Mathematics 405
Programming assignment 10
Due: Friday, November 19

The shifted QR algorithm

In class, we saw a version of the QR algorithm that started with a symmetric, tridiagonal matrix A and successively found $A_k = Q_k A_{k-1} Q_k^T$ so that the off-diagonal elements of A_k converge to 0. The eigenvalues of A then appear on the diagonal.

In this assignment, we're going to speed up the convergence of the sequence A_k. Let's assume we have the tridiagonal matrix

$$A = \begin{bmatrix} a_1 & b_2 & 0 & 0 & \dots & 0 & 0 \\ b_2 & a_2 & b_3 & 0 & \dots & 0 & 0 \\ 0 & b_3 & a_3 & b_4 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_{n-1} & b_n \\ 0 & 0 & 0 & 0 & \dots & b_n & a_n \end{bmatrix}$$

So $a_k = A_{k,k}$ and $b_k = A_{k-1,k}$. If $b_n = 0$, then the matrix has the form

$$A = \begin{bmatrix} A' & 0 \\ 0 & a_n \end{bmatrix}$$

which shows us that a_n is an eigenvalue and that we can focus on finding the eigenvalues of the $(n-1) \times (n-1)$ matrix A'. So our goal will be to perform QR steps to make $b_n = 0$.

We will use $\lambda_1, \lambda_2, \dots, \lambda_n$ to denote the eigenvalues of A. With some work, we could see that the rate at which $b_n \to 0$ in the QR algorithm is proportional to $\left|\frac{\lambda_n}{\lambda_{n-1}}\right|$. To speed up convergence, we will form an estimate of λ_n, which we call σ, and then consider $A - \sigma I$, whose eigenvalues are $\lambda_j - \sigma$. In this matrix, b_n converges to 0 at a rate proportional to $\left|\frac{\lambda_n - \sigma}{\lambda_{n-1} - \sigma}\right|$. If $\sigma \approx \lambda_n$, then $\lambda_n - \sigma$ is close to zero, which means the convergence will be rapid. To recover the eigenvalues of A, we add σ to the eigenvalues of $A - \sigma I$.

Here's how we form σ, our estimate to λ_n. An easy way to estimate λ_n is just using the entry $a_n = A_{n,n}$. But a better estimate would be to consider the 2×2 matrix in the lower right corner:

$$\begin{bmatrix} a_{n-1} & b_n \\ b_n & a_n \end{bmatrix}$$

and find its eigenvalues, which are

$$\frac{(a_{n-1} + a_n) \pm \sqrt{(a_{n-1} - a_n)^2 + 4b_n^2}}{2}.$$

Choose σ to be the eigenvalue of the 2×2 matrix that is closest to a_n.

Here's how the algorithm works then. We input an $n \times n$ symmetric, tridiagonal matrix A and a tolerance ϵ.

1. Set a variable `shift = 0`.
2. Find σ using the recipe above.
3. Redefine $A = A - \sigma I$ and `shift = shift + sigma`.
4. Perform one step of the QR method redefining $A = QAQ^T$ using your previous code.
5. If $|b_n| < \epsilon$, then a_n is an eigenvalue of the shifted matrix. Recursively, find the eigenvalues of A', the $(n-1) \times (n-1)$ matrix in the upper left corner of A and combine them with a_n. Add `shift` to all the eigenvalues and return.
6. If $|b_n| \geq \epsilon$, go back to step 2 and repeat.

Goal: Your goal is to write a function `QR(A, tolerance)` whose input parameters are an $n \times n$ symmetric, tridiagonal matrix A and a desired tolerance for the off-diagonal elements. Your function should return a list of eigenvalues of A.

Some things to take note of:

- `A[:k, :k]` will give you the $k \times k$ matrix in the upper left corner of `A`.
- The last entry in a list `l` is `l[-1]` and the second to last is `l[-2]`. That means that `A[-1,-1]` is the entry in the lower right corner.
- How do you find the eigenvalue of a 1×1 matrix? This is the final step in the recursion.

A good example to test is the 9×9 matrix that computes a discrete approximation to the second derivative of a function (2's on the diagonal and -1's on the tridiagonal). Check your results against `np.linalg.eig(A)[0]`.

If you count the number of steps, you should find convergence is incredibly fast. For instance, with a tolerance of 10^{-4} and using the 9×9 second-derivative matrix A, about 15 total steps are required, which is amazing.

Remember when we looked at rates of convergence earlier and saw that fixed point iteration is linear and Newton's method is quadratic? This method is *cubic*, which is very rare and very wonderful. Due to its rapid convergence, this algorithm was named one of the 10 most important algorithms of the 20^{th} century.

`http://pi.math.cornell.edu/~ajt/presentations/TopTenAlgorithms.pdf`