Mathematics Qualifying Exam: Study Guide
Tufts University
Fall 2021 - Spring 2022
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 If $\mathbf{A}$ is SPD, then $\mathbf{LU}$ could become $\mathbf{U} = \mathbf{L}^T$.

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 While Gauss-Seidel is still better than Jacobi, it is still not fast. But there is a way to make it better, and that is called Successive Over-Relaxation (SOR). We first start each new iteration by computing where the Gauss- Seidel step would take us, $x_{GS}^{(k+1)}$, but we don't just step in that direction. Instead,

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \omega\big(\vec{x}_{GS}^{(k+1)} - \vec{x}^{(k)}\big) \tag{17}$$

$$= (1-\omega)\vec{x}^{(k)} + \omega\vec{x}_{GS}^{(k+1)} \tag{18}$$

Okay, but what does this ω do? What is the optimal value of ω? Your guess is as good as mine. bestie. The whole point of ω is to speed up the convergence of the GS method, so it's goal is to reduce $\rho(\mathbf{M}^{-1}\mathbf{N})$. I still don't know a robust way to do this.
It is also important to note that we only care about $\omega \in (0, 2)$ because outside of that range, we know that it will diverge. Look at equation 18, and I feel that this is self explanatory. We also know that when $\omega < 1$ gives us *under*-relaxation, and $\omega > 1$ gives us *over*-relaxation. When $\omega = 1$, we just get Gauss- Seidel. Does it matter if we are doing under or over relaxation?

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \omega\left(\mathbf{D}^{-1}\big(\vec{b} - \mathbf{L}\vec{x}^{(k+1)} - \mathbf{U}\vec{x}^{(k)}\big) - \vec{x}^{(k)}\right) \tag{19}$$

$$= \big(\mathbf{D} + \omega\mathbf{L}\big)^{-1}\Big(\big(1-\omega\big)\mathbf{D} - \omega\mathbf{U}\Big)\vec{x}^{(k)} + \omega\big(\mathbf{D} + \omega\mathbf{L}\big)^{-1}\vec{b} \tag{20}$$

This implies that the splitting of the matrix $\mathbf{A}$ is

$$\mathbf{M} = \frac{1}{\omega}\mathbf{D} + \mathbf{L}, \qquad \mathbf{N} = \left(\frac{1}{\omega} - 1\right)\mathbf{D} - \mathbf{U} \tag{21}$$

- **PRO:** Faster than Gauss- Seidel
- **CON:** No way to know when it will converge, also have to now consider what ω values will work with convergence.
- **CON:** Still a stationary method so not as fast as it could be.

Conjugate Gradient Method

1.1.2 Eigenvalue Problems

Sometimes we care about eigenvalues, but it would be too computationally expensive (especially for a large sparse matrix) to compute them accurately. The current algorithms for eigenvalue finding are $\mathcal{O}(n^3)$. So let's say we just want to know if our iterative method is going to converge, we want to check that the $\rho(\mathbf{A}) < 1$, but if we need to find them exactly, it would take the same amount of time to just solve it directly.
This is why getting rough estimates of the magnitude of eigenvalues would be helpful. There is the roughest, dirtiest way to do that which is

$$|\lambda| \leq \|\mathbf{A}\| \tag{22}$$

where the norm can be any induced matrix norm.

Gerschgorin theorem But maybe we don't want the roughest, dirtiest approximation. Meet Gershgorin's Theorem, sometimes called *Gershgorin's circle theorem.*

Theorem 1. *Let $\boldsymbol{A}$ be an $n \times n$ matrix. The eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$ are all contained in a union of n disks, where the i^{th} disk is centered at a_{ii} with radius $\sum_{j \neq i} |a_{ij}|$.*

Proof. Let λ be any eigenvalue, with the corresponding normalized eigenvector $\vec{x}$, (this means that $\|x\|_\infty = 1$. Now take $\vec{x}_i$ which is an entry of $\vec{x}$ such that $|x_i| = 1$, and we are guraneteed to have one because of the infinity norm condition that we set in place.
Recall $\mathbf{A}\vec{x} = \lambda\vec{x}$, now we have

$$(\lambda - a_{ii})x_i = \sum_{j \neq i} a_{ij}x_j \tag{23}$$

Now using the Triangle inequality, we get

$$|\lambda - a_{ii}| \leq \sum_{j \neq i} |a_{ij}| \cdot |x_j| \tag{24}$$

$$\leq \sum_{j \neq i} |a_{ij}| \tag{25}$$

We know that $|x_j| \leq 1$, because we have the infinity norm of 1, so everything has to be less than or equal to the infinity norm. □

Power Method Sometimes we don't care about all of the eigenvalues, but maybe just one (think spectral radius). There is a way to find this eigenvector/eigenvalue called Power Iteration, which multpilies an arbitrary nonzero vector by a matrix over and over again $(\mathbf{A}, \mathbf{A}^2, \mathbf{A}^3, \cdots)$
The algorithm for this is inout $\vec{x}_0$, an arbitrary nonzero vector. Then iterate k times so that $\vec{x}_{k+1} = \mathbf{A}\vec{x}_k$. However, this is assuming that $\exists!\lambda_1$ of maximum modulus with corresponding $\vec{v}_1$. If that is the case, then this method will result in a vector $\vec{x}$ that is a multiple of $\vec{v}_1$.
This method is cool

- **PRO:** Almost always works in practice, but not guarenteed.

- **CON:** Fails when $\vec{x}_0$ has no component in dominant eigenvector $\vec{v}_1$. This can be reduced by picking a random starting vector, or because computational arithmetic is inexact, normally there is some rounding in that vector component which will introduce a component.
- **CON:**This analysis was done assuming that there is a unique largest eigenvalue, but what about $\lambda = \pm 1$? They are a complex conjugate pair.
- **PRO:** If we start with a real matrix and a real starting vector, we will always get a real convergent sequence.
- **CON:** This method can lead to overflow/underflow because we are just multiplying something over and over again, so maybe we can get rid of this risk?

In fact, we deserve better, so that is where the normalized Power iteration comes in.
Just like before, we start with $\vec{x}_0$, an arbitrary nonzero vector. We introduce a loop where $\vec{y}_k = \mathbf{A}\vec{x}_{k-1}$ and then we normalized $\vec{x}_k = \vec{y}_k / \|\vec{y}_k\|_\infty$. This way every iteration is normalized!

Inverse Power Method Sometimes we don't care about the largest eigenvalue, but rather the smallest. (We love our small kings!) This can be used in graph theory, like if the graph is bipartite or not. Or maybe you want to bound the conjugacy of $\vec{x}^T\mathbf{A}\vec{x}$, which is bounded below by λ_{min}.
We will use the fact that the eigenvalues of $\mathbf{A}^{-1}$ are the reciprocals of $\mathbf{A}$. This means that the smallest eigenvalues of $\mathbf{A}$ are the largest eigenvalues of $\mathbf{A}^{-1}$, and we already know a way to find the largest eigenvalues of a matrix.
This method is called the Inverse Power Method. The algorithm is start with an arbitrary nonzero vector. Then you want to loop over an iteration index solving $\mathbf{A}\vec{y}_k = \vec{x}_{k-1}$ for $\vec{y}_k$. Then you can normalize it so that $\vec{x}_k = \vec{y}_k / \|\vec{y}_k\|_\infty$.
It is also important to know that using a shift value offers far more potential for accelerating convergence, and can be used to get more eigenvalues than just the min and max one. This is super helpful in the inverse power iteration (why?). In particular, the eigenvalue of $\mathbf{A} - \sigma$
$textbf I$ of smallest magnitude is simply $\lambda - \sigma$, where λ is an eigenvalue of $\mathbf{A}$ closest to σ.

1. **PRO:** If the shift is close to an actual eigenvalue, convergence is super fast.
2. **PRO:** Can find ANY eigenvalue you want, not just the max and min.
3. **PRO:** Super useful in computing the eigenvector corresponding to an approximate eigenvalue that has already been obtained by some other mean.
4. **CON:** Still linear convergence (I mean smaller C but still)

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)