Review for Exam 2

Ben Wang and Mark Styczynski

This is a rough approximation of what we went over in the review session. This is actually more detailed in portions than what we went over.

Also, please note that this has not been proofread fully, so it may have typos or other things slightly wrong with it, but it's a good guide towards what you should know about.

Finally, be familiar with what is in your book so that if there's something on the exam that you don't know well (e.g., LU decomposition from the last exam), you at least know where to look for it.

Chapter 4: IVPs

Objective: develop iterative rules for updating trajectory with the objective of getting numerical solution to equal to the integration of the exact solution

Converting higher-order differential equations into systems of coupled first-order differential equations.
What is quadrature?
Numerical integration will involve 'integrating' a polynomial → that is why polynomial interpolation is important
Polynomial interpolation:
Polynomial interpolation is accomplished by defininin N support point which the polynomial will equal the function at. We define the polynomial:

$$p(x) = a_o + a_1 x + a_2 x^2 + ... + a_N x^N$$

such that

$$p(x_j) = a_o + a_1 x_j + a_2 x_j^{\ 2} + ... + a_N x_j^{\ N} = f(x_j)$$

for j = 0,1,2,…N

You can solve this like way back in the day with a linear system. But let's find a better way of doing this.

Lagrange interpolation
For N support points, Lagrange interpolation is the sum of N lagrange polynomials, each of which are of order x^{N-1}, designed to fit closely (exactly?) at a specified support point.

Now that we have a sum of polynomials or a single polynomial, we can integrate this numerically, using trapezoidal rule, Simpson's rule etc.

Integration: Newton-Cotes → uniformly spaced points

Different methods: trapezoid rule, simpson's rule

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{2}\left[f_0 + f_1\right] \qquad O\left(|b-a|^3\right)$$

with errors

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{6}\left[f_0 + 4f_1 + f_2\right] \qquad O\left(|b-a|^4\right)$$

How to get more accurate?

Break into smaller segments before making too many more points

How to integrate in two dimensions?

Make a rectangle containing all points, and multiply the function you are integrating by an "indicator" function that says whether you are in the area to be integrated (indicator = 1) or not (indicator = 0)

This all relates to time integrals

Linear ODE systems and dynamic stability:

Linear system, begin with a known ode problem x_dot = Ax. You can expand the exponential solution to analyze the results

Looking at a linear system which gives rise to a discussion on stability and behavior of solution.

Nonlinear ODE systems:

Same type of anlaysis, now we use a Taylor expansion, invoking the Jacobian.

Time marching ODE-IVP solvers:

Explicit methods: generates a new state by taking a time-step based on the function value at the old time-step.

Explicit Euler method

Uses Taylor approximation, gives first order error globally (second order local, but O(delta t) time steps, so O(delta t) error)

$$\underline{x}^{[k+1]} = \underline{x}^{[k]} + \Delta t \cdot \underline{f}\left(\underline{x}^{[k]}\right)$$

Runge-Kutta methods

Still explicit, but uses midpoints

Second-order RK:

$$\underline{k}^{[1]} = \Delta t \cdot \underline{f}\left(\underline{x}^{[k]}\right)$$

$$\underline{k}^{[2]} = \Delta t \cdot \underline{f}\left(\underline{x}^{[k]} + \frac{1}{2}\underline{k}^{[1]}\right)$$ where the global error is (delta t)^2\

$$\underline{x}^{[k+1]} = \underline{x}^{[k]} + \underline{k}^{[2]} + O\left((\Delta t)^3\right)$$

Can get higher order ones.

RK 4th-order is basis of ode45

4th order RK → you just have to know the value of the function at the current state and a time step and you can generate the new state.

How does ode45 get estimate of error?

Use 5 function evaluations to get 5th order accurate, also gets 4th order accurate, compare two to see how accurate you are → then know if need to make delta_t smaller or bigger

Global vs. Local error

Implicit methods:
update rule depends on previous values of x at time in the immediate past. Use of interpolation to approximate an update.

Why is it difficult to use $\underline{x}^{[k+1]}$ to get your derivative?

Because that is nonlinear, requires a difficult solution

Is this more accurate than explicit in all cases?

No, same order-accuracy for explicit Euler vs. implicit Euler

Are these methods used?

Yes, quite frequently.

Why or why not?

Because they are more stable to larger time steps for stiff equations… they won't "blow up".

Basic implicit Euler: $\underline{x}^{[k+1]} = \underline{x}^{[k]} + \Delta t \cdot \underline{f}\left(\underline{x}^{[k+1]}\right)$

Others, like Crank-Nicholson: $\underline{x}^{[k+1]} = \underline{x}^{[k]} + \Delta t \cdot \frac{1}{2}\left[\underline{f}\left(\underline{x}^{[k+1]}\right) + \underline{f}\left(\underline{x}^{[k]}\right)\right]$

Predictor/corrector method:

Use explicit method to generate an initial guess

Then use Newton to iteratively find real solution to implicit equation

Predictor/corrector method ought to converge well for Newton?

Yes, because guess should be relatively near correct value

DAE systems:

What are they?

$$\underline{\dot{y}} = \underline{F}\left(\underline{y}, \underline{z}\right)$$
$$\underline{0} = \underline{G}\left(\underline{y}, \underline{z}\right)$$

where y are variables in ODEs, z are other variables

Mass matrix: indicates where the differential parts are

Is mass matrix singular?

If it's a DAE, yes. Otherwise, it's just an ODE system.

What conditions do we need to solve this using an ODE solver?

DAE system to be of index one: the determinant of the Jacobian of the nonlinear equalities must be nonzero.

So that leaves us with:

$$\frac{\partial G}{\partial \underline{y}^T}\dot{y} + \frac{\partial G}{\partial \underline{z}^T}\dot{z} = 0$$

Stability of systems' steady states:

Take Jacobian

Evaluate at steady state

Find eigenvalues

If real part of all is < 0, stable

If <=0, quasi-stable (indifferent to perturbations in some direction)

If any > 0, unstable

Don't be confused by stability of integrators and stability of a system's steady state.

The integrator creates a "system" based on its solution method, and we evaluate that system's stability.

Which ODE integration routines are more stable?
    Explicit RK vs. Implicit Euler
Which will have less error in a non-stiff system?
    Explicit RK vs. Implicit Euler

Stiff/not stiff systems:
    Eigenvalues of Jacobian

    Definition
    Conceptual (different time scales)
    Condition number: indication of number of steps needed to integrate properly
    When to use ode15s vs. ode45?
        Condition number >>1 or not
    When will you know things will be stiff/not stiff?
        Single ODE equation: not stiff
        Discretized PDEs: stiff
    What to do if not sure?
        Try ode45, if fails, use 15s

Ways to speed things up in Matlab?
    Return Jacobian of system of ODEs at the supplied point.

Chapter 5: Optimization
Use the gradient of the cost function to find the downhill direction you may want to look in
Use the Hessian of the gradient to figure out how close to that gradient direction you want to look, and exactly how far you'd want to go if local conditions stayed constant globally
This is using (essentially) a Newton method in multiple dimensions to find the optimum
Analogy to solving nonlinear algebraic systems
Adding to the Hessian matrix: H → B
    Add to the main diagonal to make sure matrix is positive definite
        (an application of Gershgorin's theorem)
    What does this do for us?
        Ensures that we are at least going in the right direction
Then we figure out just how far in final direction we want to go:
    Alpha/2 until we can get a decrease in function value.
Gradient methods
Newton line search methods to find better updates for p and step lengths.
What about constraints?
    Use method of LaGrange multipliers
    Require not that we find local minimum of solution, but that we find place where local curvature of cost function is parallel to local curvature of constraint function
Why not just use "penalty" function?

As penalty weight gets too big, it becomes even more important than the actual minimization, so it becomes a problem that is numerically difficult to solve.

LaGrangian method converts problem into **sequence of unconstrained minimizations**, or relaxations.

Inequality constraints: use slack variables to make equality constraints.

For a minimum, the Hessian should be positive-definite.

Lagrangian methods: Penalty method, Lagrange multipliers

Equality constraints

Inequality constraints

Chapter 6: PDE-BVPs

Finite difference method

So that will give us a linear system of equations for simple problems… but in 2-D or 3-D, fill-in makes Gaussian elimination an unacceptable option.

For symmetric matrices, can turn into a minimization problem… but don't use Hessian, just use identity matrix → method of steepest descent

But that may be slow to converge at times, so can add in some term that will make it not go directly downhill but will make sure it doesn't double back on self → "conjugate gradient method"

In N-D, solves in N iterations or less

Utilizes the fact that when Ax=b, $F\left(\underline{x}\right)=\frac{1}{2}\underline{x}^T A\underline{x}-\underline{b}^T\underline{x}$ is minimized

If matrix is not symmetric, can do a similar thing but slightly more general→ "general minimized residual method"

10.50-ish stuff:

Dirichlet BCs (e.g., Ca = constant)

von Neumann BCs (e.g., dCa/dx = constant)

Uneven gridpoint spacing

How to do it (LaGrange interpolation)

Why to do it (non-linear action, or action in a small portion of the graph)

Danckwerts BC's

$$v_z\left[C_j\left(0\right)-C_{j0}\right]-D\left.\frac{dC_j}{dz}\right|_{z=0}=0$$

What it does (allows for both von Neumann and Dirichlet boundary conditions to be possible and/or balanced)

Outlet on end of reactor: BC?

If know nothing better, assume reaction is done (dCa/dz = 0)

Convection:

Central Difference Scheme: better for error

Upwind Difference Scheme: better for stability

Variable stacking:

Put things that will be interacting near each other in the matrix

Things to note:

- how to reduce order of differential equation so that you can solve a system of first order odes
- Euler's formula
- How to calculate Jacobian
- How to calculate eigenvalues
- Keep in mind Geshorgin's theorem
- Symmetric Matrices
- Positive definite or Negative Definite property of matrices

Useful Matlab functions:
**interp1
**trapz
**quad
**ode45
**ode15s
**fmincon
**fminunc
** optimset('JacobPattern',matrix)