix times another matrix of the same size?

   Solution:

   It's $O(n^3)$ using standard algorithms, though it is about $O(n^{2.83}$ via Strassen's algorithm which is sometimes, though still rarely, used for large matrices.

7. What's the complexity of performing Gaussian elimination on a $n \times n$ matrix times?

   Solution:

   It's $O(n^3)$.

8. What's the complexity of solving a linear system of equations if the matrix has a triangular structure?

   Solution:

   It's $O(n^2)$ (use forward or backward substitution).

9. How do we use the LU factorization to solve a system of equations?

   Solution:

   If $A = LU$, we can solve $Ax = b$ by solving $Ly = b$ and the $Ux = y$, and the L and U solves are done efficiently by forward and backward substitution in $O(n^2)$ time, resp.

10. How do we use a pivoted LU factorization to solve a system of equations?

    Solution:

    If $PA = LU$ where P is a permutation, we can solve $Ax = b$ by writing it as $PAx = Pb$, where Pb is a permutation of b and can be done in $O(n)$ time instead of $O(n^2)$ time. Then do the same forward/backward substitution trick except on Pb instead of b.

    If your pivoted LU factorization gives you $A = PLU$, you can write this as $P^T A = LU$ since $P^T = P^{-1}$ for a permutation matrix.

11. True/false: Numerical methods only exchange rows in the LU factorization if they have a 0 pivot

    Solution:

    False, they do it in more circumstances, since Gaussian elimination without any advanced pivoting rules is very unstable.

12. Do we need to pivot when computing an LDL factorization? when computing a Cholesky factorization?

    Solution:

    For LDL, we still need to pivot; the advantage over LU is mainly just ≈ 2 factor of time savings. For Cholesky, we do not need to pivot, and it's stable.

13. What is the standard condition number of matrix?

    Solution:

    It's the ratio of the largest to the smallest singular value; or equivalently, $\|A^{-1}\| \cdot \|A\|$ where $\|A\|$ is the spectral norm.

14. Why would someone write a blocked LU factorization code?

Solution:

To exploit level-3 BLAS, which is very efficient

15. Let A be a $n \times n$ matrix. If your code calls $A\mathbf{b}_i$ for $i = 1, 2, \dots, n$, this is $O(n^3)$ flops. If your code instead forms the matrix B with columns $\mathbf{b}_i$ and then multiplies AB, this is also $O(n^3)$. Is there a difference in speed in practice? Why?

    Solution:

    Yes, the second strategy sometimes leads to huge savings in time, and level-3 BLAS is much more efficient than repeated calls to level-2 BLAS. Why might this be? It's not all about flops, it's about what the code has to work with and optimizations it can make, taking into consideration not just flops but memory movement. This is exactly the kind of stuff we did not talk about in the class, partly because it's complicated and out-of-scope, and partly because we don't have to talk about it because as long as our BLAS is optimized, we can write most linear algebra in terms of level-3 BLAS and automatically benefit.

16. Your friend Tara needs to solve $A\mathbf{x}_i = \mathbf{b}_i$ for $i = 1, 2, \dots, 1000$. Rather than solve the systems independently, they cleverly precompute $B = A^{-1}$ once, and then can solve for $\mathbf{x}_i$ via $\mathbf{x}_i = B\mathbf{b}_i$ efficiently. What's a better way?

    Solution:

    The idea of precomputation is good, but let's use LU instead of the inverse, as we saw in the HW that it's a bit faster and a bit more accurate. So precompute $A = LU$, and now solve via backsubstitution for each i (also $O(n^2)$ like $B\mathbf{b}_i$).