AMATH 352: Problem Set 8

Instructor: Brian de Silva

March 3, 2017

Due: Friday March 10, 2017

Instructions:

Complete the following problems. Submit two Matlab files to Scorelator; one which generates the dat files containing your solutions to the problems in the Matlab section. You have five chances to submit your assignment. Your score for the Matlab portion of the assignment will be the highest score from your five submissions. The remaining problems (1-5) make up the written portion of this assignment. Turn in a write up of these problems digitally (via Canvas) by 5:00pm of the due date.

Least Squares:

1. Find the least squares solutions to the following linear systems by finding and solving the normal equations by hand (use Gaussian elimination). Show your work. You can check your answers by verifying that the residual is orthogonal to the columns of $\boldsymbol{A}$, i.e. that $\boldsymbol{A}^T\boldsymbol{r} = \boldsymbol{0}$.

   (a) $\boldsymbol{A} = \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$

   (b) $\boldsymbol{A} = \begin{bmatrix} 2 & 3 \\ 4 & -2 \\ 1 & 5 \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$

2. **Linear regression:** For the following data

t_i	1	2	3	5
y_i	1	0	-2	-3

   , find the line $y = \alpha + \beta t$ that best fits the data in the least squares sense. You may use Matlab to help you solve this problem, but in your writeup you must derive the linear system $\boldsymbol{Ac} = \boldsymbol{b}$ that needs to be approximately solved. You do not need to include the code used to solve this problem in the script you submit to Scorelator.

3. One can also use the least squares methodology to approximate data using basis functions which are more general than polynomials. For example, one can also use trigonometric functions. Given the data

t_i	0	0.5	1
y_i	1	0.5	0.25

   , find the trigonometric function of the form $g(t) = \alpha\cos(\pi t) + \beta\sin(\pi t)$ that best approximates the data in the least squares sense. To do this find and solve the normal equations by hand.

4. Consider the least squares problem

$$\min_{\boldsymbol{x}} \|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2.$$

Years after 1900 (t)	Population (y)
00	75.995
10	91.972
20	105.711
30	123.203
40	131.669
50	150.697
60	179.323
70	203.212
80	226.505
90	249.633
100	281.422
110	308.745

Figure 1: Total population of the USA measured in millions

(a) Show that if $\hat{\boldsymbol{x}}$ minimizes $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2$, then so does $\hat{\boldsymbol{x}} + \boldsymbol{w}$ for any $\boldsymbol{w} \in N(\boldsymbol{A})$. Here we have dropped the assumption from lecture that $N(\boldsymbol{A}) = \{\boldsymbol{0}\}$.

(b) Show that if $\boldsymbol{u}$ and $\boldsymbol{v}$ are both least-squares solutions to $\boldsymbol{Ax} = \boldsymbol{b}$, then $\boldsymbol{u} - \boldsymbol{v} \in N(\boldsymbol{A})$.

(c) Suppose now that $\boldsymbol{A}$ satisfies $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$. In this case, you can write down a closed form solution for $\hat{\boldsymbol{x}}$. Derive this solution. Note that $\boldsymbol{A}$ is not necessarily orthogonal since it may not be square, e.g. $\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$ satisfies $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$, but it is not orthogonal.

5. Suppose $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $m \geq n$, $\text{rank}(\boldsymbol{A}) = n$, and we have access to the reduced singular value decomposition (SVD) of $\boldsymbol{A}$:

$$\boldsymbol{A} = \hat{\boldsymbol{U}}\hat{\boldsymbol{\Sigma}}\boldsymbol{V}^T.$$

Here $\hat{\boldsymbol{U}} \in \mathbb{R}^{m\times n}$ has orthonormal columns (so $\hat{\boldsymbol{U}}^T\hat{\boldsymbol{U}} = \boldsymbol{I} \in \mathbb{R}^{n\times n}$, but $\hat{\boldsymbol{U}}$ is not orthogonal unless it is square), $\hat{\boldsymbol{\Sigma}} \in \mathbb{R}^{n\times n}$ is a diagonal matrix with positive entries on the diagonal, and $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ is an orthogonal matrix.

Derive a method of solving the least squares problem $\min_x \|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2$ using the SVD of $\boldsymbol{A}$. You should end up with a linear equation to solve which involves at most one instance of $\hat{\boldsymbol{U}}$, $\hat{\boldsymbol{\Sigma}}$, and $\boldsymbol{V}$. Hint: plug the SVD of $\boldsymbol{A}$ into the normal equations.

Matlab:

Create and submit to Scorelator a single Matlab script (.m file) which performs the following tasks. Your code should generate the dat files `A1.dat`, `A2.dat`,..., and `A9.dat`.

6. In this problem we construct a few different models of population growth to predict the population in 2020 ($t = 120$). Figure 1 gives the population of the United States in millions of people every ten years starting in 1900. You should use the reduced QR factorization to solve any least squares problems that arise in this exercise.

(a) Find the degree 1 polynomial ($y(t) \approx \alpha + \beta t$) which best approximates the population growth in the least squares sense. Evaluate this polynomial at $t = 120$ to obtain an approximation to the population in 2020. Store this approximation in `A1.dat`.

(b) Repeat the above for a degree 2 polynomial ($y(t) \approx \alpha + \beta t + \gamma t^2$). Save your estimated 2020 population in `A2.dat`.

(c) We can also use the least squares technique to form an exponential model of population growth, $g(t) = ce^{at}$, where c and a are parameters to be estimated. We wish to find the values of c and a so that

$$y(t) \approx g(t) = ce^{at},$$

however the dependence of y on t is nonlinear because of the e^{at} term, preventing us from formulating this as a least squares problem. To get around this, we apply the natural log to both sides of the above to obtain

$$\log(y(t)) \approx \log(ce^{at}) = \log(c) + \log(e^{at}) = \log(c) + at.$$

If we define $s(t) = \log(y(t))$ and $b = \log(c)$, then our new problem is to find a and b such that $b + at$ best approximates $s(t)$ in the least squares sense.

Solve this least squares problem using the following steps

- Compute $s_i = \log(y_i)$.
- Find the degree 1 polynomial $b + at$ that best fits the data (t_i, s_i) in the least squares sense.
- Set $c = e^b$.

Evaluate the exponential model, $g(t)$ at $t = 120$ to obtain an approximation to the population in 2020. Store this result in `A3.dat`.

Which model do you think best approximates the population growth? In just three short years you can come back and see if you were right!

7. In this problem you will compare two methods for solving least squares problems. In Matlab define a vector `x` containing 35 equally spaced points between 0 and 5. Consider the problem of finding a polynomial of degree 23 which best approximates the function $\sin(5x)$ at the points in `x`. To quickly set up the system that must be approximately solved to find this polynomial, you can use the following commands in Matlab:

```
A = fliplr(vander(x));
A = A(:,1:24);
```

The first constructs a square matrix corresponding to the system obtained by evaluating a general degree 34 polynomial at the 35 points in `x`. The second removes the columns corresponding to the terms in the polynomial of degree $25, 26, \ldots, 34$. To form the right-hand-side vector, you may use

```
y = sin(5*x).';
```

Solve the corresponding least squares problem using the reduced QR factorization. Compute the 2-norm of the residual for this solution and store it in `A4.dat`. Next solve the same least squares problem using the normal equations (and using the backslash command). Compute the 2-norm of the residual and store it in `A5.dat`.

Remark: The problem of fitting a polynomial to a set of data points becomes increasingly ill conditioned as the degree of the polynomial is increased (if one uses equally spaced points). It is especially important to avoid using the normal equations in this case, because, as you saw in a previous assignment, if $\boldsymbol{A}$ has condition number $\kappa(\boldsymbol{A})$, then $\kappa(\boldsymbol{A}^T\boldsymbol{A}) = \kappa(\boldsymbol{A})^2$. You should observe that the norm of the residual is much lower when this problem is solved with the QR factorization than when the normal equations are used.

8. **Ridge Regression:** We saw in class that using a high-degree polynomial to fit some given data can produce a very wiggly approximation. One way to deal with this problem is to use a lower degree polynomial. Another is to use a technique called ridge regression. The main idea is that "wiggly" polynomials can result from large coefficients in the approximating polynomial. These can be avoided by penalizing the least-squares solution for choosing large coefficients. Rather than solving the minimization problem

$$\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$$

which would give the least squares solution, we can instead solve the modified minimization problem

$$\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_2^2$$

where $\lambda \geq 0$ is some parameter we set to control how heavily we want to penalize large coefficients. It can be shown (by taking a gradient of this function and setting it equal to 0) that the solution to this problem is the solution to the following linear system

$$(\boldsymbol{A}^T\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}.$$

In Matlab define a **column** vector `t` consisting of 50 equally spaced points between 0 and 4 (inclusive). Next define the column vector `y` according to the following formula

$$y_i = f(t_i) + \epsilon_i$$

where $f(t) = t^3 - 6t^2 + 11t - 4$ and ϵ_i is a noise term computed with the Matlab command `randn`. So `y` contains points on some cubic polynomial plus some noise. Once `t` is formed, run the following commands to create `y`:

```
rng(4,'twister');
y = t.^3 - 6 * t.^2 + 11 * t - 4 + randn(50,1);
```

Suppose we wish to find the degree 14 polynomial that best fits this data in the least-squares sense.

(a) Set up and solve the least-squares problem (using the QR method). Note that you can (and should) use the code given in the instructions for problem 6 to help you form the system. Save the coefficients of the least-squares solution as a 21×1 column vector in `A6.dat`. If you store your solution coefficients in a column vector `c` then you can visualize your approximation by running the following commands

```
figure()
hold on          % Make it so that both plots appear on same figure
plot(t,y,'ro')   % Plot the original data points as red circles
t_fine = linspace(0,4,1000);          % Define a finer grid
y_apx = polyval(flipud(c),t_fine);    % Evaluate approximation on fine grid
plot(t_fine,y_apx)                    % Plot approximation
```

(b) Next set up and solve the ridge regression problem for $\lambda = 0.1, 1, 10$, storing the solution coefficients as column vectors in `A7.dat, A8.dat`, and `A9.dat`, respectively. To solve the linear system that arises you should explicitly form $\boldsymbol{A}^T\boldsymbol{A} + \lambda\boldsymbol{I}$ and $\boldsymbol{A}^T\boldsymbol{y}$ and then use backslash (though there are smarter ways about going about solving the system). You can plot this approximation using similar commands to those given above. If you already ran the lines above and you save the solution coefficients for this part in `c_ridge`, you can plot the ridge regression approximation with

```
y_apx_ridge = polyval(flipud(c_ridge),t_fine);
plot(t_fine,y_apx_ridge)
```

Remark: We can actually replace the $\lambda\|x\|_2^2$ term in ridge regression with a more general norm: $\lambda\|x\|_W^2 = \lambda\|\boldsymbol{W}\boldsymbol{x}\|_2^2$, where $\boldsymbol{W}$ is some invertible matrix (recall problem 1 from homework 6). The resulting linear system we need to solve to obtain the solution to the minimization problem

$$\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_W^2$$

is simply

$$(\boldsymbol{A}^T\boldsymbol{A} + \lambda\boldsymbol{W})\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}.$$

One possible choice of $\boldsymbol{W}$ is a diagonal matrix (with all diagonal entries nonzero). The resulting norm weights each of the entries of $\boldsymbol{x}$ differently:

$$\|\boldsymbol{x}\|_w^2 = \|\boldsymbol{W}\boldsymbol{x}\|_2^2 = w_{11}^2 x_1^2 + w_{22}^2 x_2^2 + \ldots w_{nn}^2 x_n^2.$$

This can be useful if we want to discourage particular entries of $\boldsymbol{x}$ from becoming large, for example.

Don't forget to comment out or delete your plot commands before submitting to Scorelator.

9. **Bonus problem/example:** You do not need to turn in anything for this problem–I just think this is a cool example and want those of you who are interested to see it. Consider the problem of interpolating the function

$$f(x) = \frac{1}{1 + (3x)^2}$$

for $-1 \leq x \leq 1$ with a polynomial of degree $n-1$. The points where we force the interpolating polynomial to match this function play an important role in determining how well the interpolant matches the behavior of the function away from the interpolation points. For example, it wouldn't make much sense to place all the n interpolation points right by $x = -1$ because we are interested in approximating the function on all of [-1,1]. One reasonable idea is to space the points evenly throughout the interval, e.g. using `linspace(-1,1,n)`. This corresponds to choosing each point according to

$$x_i = -1 + \frac{2(i-1)}{n-1}$$

for $i = 1, 2, \ldots n$. Here is some Matlab code which computes the degree $n-1$ polynomial interpolant of f at uniformly spaced points on [-1,1] and plots the result:

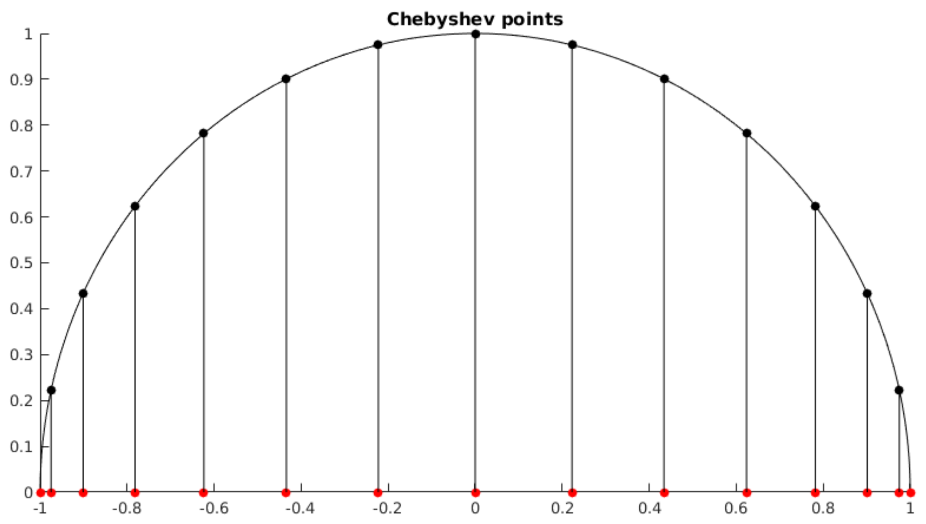
```
n = 15;                            % Number of interpolating points

x = linspace(-1,1,n);              % Set up the interpolating points
f = @(x) 1 ./ (1 + (3*x).^2);      % Define the function f as above
xfine = linspace(-1,1,1000);       % Set up a fine grid on which to plot interpolant

c = polyfit(x,f(x),n-1);           % Solve the interpolation problem

figure()                           % Plot the results
hold on
plot(x,f(x),'ro')
plot(xfine,polyval(c,xfine))
title('Uniformly spaced points')
hold off
```

You can (and should) play around with the value of n to see the effect of using higher degree polynomial interpolants. Can you get rid of the large wiggles near $x = -1$ and $x = 1$ by using higher degree polynomials?

It seems like using higher degree polynomials actually makes the fit *worse* near the boundaries. Here's another idea: what if we try placing more interpolation points near the boundaries with fewer in the middle? One such way of doing so is using the following

$$x_i = \cos(\pi(i-1)/(n-1)).$$

You can think of this as taking n equally spaced points around the top half of the unit circle, then just using their x-coordinates (the red points in the image below).



These are called Chebyshev points. Notice how this has the intended effect of bunching points up near the boundary. Try writing some Matlab code to find the interpolating polynomial using these points instead of equi-spaced ones. I have included code that performs this task on the next page.

```
xcheb = cos(pi*(0:(n-1)) / (n-1));          % Define Chebyshev points
ccheb = polyfit(xcheb,f(xcheb),n-1);        % Solve interpolation problem

figure()                                    % Plot results
hold on
plot(xcheb,f(xcheb),'ro')
plot(xfine,polyval(ccheb,xfine))
title('Chebyshev points')
hold off
```

For more information about this and for a really neat open source Matlab software pacakge, check out chebfun at `http://www.chebfun.org/`.