+3 for presentation

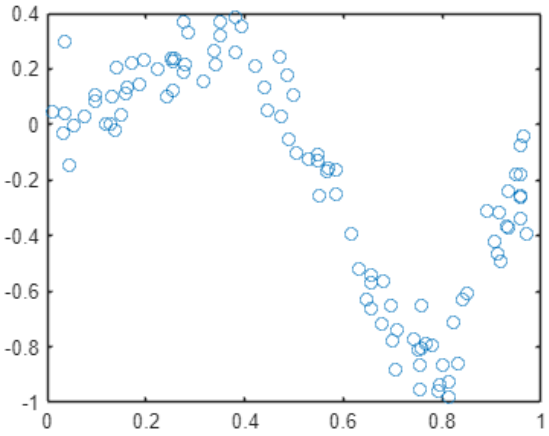
CS375 HW10

Ryan Scherbarth

November 2024

1. **Least Squares fitting (cubic!)**

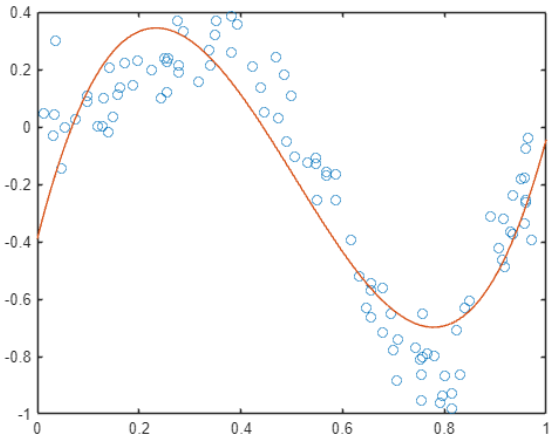
   (a) **Use the provided matlab function $[x, y] = generate\_ls\_data(N)$ with $N = 100$ to generate 100 data points. Plot the data using $plot(x, y, 'o')$ (you can add more commands to make the formatting nice, but use the $'o'$ syntax so that MATLAB does not connect the data with lines)**



   (b) **Next, we want to find the cubic polynomial given by $p_3(x) = c_0 + c_1x + c_2x^2 + c_3x^3$, that best fits the data. To do this, we will construct the Vandermonde matrix, A, given by $A = [\vec{x}^3|\vec{x}^2|\vec{x}|\vec{x}^0]$, using the data from part (a). Now, finding the cubic polynomial is equivalent to "solving" the problem**

$$A \begin{bmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \vec{y}$$

   Construct the Vandermonde matrix and compute the polynomial coefficients c_0, c_1, c_2, c_3 with a least squares approximation by solving the normal equations. It might be helpful to reuse your code from HW8 to generate the Vandermonde matrix.
   Define a fine grid of points using $xfine = linspace(0, 1, 1000;)$. Evaluate the polynomial at these points and plot the results, along with teh data from part (a). (you can use MATLAB's $polyval$ command if you ordered your c's correctly.)

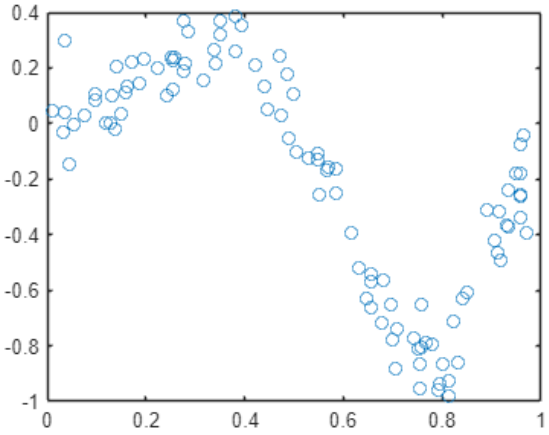


We generate the best fit cubic $p_3(x) = c_0 + c_1x + c_2x^2 + c_3x^3$. We compute each of the coefficients at each iteration; $c = (A' * A) < backslash > (A' * y)$. Then plug in to solve and graph the line with the best fit.

```
function cubic_fit(N)
    rng('default');
    x = rand(N,1);
    y = x .* sin(2 * pi * x) + 0.1 * randn(N,1);
    A = [ones(N, 1), x, x.^2, x.^3];

    c = (A' * A) \ (A' * y);

    xfine = linspace(0, 1, 1000);
    yfine = polyval(flip(c'), xfine);

    plot(x, y, 'o'); hold on;
    plot(xfine, yfine, '-');
    hold off;
end
```

2. **Least Squares fitting (cubic again!)**
we're going to redo the last problem, but now we'll use the SVD instead of the normal equations.

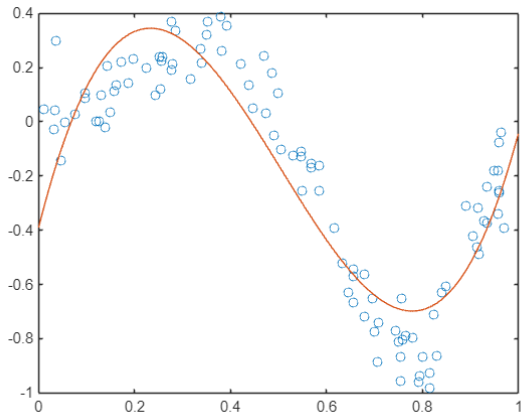
   (a) **Use the provided matlab function** $[x, y] = generate\_ls\_data(N)$ **with** $N = 100$ **to generate 100 data points. Plot the data using** $plot(x, y, 'o')$**.**
   Here, we will get the same random generated data as before;



   (b) **Next, we want to find the cubic polynomial given by** $p_3(x) = c_0 + c_1x + c_2x^2 + c_3x^3$**, that best fits the data. To do this, we will construct the Vandermonde matrix,** A**, given by** $A = [\vec{x}^3|\vec{x}^2|\vec{x}|\vec{x}^0]$**, using the data from part** (a)**. Now, finding the cubic polynomial is equivalent to "solving" the problem**

$$A \begin{bmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \vec{y}$$

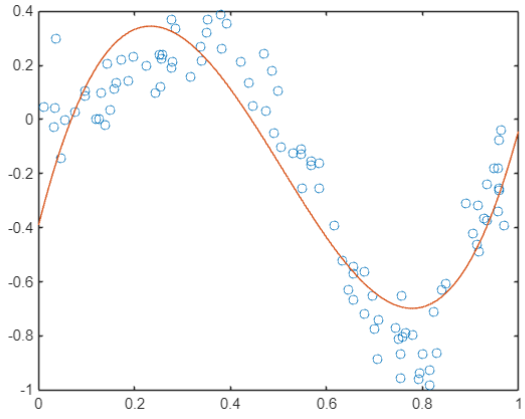
   Construct the Vandermonde matrix and compute the polynomial coefficients c_0, c_1, c_2, c_3 **with a least squares approximation using SVD fo A (you can use MATLAB's built-in** svd **command). Do you get the same answer as in the previous homework? Define a fine grid of points using** $xfine = linspace(9, 1, 1000);$**. Evaluate the polynomial at these points and plot the results, along with the data from part** (a)**.**



In the adjustment to the SVD version, we start by generating the Vandermonde matrix A, then performing SVD on A to find the matrices U, S, and V. Then we use these three to solve for c.

```
function svd_least_squares_fit(N)
    rng('default');
    x = rand(N,1);
    y = x.*sin(2*pi*x) + 0.1*randn(N,1);
    [U, S, V] = svd(A, 'econ');

    c = V * (S \ (U' * y));

    xfine = linspace(0, 1, 1000);
    yfine = polyval(c', xfine);

    plot(x, y, 'o'); hold on;
    plot(xfine, yfine, '-');
    hold off;
end
```

(c) **Repeat the two above but with $N = 500$.**



3. **QR Factorization**
Consider the matrix

$$A = \begin{bmatrix} -4 & -4 \\ -2 & 7 \\ 4 & -5 \end{bmatrix}$$

(a) **Use the Gram-Schmidt procedure (as I did in lecture) on the columns of A to produce the QR factorization of A (technically what I showed in class is the "skinny" QR factorization). Do this "by hand".**

$$A = \begin{bmatrix} -4 & -4 \\ -2 & 7 \\ 4 & -5 \end{bmatrix}, a_1 = \begin{bmatrix} -4 \\ -2 \\ 4 \end{bmatrix}, a_2 = \begin{bmatrix} -4 \\ 7 \\ -5 \end{bmatrix}$$

Start by solving q_1;

$$q_1 = \begin{bmatrix} -1 + \frac{1}{3} \\ -0 + \frac{1}{-3} \\ 1 + \frac{1}{-3} \end{bmatrix}$$

Project a_2 onto q_1;

$$\text{proj}_{q_1} a_2 = (q_1 \cdot a_2) q_1$$
$$\text{proj}_{q_1} a_2 = \begin{bmatrix} -2 + \frac{1}{-2+\frac{1}{-4}} \\ -1 + \frac{1}{-5+\frac{1}{2}} \\ 2 + \frac{1}{2+\frac{1}{4}} \end{bmatrix}$$

Solve u_2;

$$u_2 = a_2 - \text{proj}_{q_1} a_2$$
$$u_2 = \begin{bmatrix} -2 + \frac{1}{2+\frac{1}{4}} \\ 8 + \frac{1}{5+\frac{1}{-2}} \\ 3 + \frac{1}{-2+\frac{1}{-4}} \end{bmatrix}$$

...so what are the matrices Q and R'

solve R;

$$R = \begin{bmatrix} 6 & 4 + \frac{1}{-3} \\ 0 & 9 + \frac{1}{-4+\frac{1}{158}} \end{bmatrix}$$

-12

Reassemble A from Q and R:

$$A = QR = \begin{bmatrix} -4 & -4 \\ -2 & 7 \\ 4 & -5 \end{bmatrix}$$

(b) **Use Matlabs built-in qr command to compute the QR factorization of A. How does this differ from your answer in part a? (Hint: the command $rats$ can be useful when comparing Matlabs decimal representation of a matrix to the one you computed by hand.)**

```
A = [-4, -4; -2, 7; 4, 5];
[Q_builtin, R_builtin] = qr(A);

disp(rats(Q_builtin));
disp(rats(R_builtin));

A_reassembled_builtin = Q_builtin * R_builtin;
disp(rats(A_reassembled_builtin));
```

Comparing with the qr function I get Q and R values that match the values I computed in the previous part to some extent, but some positions are differing by small amounts or positive / negative. This could potentially be related to how matlab is handling the fractions as far as positivity and accuracy is concerned.

Both options do still properly reassemble to the original matrix A when multiplied together though.

Matlab values:

$$Q = \begin{bmatrix} -2/3 & 493/2773 & 422/583 \\ -1/3 & -343/365 & -8/105 \\ 2/3 & -59/202 & 635/926 \end{bmatrix}$$

$$A = \begin{bmatrix} 6 & 11/3 \\ 0 & -5521/631 \\ 0 & 0 \end{bmatrix}$$

$$QA = \begin{bmatrix} -4 & -4 \\ -2 & 7 \\ 4 & -5 \end{bmatrix}$$

This is not correct. -4

(c) **With the right hand side $\overrightarrow{b} = [3, 9, 0]^T$, use your "skinny" factorization from part a to find the least squares solution**

We can use $\overrightarrow{b} = \begin{bmatrix} 3 & 9 & 0 \end{bmatrix}^T$ To solve;

$$\begin{aligned} A &= AR \\ A\overrightarrow{x} &= \overrightarrow{b} \\ QR\overrightarrow{x} &= \overrightarrow{b} \\ R\overrightarrow{x} &= Q^T\overrightarrow{b} \end{aligned}$$

We can solve using matlab;

```
b = [3; 9; 0];

b_proj = Q' * b;

x_ls = R \ b_proj;
```

And we get a final value of;

$$x = \begin{bmatrix} -1.3868 \\ 0.9057 \end{bmatrix}$$

incorrect answer, process looks fine. EC

(d) **With the same right hand side, use the full factorization that Matlab gave you in part b to find the least squares solution. Is this the same as your answer to part c? Calculate the 2-norm of the residual without actually calculating the residual**

We can use matlab to check the least squares solution as well;

```
b = [3; 9; 0];

[Q, R] = qr(A, 0);

b_proj = Q' * b;
x_ls = R \ b_proj;

residual_norm = norm(b_proj(length(x_ls) + 1:end));
```

The result shows that we do get the same solution when using the least squares method;

$$x = \begin{bmatrix} -1.3868 \\ 0.9057 \end{bmatrix}$$

We then get the 2 norm of the residual is 0.

Your MATLAB answers and your residual are inc