Numerical Analysis (CS 450)

Homework Set 3

Due: March 12, 2014 · Out: February 27, 2014

Remarks:

- To avoid problems with Moodle's reordering of PDFs, simply turn in a single PDF with all your answers this time around (and in the future).
- Use a fixed random seed whenever this homework set asks for a random matrix:

  ```
  import numpy as np
  np.random.seed(12) # or some such
  ```

- Whenever a problem asks to 'print' something, please provide the output in the PDF you turn in.

  Make sure your program prints labels for each number being printed, so that the output can be understood without examining your code.

  Also note that we do not want pages upon pages of output—in general no more than a page or three per problem. Unless the problem specifically asks for matrix or vector entries, we usually do not want to see those. Use norms and similar ways of summarizing information.

- Unless a problem specifies otherwise, relative errors are preferable to absolute errors.

(continued on next page...)

Problem 1: QR iteration with Shifts (6 + 2.5 + 2.5 = 11 points)

Write a Python function to implement the following version of QR iteration with shifts for computing the eigenvalues of a general real matrix $\boldsymbol{A}$.

for i in $n, \ldots, 2$ **do**
  repeat
    $\sigma = a_{i,i}$ (use corner entry as shift)
    Compute QR factorization $\boldsymbol{QR} = \boldsymbol{A} - \sigma \boldsymbol{I}$
    $\boldsymbol{A} = \boldsymbol{RQ} + \sigma \boldsymbol{I}$
  until $\|\boldsymbol{A}[i-1, : i-1]\| <$ tol
end for

You may use `np.linalg.qr` for computing the QR factorization. Test your function for the following two matrices:

$$A_1 = \begin{bmatrix} 2 & 3 & 2 \\ 10 & 3 & 4 \\ 3 & 6 & 1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 6 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

and use `tol` $= 10^{-16}$. Compare the computed eigenvalues with ones computed using `numpy.linalg.eigvals`.

Note: You should incorporate the following code snippet in your program to print your computed and NumPy computed eigenvalues.

```
def qr_iteration(A, tol):
    # Your implementation goes here

A_1 = # Matrix 1
eigenvalues_1 = qr_iteration(A_1.copy(), tol)
print "Computed eigenvalues: ", eigenvalues_1
print "Actual eigenvalues: ", np.linalg.eigvals(A_1)
```


(continued on next page...)

Problem 2: Lanczos Iteration and Convergence of Ritz Values (31 points)

(a) (8 points) Prove that, in the case of $\boldsymbol{A}$ symmetric and real-valued, Arnoldi iteration (Algorithm 4.9 in the book) reduces to Lanczos iteration (Algorithm 4.10 in the book).

Specifically, show that the Hessenberg matrix $\boldsymbol{H}$ generated by Arnoldi iteration becomes symmetric and tridiagonal, with entries

$$\boldsymbol{H} = \boldsymbol{Q}^T A \boldsymbol{Q} = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

Specifically, show that:

(1) $\boldsymbol{H}$ is symmetric.

(2) $\boldsymbol{H}$ is zero above the second super-diagonal.

(3) Deduce that only two iterations (not necessarily the first two) of the orthogonalization loop in Algorithm 4.9 need to be carried out.

Hint: You may use the fact that $\boldsymbol{H}$ is upper Hessenberg, as we showed in the lecture.

(b) (8 points) Write a Python function to implement Lanczos iteration. Your function should take as input a matrix `A` and return the orthogonal matrix `Q` that is a basis for the Krylov subspace and the tridiagonal matrix `H`.

To test your function, generate a random symmetric matrix as $\boldsymbol{B} + \boldsymbol{B}^T$ for a random matrix $\boldsymbol{B}$ and print the following norms derived from the output of your routine:

- $\|\boldsymbol{Q}\boldsymbol{Q}^T - \boldsymbol{I}\|_2$,
- $\|\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{Q} - \boldsymbol{H}\|_2 / \|\boldsymbol{A}\|_2$.

Both should be small.

(c) (15 points) In this part, let $n = 32$. Set up a semi-random real symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ with eigenvalues $\lambda = 1, 2, \ldots, n$. You can do this as follows:

1. Generate a random matrix $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ using `np.random.rand`.
2. Compute its QR factorization: $\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{R}$.
3. Set up a diagonal matrix $\boldsymbol{D}$ with diagonal entries $1, 2, \ldots, n$ using `np.diag`.
4. Set $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^T$.

Perform Lanczos iteration on $\boldsymbol{A}$ and obtain the Ritz values γ_i for each iteration $i = 1, \ldots, n$. You may use `np.linalg.eigvals` to compute the Ritz values as the eigenvalues of the matrix $\boldsymbol{T}_i$.

To see graphically how the Ritz values behave as iterations proceed, construct a plot with the iteration number on the vertical axis and the Ritz values at each iteration on the horizontal axis. Plot each pair (γ_i, k), $i = 1, \ldots, k$, as a discrete point on each iteration k. As iterations proceed and the number of the Ritz values grows correspondingly, you should see vertical "trails" of Ritz values converging on the true eigenvalues.

(continued on next page...)

Problem 3: Reduction to Hessenberg form (25 points)

(a) (5 points) Let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ be a matrix. Also let $\boldsymbol{H}$ be a Householder reflector designed to annihilate rows $3, \ldots, n$ of the first column of $\boldsymbol{A}$.

Show that the matrix $\boldsymbol{B} = \boldsymbol{H}\boldsymbol{A}\boldsymbol{H}^T$ has the form

$$\boldsymbol{B} = \begin{bmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ 0 & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ 0 & * & \cdots & * \end{bmatrix}, \tag{1}$$

i.e. it has zeros below the second diagonal of the first column.

Hint: You may assume that $\boldsymbol{H}\boldsymbol{A}$ has the non-zero pattern in (1).

(b) (10 points) Write a function `reduce_to_hessenberg(A)` to transform a matrix to upper Hessenberg form using similarity transforms with Householder reflectors, as used in the previous part.

It should return two matrices $\boldsymbol{Q}$ and $\boldsymbol{U}$, where $\boldsymbol{Q}$ is orthonormal and $\boldsymbol{U}$ is upper Hessenberg, such that $\boldsymbol{Q}^T\boldsymbol{U}\boldsymbol{Q} \approx \boldsymbol{A}$.

(c) (4 points) Test your program on a 10×10 random matrix $\boldsymbol{A}$, print the relative error $\|\boldsymbol{Q}^T\boldsymbol{U}\boldsymbol{Q} - \boldsymbol{A}\|_2 / \|\boldsymbol{A}\|_2$.

(d) ($4 + 2 = 6$ points) Test your program on a 10×10 *symmetric* random matrix $\boldsymbol{A}$. (Let $\boldsymbol{B}$ be a random non-symmetric matrix. Then $\boldsymbol{B} + \boldsymbol{B}^T$ is symmetric.) Print the relative error $\|\boldsymbol{Q}^T\boldsymbol{U}\boldsymbol{Q} - \boldsymbol{A}\|_2 / \|\boldsymbol{A}\|_2$.

What do you observe about the Hessenberg matrix $\boldsymbol{U}$ in this case?

(continued on next page...)

Problem 4: Newton's method in 1D (18 points)

(a) (6 points) Newton's method for solving a scalar nonlinear equation $f(x) = 0$ requires computation of the derivative of f at each iteration. Suppose that we instead replace the true derivative with a constant value d, that is, we use the iteration scheme:

$$x_{k+1} = x_k - f(x_k)/d.$$

(i) Under what conditions on the value of d will this scheme be locally convergent?

(ii) What will be the convergence rate in general?

(iii) Is there any value of d that would still yield quadratic convergence?

(b) ($6 + 2 \times 3 = 12$ points) Write a Python function to implement Newton's method in 1d. Your function should take as inputs the 1d function, its derivative, an initial guess and a tolerance for checking convergence. Use this function to find the roots of the following functions using Newton's method with the provided initial guess.

(i) $x^2 - 1 = 0$, $x_0 = 10^6$.

(ii) $(x - 1)^4 = 0$, $x_0 = 10$.

(iii) $x - \cos x = 0$, $x_0 = 1$.

What is the observed convergence rate of Newton's method for each problem? Why does each problem exhibit this convergence rate?

You can compute the rate by using the fact that the error at each iteration $e_k = x^* - x_k$ satisfies $|e_{k+1}|/|e_k|^r = C$. Thus, $|e_{k+1}|/|e_k|^r = |e_k|/|e_{k-1}|^r$ from which you should derive an equation for obtaining r for each iteration. You can use the computed solution in the last iteration as the true solution x^* and the last of these computed rates to be the rate of convergence.

(continued on next page...)

Problem 5: Newton's method for a system (15 points)

Consider the conversion formula from spherical to Cartesian coordinates:

$$\begin{aligned} x &= r \sin\theta \cos\varphi \\ y &= r \sin\theta \sin\varphi \\ z &= r \cos\theta \end{aligned}$$

(a) (5 points) Write a function `newton(f, J, x0, tol=1e-12, maxit=500)` that implements Newton's method generically, with a starting guess of `x0`.

`f` is a function that accepts a `numpy` array x of the current state and returns the function value $f(\boldsymbol{x})$ as another `numpy` array. Write `f` so that $f(\boldsymbol{x}) = \mathbf{0}$ at the sought solution.

`J` is the Jacobian of `f`, so given a `numpy` array `x`, it returns a Jacobian matrix of the appropriate dimensions.

Terminate the iteration once either the residual is smaller than `tol` or `maxit` iterations have been performed.

(b) (5 points) Write a function that, given x, y, and z, finds r, θ and φ, using the Newton implementation from part (a). Do not reimplement Newton's method, simply pass in appropriate functions `f` and `J` to `newton` from part (a).

Find a starting guess that leads to convergence in your experiments.

The routine should accept x, y, z in *one* input vector and return r, θ and φ in *one* output vector.

(c) (5 points) Test your work from (a) and (b) by drawing 10 random (Cartesian) 3-vectors $\boldsymbol{x}^*$ using `np.random.randn` and finding their spherical coordinates.

For each case, print the final relative residual $\|\boldsymbol{x} - \boldsymbol{x}^*\|_2 / \|\boldsymbol{x}^*\|_2$, where $\boldsymbol{x}$ are the Cartesian coordinates corresponding to the spherical coordinates returned by your routine.

Also compare the polar coordinates $\widetilde{\boldsymbol{w}} = (\tilde{r}, \tilde{\theta}, \tilde{\varphi})^T$ output by your routine with the true values computed using the formulas:

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} \\ \theta &= \arccos\left(\frac{z}{r}\right) \\ \varphi &= \arctan\left(\frac{y}{x}\right) \end{aligned}$$

Let $\boldsymbol{w} = (r, \theta, \varphi)^T$. For each example, print the relative error $\|\boldsymbol{w} - \widetilde{\boldsymbol{w}}\|_2 / \|\boldsymbol{w}\|_2$ in your computed values. Is this small whenever the residual is small? Why/why not?

Hints:

- Read the documentation for and use `np.arctan2` in implementing the formulas to find r, θ, and φ.
- It is instructive to watch the conditioning of the Jacobian. A singular Jacobian will cause the method to break down or compute inaccurate results. Some starting guesses are more prone to this than others.