Second Year Engineering Mathematics Laboratory
Hilary Term 2000

Ian Reid, 15/12//1999

Exercise 3: Regression, ill-conditioning and function-norms

The purpose of this exercise is to reinforce your understanding of polynomial regression and of its associated ill-conditioning problems. Regression is a valuable tool for interpreting experimental data. It can also be used to identify patterns automatically, as the examples towards the end, on recognition of spoken words, show.

1 Preparation

Reassure yourself that the polynomial regression problem can be formulated as follows:

$$\mathbf{A}\alpha = \mathbf{y}$$

where α is the vector of unknown coefficients, $\mathbf{y}$ is a vector of known y coordinates, and the i,jth entry of the matrix **A** is given by $(x_i)^j$.

A above is known as the "Vandermonde" matrix. Compute the Vandermonde matrix for cubic regression on the points (0.1,0.2), (0.2,0.4), (0.3,0.5), (0.4,0.4), (0.5,0.2), (0.6,0.1), (0.7,0.2), (0.8, 0.4).

Let polynomials $p(x)$ and $q(x)$ have coefficients α_i and β_i ($i = 0 \ldots n$) respectively. Show that the polynomial $p(x) - q(x)$ has coefficients $\alpha_i - \beta_i$.

The $n \times n$ Hilbert Matrix is defined to be the matrix whose i,jth entry is $1/(i+j-1)$. Prove that this matrix is equivalent to the matrix whose i,jth entry is the inner product of the monomials x^{i-1} and x^{j-1}; i.e.

$$< x^{i-1}, x^{j-1} >= \int_0^1 x^{i-1}x^{j-1}dx = 1/(i+j-1)$$

The L_2 norm of a function g is denoted $||g||$ and defined by $||g||^2 = \int_0^1 g(x)^2\,dx$. Show that for a polynomial $p(x) = \sum_{i=0}^n \alpha_i x^i$:

$$||p||^2 = \sum_{i,j}^{n} \alpha_i\alpha_j < x^i, x^j >$$

and hence, using the result above, that the L_2 norm of a polynomial is given by

$$||p|| = (\alpha^\top \mathbf{H}\alpha)^{1/2}$$

where **H** is the appropriate sized Hilbert matrix. (**Hint:** *note that* $(\sum_i \alpha_i x^i)(\sum_j \alpha_j x^j) = \sum_i \sum_j \alpha_i\alpha_j x^i x^j$).

2 Lab Work

A. Regression

Begin by copying across the files `testlin.m`, `getpoints.m`, `getsound.m`, `linear.m`, `valpoly.m`, `plotpoly.m`, `play.m`, `envelope.m` and `pn.m` from the directory `/packages/demo/comp/lab3/` into a directory called `lab3` in your own area. Now, in **matlab**, type `testlin` and the graphics window will pop up, with an invitation to plot points using the mouse, and fit a linear function as in figure 1. Try this out for several different sets of points, to make sure the linear fits look as you would expect them to. Satisfy yourself that you understand how the functions `getpoints`, `linear` and `plotpoly` work.

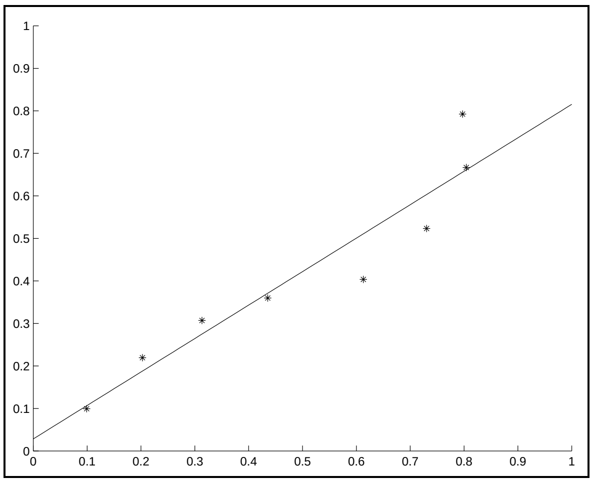

Figure 1: *Linear regression to a series of points selected using the mouse.*

Now your task is to write a function `testpoly` for polynomial regression which you should be prepared to demonstrate when your exercise is assessed. It should work just like `testlin`, except that it fits polynomials of any desired degree `M`. Its definition should begin

```
function testpoly(M)
```

and the line

```
  alpha = linear(xa, ya);
```

should be replaced by

```
  alpha = regress(xa, ya, M);
```

You need to write the function `regress` starting from the function `linear`, but with a suitably generalised form of the Vandermonde matrix. The following `matlab` stuff may be helpful:

- the operator `.^` raises each element of a vector **x** to a given power, as in $\mathbf{x}\,.\hat{}\,\mathbf{n}$
- an entire column of a matrix can be specified using the ":" special character, as in $\mathbf{A}(:, \mathbf{i})$ for the ith column of matrix **A** (similarly for rows).
- a loop over an index variable can be specified using a **for** statement, e.g.

```
for i = 0:n
    ...
end
```

[*Incidental note:* **matlab** *has built-in versions of* `regress` *and* `valpoly`, *called* `polyfit` *and* `polyval`. *Be careful if you ever use them — they assume that the vector* `alpha` *stores parameters in the opposite order, from the nth polynomial coefficient down to the 0th.*]

B. Ill-conditioning

Ill-conditioning can occur under a variety of circumstances, as you will discover.

Add the line

```
cond(A' * A)
```

to your `regress` function so that it prints out the condition number $k(A^\top A)$. Note that by omitting the "`;`" from the end of any line, `matlab` will automatically print the answer when the calcuation is complete.

Now make a data-set `xa, ya` of 11 (x, y) points, evenly spaced along the x-axis, choosing the y-values as you wish. You can do this either using the mouse as before, or from the keyboard, e.g.

```
>> xa = [0:0.01:1]'; ya = xa .^ 2;
```

1. Apply **regress** to the data-set you just made, using each of the three polynomial orders $M = 2, 3, 4$. In each case, record $k(A^\top A)$ and compare with $k(H_M)$, the condition number of the corresponding Hilbert matrix (i.e. `cond(hilb(M+1))` in **matlab**.

   Record what you notice about the values of the various condition numbers.

2. Now repeat the above with the same `xa` but a different `ya`, either selected with the mouse as above, or e.g.

   ```
   >> ya = xa .^ 3;
   ```

   Note down how $k(A^\top A)$ is affected by the change of y-values, and explain.

3. Load the arrays `xa1, xa2, ya1, ya2` from the file `arrays.mat`:

   ```
   >> load /packages/demo/comp/lab3/arrays.mat
   ```

   Apply **regress** with $M = 3$ in each of the 4 possible sets (`xa1,ya1`), (`xa1,ya2`), (`xa2,ya1`), (`xa2,ya2`).

   Note down your comments on the variation, from set to set, of

   (a) the condition numbers
   (b) the regression parameter-vector α
   (c) the shapes of the fitted curves.

C. Some speech signals

Various speech signals are provided (in `/packages/demo/comp/lab3`) which are to be used as data for the remaining parts of this lab exercise. The aim is to *recognise* utterances, and to do this some sort of measure of the *difference* between two utterances is needed.

There are three utterances each of the words "on", "off", "fast" and "slow". Type e.g.

```
>> play('/packages/demo/comp/lab3/slow1.dat');
```

to hear the utterances. (Note: if your computer doesn't have speakers, you won't hear anything. Try to hide your obvious disappointment and carry on with the exercise.)

Each utterance is a digitised electrical signal, captured from a microphone, at a sample rate of approximately 12 kHz, and stored as a sequence of 16-bit integers. You can read a signal into a **matlab** array using the function `getsound` provided. Read in the four utterances `slow1.dat`, `slow2.dat`, `fast1.dat`, `fast2.dat` into arrays:

```
>> slow1 = getsound('/packages/demo/comp/lab3/slow1.dat');
>> slow2 = getsound('/packages/demo/comp/lab3/slow2.dat');
>> fast1 = getsound('/packages/demo/comp/lab3/fast1.dat');
>> fast2 = getsound('/packages/demo/comp/lab3/fast2.dat');
```

Now use `plot` to display these 4 signals, and note how noisy they are. (The noise occurs because of the random way in which vocal chords vibrate, and because of turbulence of the airflow in throat.)

How could you use the computer to compare utterances? One idea is simply to subtract one signal array from another using a function such as the one below:

```
function diff = soundsub(s1, s2)
    % can only subtract arrays of equal length so
    % truncate both to the minimum length
    len = min(length(s1), length(s2));
    diff = s1(1:len) - s2(1:len);
```

Type this function into your computer and now plot the differences between: (slow1, slow2) and (slow1, fast1). Make sure your two plots have the same scales. You should see that array differencing is not a useful way to compare utterances.

Note down some reasons for this, based on your plots.

Direct differencing of speech signals didn't work. Now, instead, you apply polynomial regression to the processing of speech signals. The idea is to capture the general shape or *envelope* of the signal, while ignoring the noisy, fine structure. Real speech processing systems use somewhat more general methods, based on linear filtering, which are capable of dealing with larger databases of words. The approach used here is good enough for small vocabularies of a few words.

Apply the function **envelope** provided to each of the 4 utterances above, and note how the general shape of each signal is captured (use **plotpoly**), e.g.:

```
>> pslow1 = envelope(slow1);
>> plotpoly(pslow1);
```

Note that **envelope** normalises the signal to have unit length, and also rescales the envelope to have maximum value of one. Compare some speech signals with their envelopes, to check that the envelope does indeed capture gross shape. The **envelope** function uses polynomials of order 6 — was that appropriate do you think, or would more, or less have been better?

Write a function

```
function alpha = envdiff(e1, e2)
```

which takes two envelopes (computed using **envelope**) and returns the difference between them as a polynomial, represented by the parameter vector `alpha` – recall from the preparation that this will be a rather simple function.

Apply **envdiff** to the envelopes of the utterance pairs (slow1, slow2) and (slow1, fast1) from the previous exercise. Plot each difference polynomial (using **plotpoly** with appropriately modified axes).

Record the results and comment on why this new procedure – envelope differencing – might make a better basis for recognition of utterances.

D. Comparing fitted signals using the L_2 norm

The final part of this exercise illustrates an important method in pattern recognition: the use of *norms* to compare patterns. In the previous part, the difference between envelopes was plotted as a (polynomial) function. Now the aim is to express the overall size of that difference as a single number. Then, when the size of the difference between two envelopes is small, it will be more likely that they represent two utterances of the *same* word.

A function **pn** is provided that computes the L_2 norm of a polynomial by the method you proved in the preparation. Check the code to verify it implements your proof.

Complete the table below, showing the norms of the differences between the envelopes of various utterances `fast1.dat`, `off1.dat` etc., taken to be "laboratory standard", with "test" utterances `fast2.dat`, `off2.dat`, etc.:

Record what you notice about the values of norm-differences along the diagonal of your table. Does it look promising for successful recognition of spoken words?