COL 726 Homework 4

Due date: Friday, 22 March, 2019

1. When $a_{11} > a_{22}$, the 2×2 diagonal matrix $\mathbf{A} = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix}$ has eigenvectors $\mathbf{q}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\mathbf{q}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Suppose $\mathbf{A}$ is perturbed to $\mathbf{A} + \delta\mathbf{A} = \begin{bmatrix} a_{11} + \delta a_{11} & \delta a_{12} \\ \delta a_{21} & a_{22} + \delta a_{22} \end{bmatrix}$. Using a linearized analysis similar to Heath §4.3, find the resulting change in the first eigenvector $\mathbf{q}_1 + \delta\mathbf{q}_1$, ignoring second-order terms. Conclude that eigenvectors may be ill-conditioned when the eigenvalues are close together.

   (Note: You may need to use the property that the eigenvectors remain normalized, i.e. $\|\mathbf{q}_1 + \delta\mathbf{q}_1\|_2 = 1$. What does this tell you about $\delta\mathbf{q}_1$, to first order?) [3 points]

2. (a) Give an $O(m^2)$ algorithm to compute the QR factorization of an $m \times m$ upper Hessenberg matrix $\mathbf{H}$ using Householder reflections. [1 point]

   (b) Suppose you have computed such a QR factorization. Give an $O(m)$ algorithm to compute the QR factorization of an $(m+1) \times (m+1)$ upper Hessenberg matrix whose upper-left block is $\mathbf{H}$. (Hint: Remember that Householder QR outputs a sequence of reflection vectors, rather than the full matrix $\mathbf{Q}$.) [2 points]

3. Suppose a nonsingular symmetric $m \times m$ matrix $\mathbf{A}$ with distinct eigenvalues is given as a "black box", which only allows you to compute $\mathbf{Ax}$ and $\mathbf{A}^{-1}\mathbf{x}$ for any vector $\mathbf{x}$.

   (a) Describe a iterative method that converges to an eigenvector of $\mathbf{A}$ corresponding to the smallest eigenvalue in absolute value. [1 point]

   (b) Modify your method to find an eigenvector corresponding to the kth smallest eigenvalue. Your method should not take more than $O(k^2 m)$ time per iteration (ignoring calls to the black box). [3 points]

4. Consider an $m \times m$ nonsymmetric matrix $\mathbf{A}$. Suppose you find an $m \times m$ orthogonal matrix $\mathbf{Q}$ such that $\mathbf{Q}^T\mathbf{AQ}$ is of the form $\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{0} & \mathbf{A}_{22} \end{bmatrix}$ where $\mathbf{A}_{11}$ is $n \times n$, $\mathbf{A}_{22}$ is $(m-n) \times (m-n)$, and the lower-left $(m-n) \times n$ block is zero. Prove that the eigenvalues of $\mathbf{A}_{11}$ and of $\mathbf{A}_{22}$ are also eigenvalues of $\mathbf{A}$.

   (Note: This is related to the idea of "deflation" in the practical QR algorithm, which allows the matrix to be subdivided into smaller parts when possible.) [3 points]

5. The polynomial $p(x) = x^3 - x^2 - x + 1$ has a simple root $x = -1$ and a double root at $x = 1$.

   (a) Give the iteration function $x_{k+1} = g(x_k)$ for Newton's method applied to the equation $p(x) = 0$. Perform four iterations each starting from $x_0 = -2$ and $x_0 = 2$ (whether by hand or by code), and report the iterates x_1, x_2, x_3, x_4. [1 point]

(b) What is the theoretical rate of convergence at each root? If the convergence is linear, also give the constant. [1.5 points]

(c) Prove that the same convergence behaviour you showed for $x = 1$ is true for any double root of an arbitrary function $f(x)$. Then suggest, with justification, a simple way to improve the convergence of Newton's method near a double root. [1.5 points]

6. Solving a system of n nonlinear equations in n variables can be seen as finding a point of intersection of n hypersurfaces in $\mathbb{R}^n$. Here we will do the converse for $n = 2$: use Newton's method to find a point where 2 curves in the plane intersect. [4 points]

   (a) A generic conic section (a circle, ellipse, parabola, or hyperbola) can be expressed in the form $f(\mathbf{x}) = ax_1^2 + bx_1x_2 + cx_2^2 + px_1 + qx_2 + r = 0$. Implement a Python class `Conic` to represent such a conic section, which has (i) a constructor that takes the parameters (a, b, c, p, q, r) as input, (ii) a method `evaluate(x)` that returns $f(\mathbf{x})$, and (iii) a method `jacobian(x)` that returns $\mathbf{g}(\mathbf{x}) = \begin{bmatrix} \partial f/\partial x_1 & \partial f/\partial x_2 \end{bmatrix}$. Check your implementation by creating a `Conic` of known shape, e.g. $(x_1 - 1)^2 + (x_2 + 1)^2 - 2^2 = 0$, and visualizing it via the provided function `fcontour`, e.g. `fcontour(conic.evaluate, [−5, 5], [−5, 5]); matplotlib.pyplot.show()`.

   (b) Find a point of intersection of two conics f_1 and f_2 is equivalent to solving the system of equations $\begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \end{bmatrix} = \mathbf{0}$. Implement a function `newtonStep(conic1, conic2, x)` that performs one step of Newton's method for this problem. (You may use Numpy/Scipy's built-in linear solver.) Iterating `x = newtonStep(conic1, conic2, x)` should converge to a point where `conic1.evaluate(x) = conic2.evaluate(x) = 0`.

   (c) To better visualize what Newton's method is doing, add to the `Conic` class a method `linearize(x)` which returns a function $\tilde{f}$, which can be evaluated at any point $\mathbf{y}$ to give $\tilde{f}(\mathbf{y}) = f(\mathbf{x}) + \mathbf{g}(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x})$, the linearization of f about $\mathbf{x}$. (If you're not sure how to return a function, you may instead implement a method `linearize(x, y)` and then use $\tilde{f}$ = `lambda y : conic.linearize(x, y)`.) Try picking a point $\mathbf{x}$ that lies on the curve $f = 0$, and visualizing $\tilde{f} = 0$ using `fcontour`.

   Choose two interesting conics and an initial guess $\mathbf{x}^{(0)}$ not on their intersection. Make a short sequence of plots for $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ that show (i) the curves $f_1 = 0$ and $f_2 = 0$, (ii) the current Newton iterate $\mathbf{x}^{(k)}$, (iii) the linearized curves $\tilde{f}_1 = 0$ and $\tilde{f}_2 = 0$, and (iv) the next Newton iterate $\mathbf{x}^{(k+1)}$. Note that `fcontour` accepts all the formatting options of `matplotlib.pyplot.contour`; try to choose colours and line styles that make it easy to tell what is going on. Include these plots in your submission. What is the relationship of the next Newton iterate to the two linearized functions?