Mathematics Qualifying Exam: Study Guide
Tufts University
Fall 2021 - Spring 2023
Mitchell Scott (mscott04)

1 Numerical Analysis/ Linear Algebra

1.1 Systems of Equations

1.1.1 Linear Systems of Equations

Gaussian elimination

LU decompositions for full and sparse matrices

Cholesky for full and sparse matrices

Operation Counts

Stability of Linear Systems (Condition Number)

Stability of Gaussian elimination

Basic Iterative Methods The basic goal of iterative methods is to solve $A\vec{x} = \vec{b}$, but by splitting of A:

$$A = M - N \tag{1}$$
$$A\vec{x} = M\vec{x} - N\vec{x} \tag{2}$$
$$M\vec{x} = \vec{b} + N\vec{x} \tag{3}$$

But this leads to the questions, what is M? Well matrix M is typically chosen so that the linear system $M\vec{z} = \vec{f}$ is "easily solvable" for any $\vec{f}$ either because it becomes a special type of matrix like M might be diagonal, triangular, or triangular. It is also important to note that M must not be singular, because then it isn't invertable, which is crucial in solving this problem.
This means that we start with an initial guess and slowly refine it to get closer and closer to the actual answer. We set up the iterative system

$$Mx_{(k+1)} = b + Nx_{(k)} \tag{4}$$

This means we can take the general formula $x_{k+1} = Gx_k + c$, where $G = M^{-1}N$ and $c = M^{-1}b$. This has the Jacobian matrix that is

$$G(x) = M^{-1}N \tag{5}$$

and to see if the iteration scheme is convergent if the spectral radius (recall that is the largest in magnitiude eigenvalue)

$$\rho(G) = \rho(M^{-1}N) \tag{6}$$
$$< 1 \tag{7}$$

The smaller $\rho(G)$ is, the faster we see convergence.
You see though, we have a choice on what $\mathbf{M}$ and $\mathbf{N}$ are so that we get $\rho(\mathbf{M}^{-1}\mathbf{N})$ to be as small as possible. As with all things we care about, there is a trade off between the cost of solving the linear system with $\mathbf{M}$ and how many iterations we care to take. The example is if $\mathbf{M} = \mathbf{A}$, then when we solve it, it can be done in one iteration, but that is a direct solve which can be very computational expensive.

Jacobi Like I said before, we normally pick M such that it has a special property to make it, nicely invertable and with a small norm. One way to do this is with the Jacobi Method. This is where we have A

$$A = D + L + U \tag{8}$$
$$M = D, \quad N = -(L + U) \tag{9}$$

This means that we get the iterative system

$$x^{(k+1)} = \mathbf{D}^{-1}\big(\vec{b} - (\mathbf{L} + \mathbf{U})x^{(k)}\big) \tag{10}$$

Now if we look at the overall solution components as opposed to the vectorized code, we get

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}}, \quad i = 1, 2, ..., n \tag{11}$$

- **CON:** It is easy to see that we need to double store x here, because we are using all of the old values to record the new values. Or, there is no update to x so we need to let it go all the way through and then we can update it for the next iteration.
- **CON:** Its convergence rate is rather slow
- **CON:** It is not guaranteed to converge, but
- **PRO:** It normally converges given standard practice. (e.g. if the matrix is diagonally dominant by rows)

Gauss - Seidel We can forgive a lot, but a slow convergence is not something that we want to forgive. It is so slow because it doesn't take advantage of the benefit of the new information available. However, Gauss-Seidel does this by using the new component of the solution as soon as it is completed. In terms of matrix calculations, we see that

$$\mathbf{M} = \mathbf{D} + \mathbf{L} \tag{12}$$
$$\mathbf{N} = -\mathbf{U} \tag{13}$$
$$x^{(k+1)} = \mathbf{D}^{-1}\big(\vec{b} - \mathbf{L}x^{(k+1)} - \mathbf{U}x^{(k)}\big) \tag{14}$$
$$= (\mathbf{D} + \mathbf{L})^{-1}\big(\vec{b} - \mathbf{U}x^{(k)}\big) \tag{15}$$

Or if we care about the specific elemental components, we could get

$$x_i^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)}}{a_{ii}}, \quad i = 1, ..., n \tag{16}$$

- **PRO:** Faster Convergence than Jacobi
- **PRO:** Don't have to keep two versions of $\vec{x}$, as we are constantly overwriting it.
- **CON:** Has to do these recursively, and we can't use this on a parallel computing system.
- **CON:** Doesn't always converge, but
- **PRO:** Converges under most practical applications, and less required than Jacobi (e.g. SPD matrix).
- **CON:** Still kind of slow to converge.

Successive Overrelaxation method

Conjugate Gradient Method

1.1.2 Eigenvalue Problems

Gerschgorin theorem

power method

inverse power method

stability of eigenvalue problems

1.1.3 Nonlinear System of Equations, Optimization

Newton's Method

Quasi-Newton Methods

Fixed Point Iteration

Newton and Levenberg-Marquardt Methods for Unconstrained Optimization

1.2 Numerical Aproximation

1.2.1 Interpolation

Interpolating polynomials

Lagrange Interpolating Polynomials

Hermite Interpolating Polynomials

Runge phenomena

Splines

least squares approximation of functions and orthogonal polynomials

1.2.2 Integration

Newton-Cotes methods

Gaussian quadrature

Euler-MacLaurin formula

Adaptive quadrature

1.2.3 Differential Equations

Convergence of explicit one-step methods

Stiffness

A- stability

Impossibility of A-stable explicit Runge-Kutta methods

1.3 References

1. An Introduction to Numerical Analysis by K. E. Atkinson (Wiley)
2. Unconstrained Optimization by P. E. Frandsen, K. Jonasson, H. B. Nielsen, and O. Tingleff
3. Analysis of Numerical Methods by E. Isaacson and H. B. Keller (Wiley, Dover reprint)
4. Finite Difference Methods for Ordinary and Partial Differential Equations by R. LeVeque (SIAM)
5. A First Course in the Numerical Analysis of Differential Equations by A. Iserles (Cambridge University Press)

2 Analysis

3 Algebra

4 Partial Differential Equations

5 Algebraic Topology

6 Geometry