10.34 Numerical Methods Applied to Chemical Engineering Fall 2015

Homework #2: Linear Algebra, Systems of Nonlinear Equations

Problem 1 (20 points)**.** An initial value problem (IVP) in linear time-invariant ODEs is given by

$$\frac{d\mathbf{x}}{dt}(t) = \mathbf{A}\mathbf{x}(t), \quad \mathbf{x}(0) = \mathbf{x}_0, \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$, and where $\mathbf{x}_0 \in \mathbb{R}^N$ is the initial condition. Suppose that $\mathbf{A}$ is diagonalizable as $\mathbf{A} = \mathbf{W}\boldsymbol{\Lambda}\mathbf{W}^{-1}$.

1. Determine the exact analytical solution to (1) by carrying out the following steps. Firstly, rearrange (1) to obtain a simpler IVP which describes the time evolution of the modified variables $\mathbf{y} := \mathbf{W}^{-1}\mathbf{x}$. Include appropriate initial conditions for $\mathbf{y}(0)$. Solve this IVP to obtain an analytical expression for $\mathbf{y}(t)$ at any time t. Lastly, use the definition of $\mathbf{y}(t)$ to obtain an analytical expression for $\mathbf{x}(t)$.

   HINT: when solving the simpler IVP for $\mathbf{y}(t)$, it might help to consider each equation in the ODE system separately.

2. The IVP (1) is said to be *unstable* if the solution grows in an unbounded fashion, *asymptotically stable* if it decays to the origin, and *neutrally stable* if neither of these happen. Given your exact analytical solution to (1), describe how you could use the eigenvalues and/or eigenvectors of $\mathbf{A}$ to characterize the stability of (1).

3. Consider the following batch chemical system which we wish to model similarly to (1): $A \rightleftharpoons B \rightarrow P$. The time evolution of the system is given according to the following model equations:

$$\frac{d[A]}{dt} = -k_1[A] + k_{-1}[B],$$
$$\frac{d[B]}{dt} = k_1[A] - (k_{-1} + k_2)[B],$$
$$\frac{d[P]}{dt} = k_2[B].$$

   The rate coefficients are $k_1 = 10^{-4}$ s^{-1}, $k_{-1} = 10^{-2}$ s^{-1}, and $k_2 = 10^{-3}$ s^{-1}. We are particularly interested in the situation where initially (time t_0), $[A] = 10^{-7}$ mol/L, $[B] = 0$, and $[P] = 0$.

   (a) Write an analytical solution for $[P]$ in terms of $[B]$.

   (b) Show how eigen-decomposition of the first two differential equations IVPs can be used to determine $[A]$ and $[B]$ as a function of time.

   (c) Write the entire set of equations in matrix vector form $\frac{d\mathbf{v}}{dt}(t) = \mathbf{M}\mathbf{v}(t)$ where $\mathbf{M}$ is a 3x3 square matrix and $\mathbf{v}$ is a column vector. What is the initial condition vector $\mathbf{v}(t_0)$?

   (d) Although it is convenient to transform the matrix $\mathbf{M}$ to a diagonal form that is not always possible. If $\mathbf{M}$ is singular does that imply it is not diagonalizable? Is $\mathbf{M}$ being singular a necessary condition for this system of differential equations to have a non-trivial steady state solution (other than $\mathbf{v}(t) = \mathbf{0}$)?

(e) Describe how to test if $\mathbf{M}$ is diagonalizable and fully explain your logic, and what criteria are necessary to apply. Additionally report a series of MATLAB®commands to execute such a test.

(f) In this case, $\mathbf{M}$ is not diagonalizable. You can perform a Schur decomposition to factorize $\mathbf{M} = \mathbf{UT}(\mathbf{U}^H)$ where $\mathbf{U}$ is an unitary matrix, and $\mathbf{T}$ is upper triangular. (Note $\mathbf{U}^H$ denotes complex conjugate transpose of $\mathbf{U}$.) Show how to use the Schur decomposition in the same fashion as the eigen-decomposition to change variables and form a new set of differential equations which is decoupled in at least one variable.

(g) Perform the Schur decomposition in MATLAB using the built-in function `schur`. Report $\mathbf{T}$ and $\mathbf{U}$. Give an analytical expression for $y_3(t)$ where $\mathbf{y} = \mathbf{U}^H\mathbf{v}$ in terms of $\mathbf{M}, \mathbf{T}, \mathbf{U}$ and/or the initial conditions. Explain the physical significance of y_3 and its behavior.

Problem 2 (15 points). Consider the problem of computing the eigenvalues and eigenvectors of a matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. That is, find any values of $(\mathbf{v}, \lambda)$ satisfying the nonlinear equation:

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{v}.$$

To uniquely define the eigenvectors, suppose this nonlinear equation is supplemented with an equation restricting the length of the eigenvectors:

$$\|\mathbf{v}\|^2 = 1.$$

In many applications, approximations for only a few of the largest or smallest eigenvalues and their corresponding eigenvectors are needed, and iterative methods are employed to find them. In this problem, you will explore application of the Newton-Raphson method to finding eigenvalue/eigenvector pairs.

1. Write a vector valued function of $\mathbf{v}$ and λ whose roots satisfy the two equations above.

2. Derive the Newton-Raphson step for this function. Write a short MATLAB script that applies this iterative map to a random 10×10 matrix to find an eigenvalue/eigenvector pair to within a tolerance of 10^{-8}. Use the built-in MATLAB routine "eig" to check your result.

3. Suppose Newton's method can be made to typically converge to different eigenvalue/eigenvector pairs after k iterations. Estimate the number of arithmetic operations required to find m eigenvalue/eigenvector pairs of $\mathbf{A}$.

4. Are there properties of the Newton-Raphson method that make it ill-suited to target specific eigenvalues, say the largest or the smallest in magnitude? Does it cause a problem if two eigenvalues are equal, or complex-conjugates?

5. In practice, the eigenvalues and eigenvectors of matrices are approximated using a method called Arnoldi iteration, and not the Newton-Raphson approach derived in the previous parts. The Arnoldi method is a much more sophisticated version of the classic power iteration for determining an eigenvector denoted $\mathbf{v}_{\max}$ corresponding to the largest magnitude eigenvalue of a matrix:
$$\mathbf{A}^k \mathbf{b} \to \mathbf{v}_{\max} \quad \text{as} \quad k \to \infty,$$
where $\mathbf{b}$ is an arbitrary column vector of length N. Given an approximation for $\mathbf{v}_{\max}$, derive an equation for the corresponding eigenvalue. Show that the power iteration will exhibit linear convergence to $\mathbf{v}_{\max}$ when $\mathbf{A}$ has a complete set of eigenvectors, the two largest eigenvalues of $\mathbf{A}$ are distinct, and $\mathbf{b}^T \mathbf{v}_{\max} \neq 0$.

6. Estimate the number of arithmetic operations needed to find an eigenvalue, eigenvector pair if the power method were truncated after a finite number of iterations, k. How does this compare with the Newton-Raphson approach you derived in the previous parts?

Problem 3 (15 Points)**.** This problem requires you to implement Newton's method in MATLAB. To assist you with this task portions of a working implementation have been provided in the function file `FunctionEvaluator.m`. This function, when given an input vector $\mathbf{x}$ returns a vector $\mathbf{f}(\mathbf{x})$ of function values of size $n =$ `length(x)` and a Jacobian matrix of partial derivatives $\mathbf{J} \in \mathbb{R}^{n\times n}$. This system has a unique solution of $\mathbf{f}(\mathbf{1}) = \mathbf{0}$ which you may find helpful for testing your code.

This system consists of an even number of a system of nonlinear equations. The number of equations and dimension of the input are equal.

$$
\begin{aligned}
f_1(\mathbf{x}) &= (1 - x_1)^2 \\
f_2(\mathbf{x}) &= 10(x_1 - x_2^2)^2 \\
f_3(\mathbf{x}) &= (1 - x_3)^2 \\
f_4(\mathbf{x}) &= 10(x_3 - x_4^2)^2 \\
&\cdots
\end{aligned}
$$

1. Complete the implementation of Newton's Method. Make sure that your subroutine detects convergence to an input tolerance, but also detects, and reports when something goes wrong with the algorithm, and cannot go into an infinite loop. Produce a table showing the steps taken by your implementation to reach a solution for the inputs $\mathbf{x} = [1.5, 1.5, 2, 2]'$, and $\mathbf{x} = [-1.5, 2.1, -1.9, 2.1, -1.9, 2.1, -1.5, 2.1]'$. The columns of your table should be: iteration number, norm of $\mathbf{x}$, and norm of objective value.

2. For the initial input of $\mathbf{x} = [1.5, 1.5, 2, 2]'$ produce two figures, each with 4 subplots. In the first figure, plot the function values $f(x_i)$ for $i = 1, \ldots, 4$ as a function of the first 5 Newton's method steps. In the second figure, plot x_i for $i = 1, \ldots, 4$ as a function of the first 5 Newton's method steps.

3. The code provided in `FunctionEvaluator` uses dense matrix storage. In this system significant performance increases are possible by utilizing the sparse nature of the Jacobian. Report the sparsity pattern, and bandwidth of the Jacobian matrix from the `FunctionEvaluator` function.

4. With the current sparsity pattern, or an improved variable ordering of your own devising, implement a new version of `FunctionEvaluator` called `SparseEvaluator` using a minimal number of vectors for Jacobian storage. Also, update your Newton's Method code to utilize the sparsely stored Jacobian. Verify the same results are found for the two inputs in the first part of this problem. Compare the time it takes to run your Newton's Method using `SparseEvaluator` to that using the original `FunctionEvaluator` for $n = 4$.

10.34 Staff 2010-2015: September 17, 2015