MATHS 2104 Numerical Methods
Assignment 4:
Least-squares polynomial fitting & solving the heat equation using Jacobi iteration
Due: Tuesday 10 October 2017 at 1:00pm

CHECKLIST

- ☐: Have you shown all of your working for every mathematical question?
- ☐: Have you included all Matlab code and plots to support your answers in programming questions?
- ☐: Have you made sure that all plots have labelled axes, and legend where appropriate?
- ☐: Have you submitted all code required to Cody Coursework?
- ☐: If before the deadline, have you submitted your assignment via the online submission on Canvas?
- ☐: Is your submission a single pdf file - correctly orientated, easy to read? If not, penalties apply.
- ☐: Penalty for submitting more than one document: 10% of final mark, per extra document. Note: you may resubmit and your final version is marked, but final document must be a single file.
- ☐: If after the deadline, but within 24 hours, have you contacted us via the enquiry page on Canvas (if applying for an extension) and submitted your assignment online via Canvas?
- ☐: Penalties for late submission: within 24 hours, 40% of final mark. After 24 hours, assignment cannot be submitted and you get zero.
- ☐: Assignments emailed instead of submitted via Canvas will not be marked and will receive zero.
- ☐: Have you checked that the assignment submitted is the correct one, as we cannot accept other submissions after the due date?

Preamble

The aims of this assignment are to:

- Develop code for polynomial regression (least-squares fitting);
- Explore a method for model selection in data science;
- Understand how Jacobi iteration can be used to solve PDEs (the heat equation).

Contents

1 Polynomial regression and model selection

Consider the problem of finding the best least-squares fit for a polynomial of degree n,

$$y(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

to a set of data values (x_j, y_j), for $j = 1, \ldots, m$.

The difference between the model and the data is

$$e_j = y(x_j) - y_j, \quad j = 1, \ldots, m, \tag{1}$$

which can be written in matrix form, $\boldsymbol{e} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$, and the coefficients $\boldsymbol{x} = [a_0, \ldots, a_n]^T$ can be found by minimising the error $||\boldsymbol{e}||$.

1.1. Assuming $m > n$, write down the system of equations (1) in matrix form, and therefore identify $\boldsymbol{A}$, $\boldsymbol{x}$, and $\boldsymbol{b}$. How many solutions to the system of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ are expected to exist in general?

1.2. Write a MATLAB function called `fitpoly.m` that fits a polynomial of degree n to any data set. The function must have the interface

```
function [y,coefs,c] = fitpoly(x, xj, yj, n)
```

where `x` is a column vector containing the values of x at which you want to evaluate the fitted function $y(x)$, `coefs` is a vector containing the best-fitting coefficients $a_0, \ldots, a_n$, `c` is the condition number of the matrix $\boldsymbol{A}$, `xj` is a column vector containing the values of x_j, `yj` is a column vector containing the corresponding values of y_j, `n` is the degree of the polynomial to be fitted, and `y` is a column vector containing the corresponding values of $y(x)$. *Your function should be fully vectorised, with no* `for` *loops.*

Realistically (as with all these assignments), you should be developing this function in tandem with the test script `testfitpoly` in Q1.4.

Now, suppose you're presented with a dataset, and want to fit an n-th degree polynomial to it. How do you choose n? You want to minimise the sum of squared errors $||\boldsymbol{e}||^2 = \sum_{j=1}^{m} e_j^2$, which can be done by increasing n, but this can lead to overfitting, and besides, ideally you'd like to keep your model as simple as possible (ie, lower n).

1.3. Choosing $n = m$ leads to $||\boldsymbol{e}|| = 0$. Why? (A one-sentence answer will suffice.)

Statisticians and data scientists achieve this balance between model complexity and goodness-of-fit by minimising the Akiake information criterion (AIC). For least-squares model fitting, we can take this quantity to be:

$$AIC = 2n + m \ln \left(\sum_{j=1}^{m} e_j^2 \right) \tag{2}$$

As you can see, as n increases the first term goes up while the second goes down. There's therefore a minimum for some value of n. When fitting to a dataset we want to find and use this value of n.

The ASCII file `facebook.dat` which contains (normalised) weekly search volume counts for Facebook from 2007—2015, collected via Google Trends[1]. The first column of the file gives times x_j since 1 January 2007 in years, and the second column gives the corresponding normalised search volumes y_j.

1.4. Write a test script called `testfitpoly.m` which loads the Facebook data, and then uses a `for` loop to fit increasingly complex polynomial models to the data (starting from $n = 1$ until it finds one which minimises AIC, and then `break`s out of the loop. (See Practical 4.)

This is called *stepwise regression* in the statistical literature.

1.5. Plot the data and your n-th degree polynomial fit over the same time period. You may leave your x-axis label as "number of years since 1 January 2007".

1.6. Do you think your model could be reasonably used to predict Facebook search volume today, in September 2017? Why or why not?

Try it!

1.7. Fit a degree-20 polynomial to the dataset. With reference to the output arguments of your `fitpoly` function, explain why Matlab throws a warning message here.

Congratulations! That was a crash course in data science. If from this you remember terms like "model selection", "overfitting", "Akiake information criterion", "out-of-sample prediction", and can use them sensibly in a sentence, you're well on your way to the so-called "sexiest job of the 21st century"!

2 Solution of the heat equation using Jacobi iteration

The equilibrium temperature u at a position x along a heated rod is governed by the one-dimensional steady heat equation

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} - \alpha u = -f(x), \tag{3}$$

[1] See Cannarella & Spechler, *Epidemiological modeling of online social network dynamics* (2014) for more information, as well as some entertaining mathematical modelling which treats Facebook and other social networks as infectious diseases!

where α is a positive parameter that is related to the rate at which heat is lost laterally and $f(x)$ is a heat source. If the temperature at either end is fixed, then the boundary conditions are

$$u(0) = u_0 \quad \text{and} \quad u(L) = u_L. \tag{4}$$

Consider N equispaced points along the rod such that $x_i = (i-1)h$, where $h = L/(N-1)$ and $i = 1, \ldots, N$. Let the temperature at the i-th point be denoted by $u_i = u(x_i)$. As we have seen, an approximate formula for the second derivative is

$$u''(x_i) = u_i'' \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}.$$

Substituting this into the heat equation (3) yields

$$u_{i-1} - (2 + \alpha h^2)u_i + u_{i+1} = -h^2 f_i, \quad i = 2, \ldots, N-1, \tag{5}$$

where $f_i = f(x_i)$, together with the boundary conditions

$$u_1 = u_0 \quad \text{and} \quad u_N = u_L.$$

This can be written as the linear system

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}, \tag{6}$$

that is,

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2-\alpha h^2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2-\alpha h^2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & -2-\alpha h^2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2-\alpha h^2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \\ u_N \end{bmatrix}}_{\boldsymbol{u}} = \underbrace{\begin{bmatrix} u_0 \\ -h^2 f_2 \\ -h^2 f_3 \\ \vdots \\ -h^2 f_{N-2} \\ -h^2 f_{N-1} \\ u_L \end{bmatrix}}_{\boldsymbol{b}}$$

The aim of this exercise is to solve (6) using Jacobi iteration.

2.1. Referring to the relevant theorem from your notes, explain why Jacobi iteration is guaranteed to converge to the solution of (6) when $\alpha > 0$.

Jacobi iteration is given by

$$\boldsymbol{u}^{(k)} = \boldsymbol{u}^{(k-1)} - \boldsymbol{D}^{-1}\boldsymbol{r}^{(k-1)},$$

where $\boldsymbol{D}$ is a diagonal matrix whose diagonal elements are the same as those of $\boldsymbol{A}$ and $\boldsymbol{r}^{(k)}$ is the residual

$$\boldsymbol{r}^{(k)} = \boldsymbol{A}\boldsymbol{u}^{(k)} - \boldsymbol{b}. \tag{7}$$

In this case, it is not necessary to construct the matrix $\boldsymbol{A}$. Instead the components of the residual can be calculated directly from (5) as

$$r_i = u_{i-1} - (2 + \alpha h^2)u_i + u_{i+1} + h^2 f_i, \quad i = 2, \ldots, N-1,$$

together with

$$r_1 = u_1 - u_0 \quad \text{and} \quad r_N = u_N - u_L.$$

2.2. Use the above information to write down a formula for the elements of the Jacobi iterate $\boldsymbol{u}^{(k)}$ in the form

$$u_i^{(k)} = u_i^{(k-1)} - \,?\,, \quad i = 2, \ldots, N-1,$$

Note that if $u_1 = u_0$ and $u_N = u_L$ then $r_1 = r_N = 0$, in which case there is no need to update u_1 and u_N.

2.3. Write a MATLAB function called `solveheat.m` that uses Jacobi iteration to solve (6).

Your function must use the interface

```
function [u, x, rnorm] = solveheat(alpha, func, L, u0, uL, N, tol, maxits)
```

where `alpha` is the value α, `func` is the user-supplied function which evaluates $f(x)$ at each grid point, `L` is the length of the rod, `u0` and `uL` are the boundary conditions u_0 and u_L, respectively, `N` is the number of grid points N, `tol` is the tolerance, and `maxits` is the maximum allowable number of iterations. (See Practical 4.)

You should assume that `a`, `b`, `N`, `tol` and `maxits` are all scalar arguments.

Your function will have to create a suitable initial guess that satisfies the boundary conditions and iterate until

```
norm(r) < tol
```

where `r` is the residual $\boldsymbol{r}$ from (7), or the number of iterations exceeds `maxits`. You must use vectorised code to calculate the residual `r`. You should not construct the matrix $\boldsymbol{A}$ or the inverse $\boldsymbol{D}^{-1}$.

On output, the arguments `u` and `x` should be column vectors that contain the approximate values of the solution $u_i = u(x_i)$ at each of the corresponding grid-points x_i for $i = 1, \ldots, n$, and `rnorm = norm(r)`, where `r` is the residual of the output solution `u`.

2.4. Consider the boundary value problem

$$\begin{aligned} \frac{\mathrm{d}^2 u}{\mathrm{d}x^2} - u &= \sin x, \quad 0 < x < 1, \\ u(0) = 0, &\quad u(1) = 1. \end{aligned}$$

Verify that the exact solution is

$$u(x) = \frac{1 + \frac{1}{2}\sin 1}{\sinh 1} \sinh x - \frac{1}{2} \sin x.$$

Don't forget to check the boundary conditions!

2.5. Write a script called `testsolveheat.m` that uses the function `solveheat.m` to obtain an approximate solution of the problem in (2.4) for $N = 20$. Your script should create two plots:

(a) One showing the exact solution and the approximate solution as functions of x.

(b) Another showing the absolute value of the difference between the exact solution and the approximate solution as a function of x.

As always, make sure your plots are properly labelled. Where applicable, use a legend and appropriate line and symbol specifications to distinguish between different curves.

Use anonymous functions to define $f(x)$ within your script.

You will need to experiment a little with the parameters `tol` and `maxits` to get satisfactory results. If `tol` is too small, many iterations will be required to converge, increasing computational expense without increasing the *overall* accuracy of the solution. If `tol` is too large, the solution will not be sufficiently accurate.

2.6. Add code to `testsolveheat.m` that uses Jacobi iteration to solve the heat equation (6) for

$$N = 100,\ L = 1,\ \alpha = 1,\ u_0 = 0,\ u_L = 1 \quad \text{and}$$
$$f(x) = 80 \exp\left(-\frac{(x - 0.4)^2}{0.001}\right)$$

and plots the result.

This Gaussian $f(x)$ approximates heating in the centre of the bar, say by a candle. Note that while this BVP looks similar to Q2.4, an analytic solution is not available! (At least not to me.)

Include a hardcopy of your code and all plots in your report.

Jacobi iteration is usually considered too slow for practical use. However, methods to accelerate Jacobi iteration have recently been developed. See:

Yang, X. I. A. & Mittal, R. (2014). Acceleration of the Jacobi iterative method by factors exceeding 100 using scheduled relaxation. *J. Computational Physics*, **274**, 695–708.

Nineteenth century math tactic gets a makeover—and yields answers up to 200 times faster.

3 Assignment requirements

3.1. You must submit the final versions of your functions `fitpoly` and `solveheat` to Cody Coursework, as follows:

(a) Login to Cody Coursework.

(b) Click on "Assignment 4" and then on "Polynomial regression"/"Heat equation using Jacobi iteration".

(c) To submit your code, click on "Solve" and copy and paste your functions `fitpoly`/`solveheat` into the text boxes provided.

(d) Test your function by clicking on "Test". When you are happy, click "Submit" to save your function. You will receive one mark for each test that is passed.

(e) You can revise and resubmit your function as often as you like, up until the deadline. Your best solution will be used to assign your execution mark.

(f) For general help, consult the Mathworks Cody Coursework documentation.

(g) You do not have to submit your scripts `testfitpoly` or `testsolveheat` to Cody Coursework.

3.2. In addition, you must also submit a written report answering each item in Sections 1 and 2 as a **single PDF file** via Canvas. Where relevant, this will include:

(a) printouts of relevant scripts and functions (including code submitted to Cody Coursework),

(b) printouts of numeric output and/or graphs,

(c) any supporting analysis, and

(d) answers to any questions.

These answers must be professionally presented in a readable and informative manner. Graphs and tables must be properly labelled. All scripts and functions must be properly documented and and set out using the conventions established for this course. Haphazard and illegible work will not be marked.

Please see the Assignments directory on MyUni for further notes on the Assignment marking scheme.

3.3. You may discuss problems and thrash out the answers together but the work you submit must be your own and not copied from someone else.