**The ‘Keeling Curve’ and analyzing time series data in MATLAB**

The Mauna Loa Observatory in Hawaii, operated by the Scripps Institute of Oceanography, is home to one of the longest-running atmospheric CO2 monitoring stations in the world. Constructed from the measurements initiated by Charles David Keeling in 1958, the ‘Keeling Curve’ is one of the iconic instrumental records in climate science, showing the rise in atmospheric CO2 over the past five decades.

In this exercise, we are going to analyze the trends in the CO2 record monitored at Mauna Loa, and make some short term predictions based upon it. Along the way, we will also detour into practical data handling, interpolation and plotting.

**1. The raw data**

The Mauna Loa CO2 data are available for download in comma separated variable (CSV) form from the Scripps website: <http://scrippsco2.ucsd.edu/data/atmospheric_co2>

I recommend you download the monthly data file (‘monthly\_mlo.csv’) as it turns out it is much easier to parse in MATLAB!

The CSV format is a convenient format for saving tables of numbers (e.g. spreadsheets) in ASCII-readable files. Each line in the file represents a row in the table, and these are themselves separated into columns by commas, like so:

Yr, Mn, Date, Date, CO2,...

, , , , ,...

, , Excel, , [ppm],...

1958, 01, 21200, 1958.0411, -99.99,...

1958, 02, 21231, 1958.1260, -99.99,...

1958, 03, 21259, 1958.2027, 315.69,...

1958, 04, 21290, 1958.2877, 317.46,...

1958, 05, 21320, 1958.3699, 317.50,...

[The first few data rows of the ‘monthy\_mlo.csv’ file. Here, the observation dates in decimal years are in the fourth column, and the raw CO, measurements are in the fifth column.]

There is a built-in MATLAB command, csvread, that can be used to read numerical data (and only numerical data – no strings allowed!) from CSV files and load them into a matrix. This has the syntax

data\_matrix = csvread(infile,start\_row,start\_column)

where ‘infile’ is the name of your input file, and ‘start\_row’ and ‘start\_column’ are the row and column where you want the data reading to start, with ‘0’ representing the first row or column in the file.

In this specific case, you want to choose a value of ‘start\_row’ that would exclude all of the header information in the input data file, since csvread will give an error if it encounters anything non-numeric that is not a comma!

|  |
| --- |
| * Write a MATLAB script that reads the Mauna Loa monthly CO2 time series data, and plots them. I recommend using the date in decimal years (column 4) for the x values, and the raw, unadjusted CO2 data (column 5) for the y values. Label your plot and axes accordingly, and paste the figure into the document below here.   Figure:     * Comment on any issues that your data set may have that have revealed themselves through this plotting exercise.   **1) The data file has many lines of header information (57 in my version of the file) that have to be bypassed in order to read the file using csvread without errors.**  **2) The CO2 time series has a number of data gaps (‘holes’) presumably due to problems with the monitoring equipment at certain times. It appears that where no measurement was made, a ‘null’ value of –99.99 ppm was used to fill the gap. (This is documented in the header information of the file.)**  **The choice of number was deliberate – you can’t have negative partial pressures of CO2 in nature, so it must be a null value!**  **3) The first two and (possibly) the last few lines of data, depending on the date the file was downloaded, are also data gaps, representing months where data were not, and months where data have not yet been, collected.** |

**2. Interpolation**

In the case where time series data contain gaps, we can attempt to estimate the missing values by using interpolation methods. These generally rely on the principle that if the time series has a smooth functional form either side of, and across the gap – in other words, if a smooth function can be fitted to it – then we can use that property to calculate approximate values for the missing data.

MATLAB has a number of built-in functions that can be used to interpolate data. The most appropriate of these for our purposes are pchip and spline.

pchip is a piecewise cubic interpolator that respects ‘monitonicity’ of data (i.e. it maintains linear trends, where possible), but is also capable of abrupt changes in trend. It achieves this by allowing abrupt changes in the second differential of the function that it fits to the data.

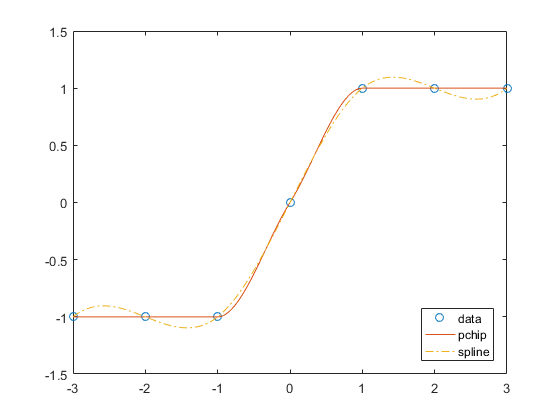
spline attempts to fit a cubic spline function to the data provided, which generally results in a smoother, and more curved result. It achieves this by trying to have any changes in the second differential of the function vary smoothly in the data fitting process.

Both functions have the same syntax in MATLAB:

yi = pchip(x,y,xi)

yi = spline(x,y,xi)

where x is a vector of locations (e.g. dates) and y a vector of the corresponding data (e.g. CO2 measurements) that you want to interpolate, xi is a vector of locations where you want to estimate interpolated data, and yi is the output vector of interpolated data value.



[Difference of performance of the pchip and spline interpolators, figure downloaded from <https://www.mathworks.com/help/matlab/ref/pchip.html>]

|  |
| --- |
| * Write a MATLAB script to identify and fill gaps in the time series by interpolation. [HINT: extract a few data points either side of each gap to use as your x and y vectors.] Paste in a plot of your ‘cleaned’ time series below, appropriately titled and labeled.   Cleaned time series plot:     * Describe briefly your approach.   **I extracted up to three data points either side of each data gap (a maximum of 6), and used the pchip interpolator to estimate the value at the location of the gap.**   * Describe the main features of the cleaned time series.   **Overall, the atmospheric CO2 concentration is increasing with time, with a gradient that also is increasing with time (i.e. the rate of increase is increasing; CO2 is accelerating). Superimposed on that pattern is an annual oscillation, with a peak in the spring and a trough in the fall.**   * Which of the pchip or spline functions is most appropriate for this particular task, do you think, and why?   **I found that the pchip function was more effective at filling the largest gap in the data, a 3 month hiatus in 1964. The spline function had a tendency to ‘undershoot’ the peak, whereas pchip did not, as shown below (red dots indicate interpolated values):** |

**3. Trend-fitting and extrapolation**

The simplest form of trend estimation is fitting a straight line to data.

Take the simple equation for a straight line:

Of course, in our data we have multiple estimates of y (CO2 concentrations) at multiple values of x (dates). Therefore, we can write multiple equations of the form

⋮ ⋮ ⋮

We can recast this system of equations in terms of a matrix equation:

**y** ≈ **A** **m**

where **y** = [y1 y2 y3 ... yn]T, **x** = [m c]T and **A** is the matrix:

|  |  |
| --- | --- |
| **A** = | matrix.png |

|  |
| --- |
| * Do the matrix multiplication **y’** = **A** **m** for the first three rows of A to demonstrate that it is indeed equivalent to the simultaneous equations shown above.   **y1 = (x1 ⨉ m) + (1 ⨉ c) = mx1 + c, etc.** |

We want to solve this matrix equation for the vector **m**. This can be achieved by the least squares method.

We premultiply both sides of the equation by the transpose of the **A** matrix:

**AT y** = **AT A m .**

And now premultiply both sides by the inverse of **AT A**:

(**AT A**)-1 **AT y** = (**AT A**)-1 **AT A m**

**m** = (**AT A**)-1 **AT y** ,

since any matrix multiplied by its inverse is the identity matrix. The implication is that if you can invert **AT A**, you can solve for **m**.

MATLAB implements this least squares solution method using the ‘backslash’ operator, \:

m = A\y;

|  |  |  |
| --- | --- | --- |
| * Write a MATLAB script implementing this least-squares line-fitting scheme for a chosen subset of the interpolated Mauna Loa CO2 data. [HINT: for the purposes of your calculations you may want to make your dates start from zero, rather than 1958, to avoid numerical error warnings.] * Hence, fit straight lines to the data for the periods 1959–1965 and 2000–2015.   straight_fits.png   * By how much did the rate of CO2 increase change between those two periods?   **My estimates: 0.621 ppm/year for 1959–1965**  **2.057 ppm/year for 2000–2015**  **The rate of CO2 accumulation has more than trebled between the two periods.**   * Predict the atmospheric CO2 concentration at Mauna Loa in 2040 if it continues to increase at the average rate from the last 15 years.   **By extrapolating the straight line fit for 2000-2015, I estimate a CO2 concentration of 450.51 ppm.**   * How could you modify the line-fitting scheme described above to solve for the best-fitting parabola (i.e. the best-fitting quadratic function)? Write the matrix equation and define the contents of its matrices and vectors.   **The general form of a quadratic function** **is** y = ax2 + bx + c. **This can be represented by the matrix equation,**  **y** = **A m** ,  **where y is the same as above,**  **m** = [a b c]T, **and**   |  |  | | --- | --- | | **A** = | matrix_quadratic.png | |

This functionality is provided within MATLAB by the function polyfit, which can fit a polynomial function of specified order to a given data set. (A 2nd order polynomial would be a quadratic function, a 3rd order polynomial a cubic function, an nth order polynomial will be a function of xn and smaller powers of x.)

The syntax of the polyfit command is:

polyparams = polyfit(x, y, order),

where x and y are the same quantities as before (locations/dates and data, respectively), order defines the order of the polynomial being fitted to the data, and polyparams is an output vector of the (order+1) coefficients for the polynomial terms (equivalent to the vector m in our example above, although by default polyfit will output this as a row vector).

The polynomial function thus obtained can be evaluated using the companion function polyval:

ye = polyval(xe,polyparams) ,

where xe is a vector of locations where you would like to evaluate the polynomial function, polyparams is the output from polyfit, as defined above, and ye are the evaluated values of the polynomial function that correspond to the values in xe.

|  |
| --- |
| * Make a new version of your MATLAB script that fits a polynomial of a chosen order to the full interpolated CO2 data set. * Hence, predict the atmospheric CO2 concentration in 2040, and plot it against the interpolated data. Comment on your result, and how it relates to your ‘linear prediction’ made earlier.   **I estimate a CO2 concentration of 462.81 ppm in 2040 from my best-fitting parabola. This is ~12 ppm larger than the estimate from a linear extrapolation.**  parabola_fit.png   * Evaluate what happens when you change the order of the polynomial function you fit to the data (e.g. try order 3, 4, 5, 6, … and plot them on the same graph). What effect does increasing the order of the polynomial have on your prediction for CO2 in 2040? What is the reason for any variability in your predictions, and which prediction do you consider to be the most robust, and why?   **There are dramatic differences between the extrapolated curves as you move to higher order polynomials. While the 3rd order polynomial follows a similar trajectory to the parabola, the 4th and 5th order polynomials show a more rapid increase in CO2 after 2016, whereas the 6th order polynomial shows a similar trajectory to the parabola for the first few years, then a sharp decrease after 2030. And of course all of these curves are fitted to the same data! Adding higher-order polynomial components to the curves when they are not required makes minimal difference to the fit to the data, but can add an unpredictable long-term trajectory to the extrapolated curves. (In climate science, the predictions of different climate models with different numbers of assumptions and different model parameters tend to diverge in a similar way, although the cause of the divergence is less straightforward to pinpoint.)**  **Since the parabola is a simple model (only three parameters) that fits the data well, it is likely to give the most robust prediction.**  polynomial_fits.png |