Scientific Computing, Fall 2022
http://www.math.nyu.edu/faculty/goodman/teaching/SciComp2022/index.html

Assignment 1

1. The residual in a linear least squares problem is

$$r = Ax - b \; .$$

Assume A is $m \times n$ with $m > n$ and that A has full rank, n. Define a function $f(x) = \frac{1}{2}\|r\|^2$. The gradient if f is the column vector whose entries are the partial derivatives

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \; .$$

Find a formula for ∇f and show that $\nabla f(x) = 0$ is equivalent to the normal equation system $A^T A x = A^T b$.

2. The system of equations $Ax = b$ is *underdetermined* if there are more unknowns than equations. If A is $m \times n$, with an $n-$component vector x of unknowns and an $m-$component vector b of data, the system is underdetermined if $m < n$. The *minimum norm* solution is x that satisfies

$$\min \|x\|_2 \quad \text{with } Ax = b \; .$$

Describe an algorithm for solving the minimum norm problem that uses the QR decomposition of A^T, assuming A has full rank.

3. (I learned this algorithm Professor Anthony Jameson.) The *matrix Lyapunov equation*, for continuous time problems, is

$$AC + CA^T = R \; . \qquad (1)$$

The unknown is a symmetric positive definite $n \times n$ matrix C. The givens are a general matrix A and a symmetric positive definite matrix R. The problem is to find an algorithm to compute C.

The problem comes from a stochastic process $\frac{d}{dt}X(t) = AX(t) + \xi(t)$. Without going into details, $\xi(t)$ represents input noise that is characterized (details missing) by a positive definite and symmetric covariance matrix R. If the process goes on a long time, then X comes to a statistical steady state with covariance matrix C, which is determined by the Lyapunov equation. This is true only if the process without noise is stable, in the sense that the eigenvalues of A all have negative real parts.

Your computational algorithm should use the upper Schur factorization $A = QLQ^*$. Note that even if A is real, its eigenvalues and Schur factorization may not be. For that reason, we say Q is *unitary* and satisfies the complex version of the orthogonality relation, which is $Q^*Q = QQ^* = I$. Note that $A^T = A^*$ (because A is real) but $A^T = A^* = QL^*Q^*$. The matrix L^* is upper triangular and its diagonal entries are the complex conjugates of the corresponding diagonal entries of L.

Your algorithm should first ask `scipy` for the Schur factorization of A. Show that the Lyapunov equation is equivalent to

$$L\widetilde{C} + \widetilde{C}L^* = \widetilde{R}\ .$$

where $\widetilde{C}$ and $\widetilde{R}$ come from C and R using Q. Then show that the entries of $\widetilde{C}$ can be found one-by-one using a back substitution strategy and the fact that L is triangular. A final step would be to get C from $\widetilde{C}$. The total work of your part (not the part done by `scipy` should be on the order of n^3, like ordinary matrix multiplication. The stability hypothesis about the eigenvalues of A is needed to guarantee that the algorithm works. Why?

4. A *Tikhonov regularized* linear least squares problem is

$$\min_x \|Ax - b\|_2^2 + \lambda \|x\|_2^2\ .$$

The regularization parameter is λ. It might be called a *hyper-parameter* to distinguish λ from the problem parameters x_k, and to indicate that it's a parameter in the algorithm, not in the solution. The parameter/hyper-parameter distinction is not absolutely clear. Explain how to solve the Tikhonov regularization problem using the SVD of A. Show that the algorithm is correct. (This was done in class, but possibly too quickly and without enough detail.)

5. (This exercise is motivated by the PhD thesis of Dan Foreman-Mackey.) A function $R(x)$ is *radial* if $R(Qx) = R(x)$ whenever Q is an orthogonal matrix. A radial function is a function of the length of x, measured in the 2−norm, which means $R(x) = \phi(\|x\|_2)$. A *radial basis function* is a radial function that is smooth and goes to zero as $\|x\| \to \infty$. It is called a "basis function" because other functions may be written as, or approximated by, linear combinations of radial functions. A radial function approximation with *centers* c_j and weights w_j has the form

$$g(x) = \sum_{j=1}^{n} w_j \phi(\|x - c_j\|)\ . \qquad (2)$$

This is a representation of g as a linear combination of shifted radial basis functions. A popular radial basis function, with length scale S, is the *quadratic exponential*

$$\phi(r) = e^{-\frac{1}{2}\frac{r^2}{S^2}}\ .$$

This exercise is to use radial basis functions, in one dimension, to estimate a function $f(x)$ from noisy data. There are m observation points x_k and data values $y_k = f(x_k) + \text{noise}_k$. We seek a function $g(x)$ of the form (2) to fit the data. Take the centers c_j to be uniformly spaced in $[0, 1]$. The fitting residuals are $r_k = g(x_k) - y_k$. Write a code that estimates the weights w_j by minimizing the loss function (Tikhonov regularized linear least squares)

$$C = \sum_{k=1}^{m} r_k^2 + \lambda \sum_{j=1}^{n} w_j^2 \; .$$

The code should *assemble* the matrix A and use the function from `numpy.linalg` to find its SVD, then use the formula from Exercise 4 to find the optimal weights w_j. Formulation and notation are part of this Exercise. The fitting parameters are called w_j here and x_j in the general theory. The data values are y_k here and b_k in the theory. The matrix entries A_{kj} depend on the function ϕ and the observation points and centers.

Start with the posted code `fakeData.py`. This generates and plots *fake data* ("data" values created by a computer random number generator rather than being measured in nature, used to test algorithms on problems with known answers). Add the SVD and fitting and plot the best fit function $g(x)$ along with the data. Also output the condition number of A. The plot should give the values of λ, S, and n.

Experiment with your code to see how well you can recover the underlying function $f(x)$, depending on the number of data points, the observation noise, and the steepness parameter L (in the `fakeData.py` code). Try to verify the following points:

- The condition number is too high to do fitting without regularization if the distance between observation points x_k is too small relative to the length scale S.
- If S is too small, the function $g(x)$ has too much fine scale oscillation to approximate f well.
- When A is ill conditioned, a small amount of regularization makes the weights w_j much smaller while making the fit quality only a little worse.
- The fitting function $g(x)$ may "overshoot" the true function f, particularly on either side of the "corners" $\frac{1}{2} \pm \frac{L}{2}$.
- RBF fitting is harder for small L.