The Sum Function

Leigh J. Little SUNY Brockport September 14, 2018

Do the following:

- Download the 2 files from the website (`sc1.m` and `testdata.dat`).
- Configure Windows to show file name extensions by clicking on `View` at the top of any operating system window and checking the box that says 'Show Filetypes'.
- Change MATLAB into your usual working directory and copy these two files into that directory.
- Make sure your browser did not change the file extensions (Firefox sometimes does this). Change the filenames to the correct ones if necessary.
- Run the `sc1` script.

You should recognize the output. In this case, MATLAB computed and plotted the best-fit line through a set of data points. If you look at the `sc1.m` script, you will notice that some commands are familiar, some commands we have not looked at, but their purpose seems clear (`max` function). Some operations are a little less clear. The meat of the whole calculation is in the `polyfit` function, which we will come back to later in the semester.

If you have taken a statistics course, you know that there is more to the best-fit line than just what you are looking at now. Equally as important as the graphical part is assessing the quallity of the line itself. Is the line reasonably approximating the data and can we use this line to reliably make other calculations?

This is a good example of a real problem that illustrates the need to be able to augment MATLAB with our own calculations. MATLAB provides a large number of general purpose functions for calculations, but it can't predict everything we will need. The basic quantities we need to assess the quality of this best-fit line are all easily computed, but we are going to have to do these calcullations ourselves.

Let x be the vector of x-coordinates and y be the vector of y-coordinates. We would like to compute the following:

- The Pearson correlation coefficient. This is defined as $r = \dfrac{\text{cov}(x,y)}{\sigma_x \sigma_y}$ where σ_x is the standard deviation in x, σ_y is the standard deviation in y and $\text{cov}(x,y)$ is the covariance between the vectors x and y. This covariance is defined as

$$\text{cov}(x,y) = \frac{\sum_{i=1}^{n}(x_i - x_{\text{mean}})(y_i - y_{\text{mean}})}{n-1}$$

  where x_{mean} and y_{mean} are the means of x and y respectively.
- The R^2 value. This is defined as $R^2 = r^2$.

There are other things we would typically compute, but these are the most common values needed.

As it turns out, all of these quantities can be quickly computed using the MATLAB operations we have seen, provided you understand what the summation notation means. Recall that for a vector $x = x_1, x_2, \ldots, x_n$ of length n,

$$\sum_{i=1}^{n} x_i = x_1 + x_2 + \cdots + x_n.$$

This basic sum simply adds up the elements of the vector x. From this sum, we can quickly determine the mean value of x.

$$x_{\text{mean}} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

This formula says add up the elements of x and divide the sum by the number of elements.

Fortunately, we have a MATLAB command for computing sums of vectors.

```
>> x = [1 3 4 2 6]
x =
     1     3     4     2     6
>> sum(x)
ans =
    16
```

As can be seen, the `sum` function will add up the elements of a vector.

Note that if the input is a matrix, the `sum` function does something slightly different

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> sum(A)
ans =
    12    15    18
>> sum(sum(A))
ans =
    45
```

If the input is a matrix, `sum` returns a vector consisting of the sums down the columns of A. If we want to add up all of the elements of A, we need to do a sum of the sum. This last example illustrates an important concept and this is the idea of *nested function calls*. It is possible to use the output of one function directly as the input to another function, for example

```
>> sum(sqrt(x))
ans =
    8.5958
```

To begin building our computation, we need the means of x and y. This is easy since we have the `sum` function and another function, the `length` function

```
>> length(x)
ans =
     5
```

This function will give you the number of elements in a vector. Thus, we can compute the mean of x by doing

```
>> meanx = sum(x)/length(x)
ans =
    3.2000
```

The next quantity we need is the standard deviation. This is defined as

$$\sigma_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - x_{\text{mean}})^2}.$$

This formula looks bad, but it turns out that by combining the `sum` function and the basic operations, it can be done almost as easily as the calculation of `meanx`. The worst-looking part of this formula is the the sum

$$\sum_{i=1}^{n}(x_i - x_{\text{mean}})^2 = (x_1 - x_{\text{mean}})^2 + (x_2 - x_{\text{mean}})^2 + \cdots + (x_n - x_{\text{mean}})^2$$

Consider the calculation

```
>> x = [1 3 4 2 6]
x =
    1     3     4     2     6
>> x - 4
ans =
    -3    -1     0    -2     2
```

If you subtract a constant from a vector, that constant gets subtracted from each element of the vector. Moreover, we can easily square each element of this result

```
>> (x-4)^2
Error using  ^
Incorrect dimensions for raising a matrix to a power. Check that the matrix
is square and the power is a scalar. To perform elementwise matrix powers,
use '.^'.
>> (x-4).^2
ans =
    9     1     0     4     4
```

We need to remember to use the `.^` operator since we want to square each element of $x-4$ and not multiply $x-4$ by itself (which can't be done). We can then do

```
>> sum(  (x-4).^2  )
ans =
    18
```

This calculation subtracts 4 from each element of x, squares each element of the resulting vector and sums the elements of the vector. Essentially, this is the ugly part of the standard deviation formula. Putting it all together, we have

```
>> stdx = sqrt(  sum( (x-meanx).^2 )/(length(x)-1))
stdx =
    1.9235
```

The last part of our calculation that we need is the nasty looking sum in the formula for covariance

$$\mathrm{cov}(x,y) = \frac{\sum_{i=1}^{n}(x_i - x_{\mathrm{mean}})(y_i - y_{\mathrm{mean}})}{n-1}$$

However, this is almost as easy as the standard deviation. We need to take x subtract its mean value, then do the same for y. We then need to multiply these two vectors element by element, then sum the result. This gives

```
>> cov = sum(  (x-meanx).*(y-meany) )/(length(x)-1)
```

assuming that we had defined a vector y and computed its mean.

Finally, we compute the Pearson correlation coefficient using $r = \dfrac{\mathrm{cov}(x,y)}{\sigma_x \sigma_y}$ The sequence of steps in a script file would look like

```
n = length(x);
meanx = sum(x)/n;
meany = sum(y)/n;
stdx = sqrt(  sum( (x-meanx).^2 )/(n-1));
stdy = sqrt(  sum( (y-meany).^2 )/(n-1));
cov = sum(  (x-meanx).*(y-meany) )/(n-1);
r = cov/(stdx*stdy);
rsquared = r^2;
```

This represents a generic process for computing r and R^2. You would need to replace x and y with the variable names that you are actually using in your script file. For example, the script we started with initially used `xval` and `yval` instead of `x` and `y`.