Math 4610 Exam II Questions - Fall 2022

Jacob Fitzgerald
A02261889

1. Concurrencent Process: Multiple logically independent processes are logically active at the same time. AKA multiprogramming. Parellel: Multiple processes are actually executing at the same time.
2. When datafitting with a larger matrix than the number of coefficients, the problem is said to be overfitted, and it's often hard if not impossible to find a solution. If it's smaller, you don't have enough information to find a solution. The optimal case is where the matrix is the same size as the number of coeficients being determined. In this case, if the rows are linearly independant, you can find a solution.
3. A direct method, such as gaussian elimination with back substitution, seeks to solve a linear system of equations algebraically. By reducing the matrix of equations to a uper triangular one, you can logically solve for the value of the variables one by one. A solution using this method is gaurenteed to be exact (not accounting for rounding error), but takes n^3 work to solve. If you don't want to use a pivoting strategy, the matrix should be diagonally dominant as well, otherwise your pivoting strategy may or may not yield results. Iterative methods, such as Jacoby, use consecutive guesses for the solution to aproximate it's value, and our not gaurenteed to converge. Certain properties must be present in the matrix (such as diagonal dominance for the Jacoby method) in order to gaurentee convergence. The work done in a single loop of the Jacoby method is around n ^ 2, so it will take less work if it converges in under n iterations. A convergent solution is also not gaurenteed to be close to the actual solution, just "close" to satsifying the system.
4. The residual vector represents the difference between the values vector (b), and the equations matrix applied to your approximation (AXi). This means that, as the residual vector approches zero, the equation matrix applied to your approximation approaches the desired result (b). This means the residual vector can be used as a decent approximation of error. However, a small residual only means the approximation is close to satsfying the system, not that it's close to the real solution. It's preferable to estimating the error using the difference in successive approximations when the system may have "flat sections", which will fool the algorithm into thinking it's converged, due to small differences in approximations, when in reality a solution does not exist at those points. The residual vector also only needs the information from the current iteration, not past iterations, which is a nice plus.
5. The normal equations are used to perform linear regression on a data set in order to determine a linear relationship between two sets of points, inputs and outputs. In the case of error convergence, where the relationship is usually geometric instead of linear, this can be done by taking the log of each data set, the step size and the approximate error, which should shift the exponent into a coefficient, allowing you to determine it with linear regression. The slope of the line will then correspond to the power on the error term.
6. Gaus-seidel is generally superior to the Jacoby method. They both converge for basically the same conditions, where the matrix in quesiton is diagonally dominant, with the Gaus-seidel having a few more very specific cases where it will also converge. Overall, they're approximatley equally robust. When it comes to efficiency, the gaus-siedel method will converge in fewer iterations than the Jacoby method. This means that, for the same number of iterations, you'll get a more accurate result out of the Gaus-seidel method. Since the work done per iteration is approximatley the same, the Gaus-seidel method will give more

accurate answers for less work. The exception comes when you're able to parrelellize your code. The Jacoby method allows for independent calculations of each vector componenet, meaning that you can benefit from additional cores up to the size of your vector. The Gaus-seidle method requires results from lower rows to calculate those in higher rows, making parreleization pretty much impossible. So, depending on the system the code is being run on, Jacoby could end up being faster for the same level of accuracy.

7. Given an initial guess for an eigenvector for matrix A, x, you can assume it will be represented by some linear combination of the eigenvectors of A, e.g. c1 * v1 + c2 * v2 + c3 * v3 + ... Assuming that we order the eigenvalues in order of magnitude, that is abs(l1) > abs(l2) > ... > abs(ln), we can say that l1 is the largest eigenvector in magnitude. If we take the action of A an arbitrary number of times, k, on x we get A^k * x. If we apply this to teh eigenvector form of x, we get c1 * A ^ k * v1 + c2 * A^k * v2 + ... Using the identity of eigenvalues, we can express this as c1 * l1 ^ k * v1 + c2 * l2 ^k * v2 + ... If we factor out l1 ^ k from every term we get l1 ^ k * (c1 v1 + c1 * (l2 / l1) ^ k + ...). As l1 is greater than every other lambda value in magnitude, as k approaches infinity, every other term besides the first aproaches zero, and A ^ k * x begins to equal l1 ^ k * c1 * v1. Essentially, with this repeated multiplication, we're able to filter out the largest lambda value of A.
8. The inverse power method allows you to find the smallest (in magnitude) eigenvalue in a matrix. When a matrix is inverted, it's eigen values are inverted as well. That is, if matrix A has eigenvalues l1, l2, l3, ..., matrix B = A^-1 has eigenvalues 1 / l1, 1 / l2, 1 / l3, ... This means that The largest in magnitude eigenvector of B is the smallest in magnitude eigenvalue of A. We can utilize the same concept of repeated multiplication to find this, however, inverting a matrix is expensive. Instead, we can solve the linear system of equations set up in our multiplication step, using methods like Jacoby iteration and Gaus-seidel. Overall, this is still more expensive than the normal power method, but not as expensive as inversion.
9. For a given matrix A, and a vector V, the raleigh quotient is given by (V^T * A * V) / (V ^ T * V). This is actually equivelent to an eigenvalue of A, given that V is an eigenvector of A. This can be shown via simple algebra. It's useful in the power method because it allows us to get an approximation of our eigevnalue using an approximation of our eigenvector, especially because, if we use a unit vector for V, V ^ T * V is just 1.
10. If we subtract a value from the diagonal elements of a matrix, all of it's eigenvalues are affected in the same way. That is, for B = A - vI, if A has lambda values l1, l2, ..., B will have lamba values l1 - v, l2 - v, ... This means that the smallest eigenvalue (in magnitude), is now the one that is cloest to v in value. Now, we can use the inverse power method to search for this value. Once we get our answer, we have to add back in the value of V in order to get the eigenvalue equivelant from A.
11. Moore's law states that the number of transitors used in a chip will approximatley double every 2 years. It still applies, but we've run into pretty much a "power wall", because of the way computer arcitechture works, the power usage will scale geometrically with the transitor count, meaning we've run into unsustainable power requirments for chips. Parrelell processing bypasses this by using chips with slower clock speeds in parrellel. If you have two chips with half the frequency, you're power draw is about 60% of the single chip, while having the same "theoretical" throughput.
12. Preprocessor Directives: Defining the number of cores, declaring a parellel region. Setting Cores: Calling omp_set_cores to requrest a specific number of cores. Getting Information: Calling functions like omp_get_cores() or omp_get_id() to determine the actual runtime OMP configuration. You can also set environmental variables when interacting with programs, to set things like the number of available threads.