FRS 135 State of the Earth: Shifts and Cycles (in Spain)

Lab 02b: Campus GPS — II

►10/08/2015

Due: 10/08/15 – upload your PDF to *Blackboard* by 1:30 pm Thursday.

Your `csv` file contains your observations and you have stripped the header and resaved it. Now you use the command `load` to load the numeric variables into your workspace.

My own solutions would be in `fjsimonsl02a.csv`, so after typing `load fjsimonsl02a.csv`, I will have the variable named `fjsimonsl02a` in my workspace. I could also use the *functional form* of `load` by typing `bla=load('fjsimonsl02a.csv')` in which case I would now have the variable `bla` in my workspace. Note that this option requires you to identify, by the straight-up quotes, the *filename* as a *string*: a set of characters instead of a *variable*. **Watch out:** If you don't have the right number of columns (17), your file was not saved correctly. Turn it into the right size as follows:`if min(size(bla))==1; bla=reshape(bla(:),17,[])'; end`

I prefer descriptive but short variable names. Of course you could have gone with the first solution and renamed the variable by typing `bla=fjsimonsl02a` but then you'd have two variables with the same information. Find that out by typing `whos`, and then you could get rid of one by typing, for example, `clear fjsimonsl02a`.

Now you have the data you can do anything you want. The commands that speak for themselves are `help`, `lookfor`, `type`, and of course `plot`, `xlabel`, `ylabel`, `title` and `hist`, which we have demonstrated in class. Let's say you wanted to make a histogram but forgot how to: `lookfor histogram` would turn up `hist`, and `help hist` would tell you how to use it. Then, `type hist` will give you read access to the source code if you want to dig in.

More subtle is how to address the entries in your matrix named `bla`. You might want to pick out the fourth column, `bla(:,4)`, or the third row, `bla(3,:)`, or simply the last column, whichever that is, `bla(:,end)`. Remember the semicolon (`;`) is a screen output "silencer".

You now might want to *logically* address all elements in the fourth through sixth columns for all rows whose last column was the number five; this would be `bla(bla(:,end)==5,4:6)`. Or the last column for all those rows whose first column contains numbers greater than twenty-four, `bla(bla(:,1)>24,end)`. Type `help relop` for logical "relational" operators.

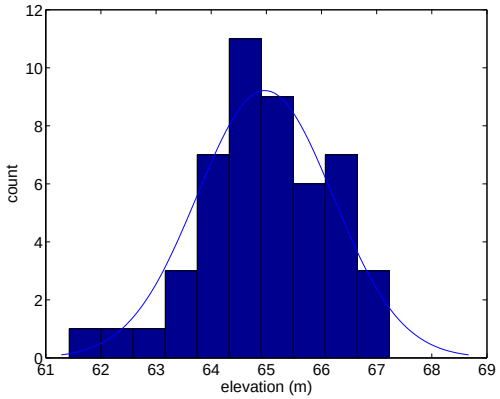
Warm-Up Exercises

Now it's your turn to do **something interesting** with your data, and with everyone's data. What are the *mean* and *variance* of the elevation on your third control point? That would be `melev=mean(bla(bla(:,2)==3,6))` and thereby you would have this mean computed and assigned to variable `melev`; similarly, you might calculate `velev=var(bla(bla(:,2)==3,6))`.

What is the **histogram** of all the weather conditions in your survey? That would be the categorical `hist(bla(:,15))`, and you would annotate the figure with `xlabel('met indices')` and `ylabel('count')`, `title('weather conditions during my survey')`. What is the data *drift* with time? How is my precision compared to that of the others? Of the entire class? Is my precision a function of the number of satellites? Is x more accurate than z?

Let's pretend you have all elevations collected in a variable `elv`, and for the sake of the argument, let me simulate some **Gaussian** random data in there, with standard deviation 1.2 and mean 65: `elv=randn(49,1)*1.2+65`. From this sample of 49 data, you expect the `mean(elv)` and `sqrt(var(elv))` or `std(elv)` to be close to 65 and 1.2, respectively. Check for yourself, and note it will get better as the sample size increases. By executing `hist(elv)` you'll get the standard histogram, in *counts*, and with the default number of *bins*. Is this close to being *normally* distributed? We could plot a **normal curve** on top of it by freezing the image using `hold on` and then adding a Gaussian with the same mean and variance.

Let's see. First we would define some kind of an x-axis for this problem, for example, the linearly spaced vector `x=linspace(mean(elv)-3*std(elv),mean(elv)+3*std(elv),100);` and then we'd calculate the values of the normal distribution at these points, taking care to have the *area under this curve be the same as the area under the histogram*. See if you can make sense of this: `g=normpdf(x,mean(elv),std(elv))*length(elv)*range(elv)/10;` which we would plot right on top of the histogram by now writing `plot(x,g)`, the 10 for the default number of 10 bins in the histogram. Here is the bare-bones plot that results:



This visual procedure of checking for normality is very sensitive to the number of observations and bins. Better use a *quantile-quantile* plot, which is built right into Matlab as `qqplot(elv)`. If the crosses fall on a line, your data are likely normally distributed.

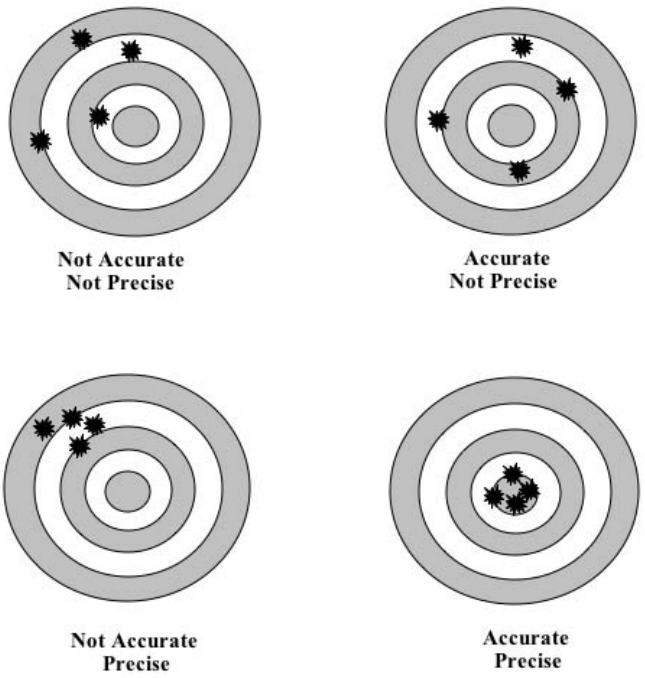
Assignment

Your job now is to prepare a short lab report where you (1) make and present an analysis of your own GPS data and (2) compare your results with the data collected by the entire class. Use the LaTeX templates that we provided, and include at least **three** figures.

Since you are using the templates, it is quickest to include PDF (`*.pdf`) figures in your document. Once you have a figure on the screen, type `print -dpdf figurename` and you'll see the file `figurename.pdf` appear in your working directory. Remember to crop the white-space off it by using `trim` and `clip` in `\includegraphics`!

This needs to be done by all — One figure

The goal of the required exercise is to get an idea of the **precision** *and* the **accuracy** of your GPS. First, distinguish both terms. *Accuracy* is how close to the *truth* your measurements are *on average*. *Precision* is how distributed the measurements are about *their* average. Take a look at the figure below. You're shooting four times at a target. The average mark of the accurate shooter is right on target. The marks of the precise shooter are tightly clustered around their average. But of course the average of the shots may not be close to the truth.



Given how confusingly used these terms are in the (popular) literature, I prefer to use the proper terms: **bias** (how close the measurements are to the truth, on average) and **variance** (how tightly they cluster about their average). Other terms are **systematic** (bias) and **random error** (variance). The ideal measurement technique yields low bias, and low variance, and the combination of both (bias squared plus variance) is the **mean-squared error**. Keeping *that* to a minimum is what you're really trying.

Re-mapping Control Points gives you an idea of the **variance** (the expected value of the squared distance to their expected value, where the expectation itself is defined as that number with respect to which the expected squared distance of the points, itself, is a minimum!) of your measurements, which is due to all sorts of factors: the weather being one of them.

Re-measuring the Control Line gives you an idea of the **bias**: you (think you) *know* the truth if you have walked a straight line such as a curb. By plotting your measurements against one another, determining the best-fit line, and comparing that to what you (think you) know *is* a straight line: it's not a watertight argument but you should be very sensitive to biases in the measurement, at least over the course of your experiments.

Inspect the distribution and calculating the variance of your Control Points was covered under the "Warm-Up Exercises", but let's see how the Control-Line procedure works.

Let me make some **synthetic** data like the ones that you'll be collecting. Pretend the following (x, y) positions are the truth: ten linearly spaced x positions between 0 and 100, and ten corresponding y positions with an offset of 3 and a slope of 0.5, in other words: a true **straight line**:

```
x=linspace(0,100,10);
y=3+0.5*x;
p1=plot(x,y,'o-');
title('true intercept 3 ; true  slope 0.5')
```

Notice that I plot the (x, y) pairs as symbols connected by lines, and notice that I request an output to the `plot` statetemnt, this variable `p1` is a *handle* that I can manipulate later.

Now pretend you made measurements at these same locations, but there is **additive noise** in both components. From your experiments at the Control Points, you might have been able to conclude that the noise is normally distributed with variances $\sigma_x^2 = 3.8$ and $\sigma_y^2 = 4.2$. A small bias in both components could take the form of a nonzero mean in those noise components, $\mu_x = 3.3$ and $\mu_y = 3.8$. Generate a total of 12 experiments with those characteristics:

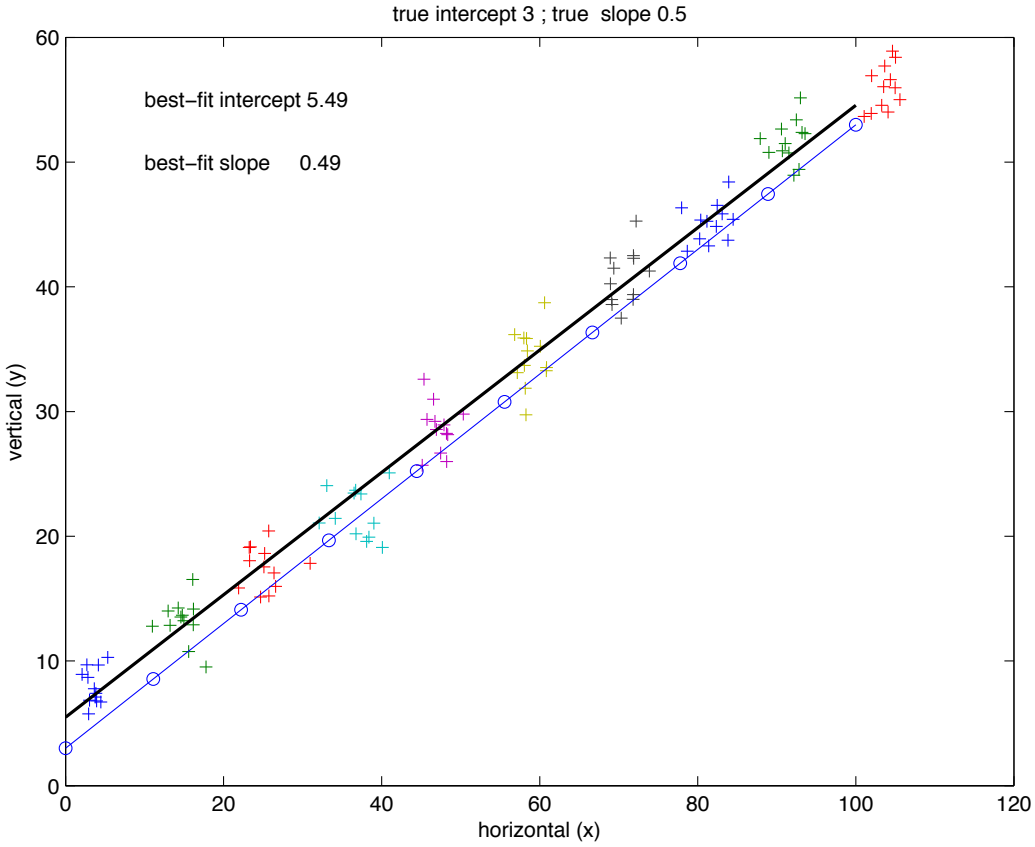
```
nx=random('norm',3.3,sqrt(3.8),[12 10]);
ny=random('norm',3.8,sqrt(4.2),[12 10]);
```

When trying to add these two noise matrices to the "true" locations, you have to grow the arrays `x` and `y` to have the same dimensionas as `nx` and `ny` so that you *can* add them.

```
xm=repmat(x,12,1)+nx;
ym=repmat(y,12,1)+ny;
hold on
plot(xm,ym,'+')
xlabel('horizontal (x)')
ylabel('vertical (y)')
```

If we simply tried to find the best-fitting **least-squares** regression line through all of our points, we would look for the **straight line** that is **closest** to all of the noisy measured points,

in the sense that the **sum of the squared distances** of every point to the regression line is minimized. In its simplest form we would measure the distance along the y direction only. What this means is that we would conside the horizontal variable x to be the truth, and the vertical variable y to be subject to noise. *As you know (since you made up the data!) this oversimplification ignores the noise in the horizontal variable.*



Now, bear with me: the **regression line** is given by a bit of a formula that derives from linear algebra. Later, we will redo it using a preprogrammed Matlab function. First, we collect *all* of the measurements `xm` and `ym` into one big, concatenated *column* vector:

```
XM=xm(:);
YM=ym(:);
```

Then, we would make the **sensitivity matrix** that expresses the linear relation that we suppose exists between the variables: `XM` and `YM`:

```
A=[ones(size(XM)) XM];
```

Finally, we would obtain the *two* coefficients, one *intercept* and one *slope* from the *generalized inverse* applied to the data vector. Don't worry about the details if this is opaque — yet.

```
m=inv(A'*A)*A'*YM;
```

The first entry of the solution is the best-fitting intercept, the second the best-fitting slope. We'll add those results to the plot using a *formatted string*:

```
text(10,55,sprintf('best-fit intercept %3.2f',m(1)))
text(10,50,sprintf('best-fit slope     %3.2f',m(2)))
```

And finally, we visualize the regression line by evaluating what the y values would be if we requested them at any x points of our choice. We might as well take the original `x` vector as the points where we evaluate the regression, and connect everything by a line. Of course *two* points, e.g. the endpoints, would do, since Matlab, if you ask for a line plot, will just connect them by a straight line.

```
ypred=m(1)+m(2)*x;
plot(x,ypred,'k-','linew',1.5)
```

Now you see that we've gotten the *slope* pretty right, but we've discovered some of the bias in the y direction, as our *intercept* is off. In this example we just *know* the truth, and we *know* the bias in y that we put in (it was the mean value of the noise in y, or $\mu_y = 3.8$). Not too bad; my intercept is 5.49 and it should be 3. The *apparent bias* in this example is 2.49.

At this point I am no longer pretending that I have the truth. Take the truth off the plot:

```
delete(p1)
```

All I have left is my **data points** and my **regression line**. For good measure, let's use Matlab's `refline` to make sure it gives us the same result. Clear the plot and start over: plot every point, use `refline`, and overlay our own best-fitting line. They should coincide.

```
clf
plot(XM,YM,'k+')
refline
hold on
plot(x,ypred,'bo')
```

Yup, they do. Matlab's `refline` calls `lsline` calls `polyfit` calls `qr` calls etc... you would never guess that it was as simple as the almost-one-liner that I just gave you. Moving on.

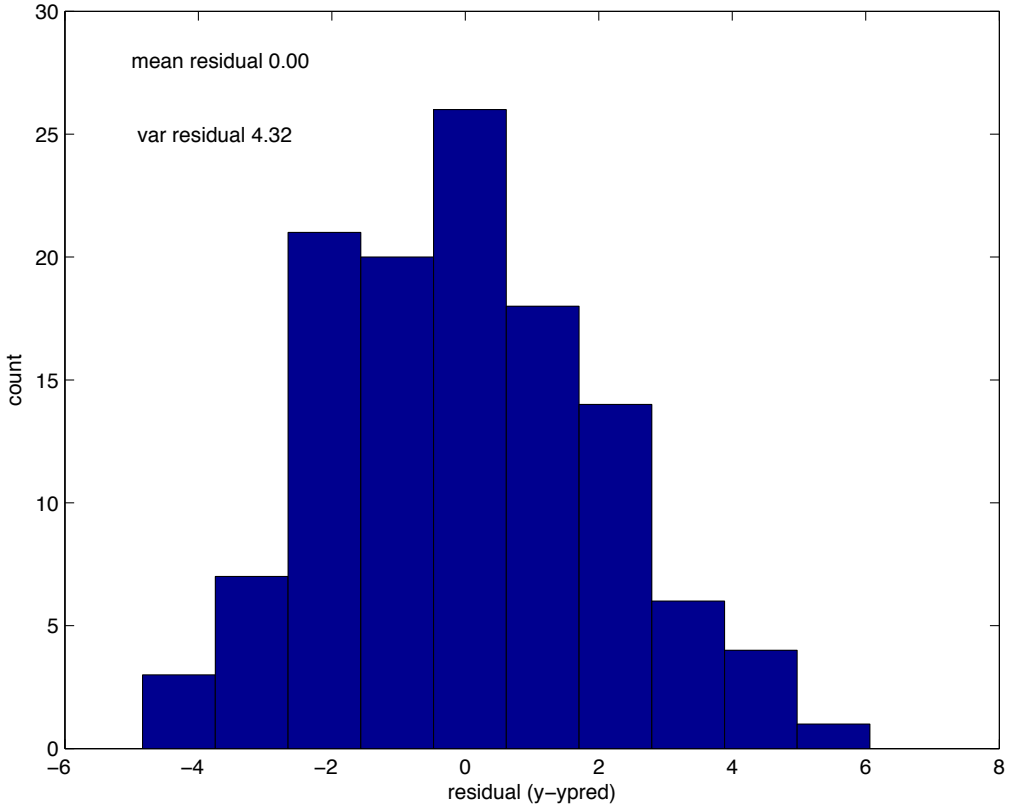
The fact that you have a *best-fit line* does not mean *at all* that my line is a *good fit*! Ponder that. It could be terrible! **How good is the fit?**

The first thing I should do is **inspect the residuals**. Those are the difference between the measured y values and the predicted y values. How should they be distributed? We need to make predicted y values for *every one* of the measured x values, not just at some small convenient set for display purposes. In other words, we *recompute* predictions and subtract the observations. And then we make an annotated histogram of those!

```
ypredm=m(1)+m(2)*xm;
resid=ypredm-ym;
```

```
clf
hist(resid(:))
xlabel('residual (y-ypred)')
ylabel('count')
text(-5,28,sprintf('mean residual %3.2f',mean(resid(:))))
text(-5,25,sprintf(' var residual %3.2f',var(resid(:))))
```

Our histogram does look pretty normal (and elsewhere in this document and in the previous lab you will find some easy ways to test for normality in a more formal manner). We have a **mean** residual that is very close zero, which is another way of telling us that we have lost all information about the possible *bias* (we no longer pretend we know the truth!), but we do know something about the *variance*. In my experiment it is 4.32, and remember that I *put in* the value $\sigma_y^2 = 4.2$. Not bad! (What we have learned about the *variance* of y, without knowing the truth, is actually pretty close to the truth.)



How good is our regression line? Well, it predicts the y values of an experiment where we consider the x values as perfectly known to within a standard deviation of $\sqrt{4.32}$, which we take as our *precision*. Compare this to the precision that you derive from your Control Points, as it will be close. And if we had access to the truth (a much better measurement, say), we could even say something about the *accuracy*, in the end. Now, a popular way to asses the quality of a least-squares regression line fit is by calculating the **correlation coefficient**.

Once again, I'll give you the relation to calculate this coefficient, and then we'll verify using Matlab's canned function `corrcoef`, which has some bells and whistles. The correlation coefficient is the normalized **covariance** between the variables. If it's close to 1 the two variables go up together, if it's close to -1 they go in opposite directions, and if it is close to 0 there is no correlation between the variables. Using Matlab's `cov` and `var` we compute the 2×2 matrix whose off-diagonal elements (the matrix is symmetric, so only one of the matters) are the correlation coefficient:

```
RM=cov(XM,YM)/sqrt(var(XM))/sqrt(var(YM)); RM(2)
```

Now we're at it, an alternative for the the *slope* of the regression line is a slight variation:

```
SM=cov(XM,YM)/var(XM); SM(2)
```

and the *intercept* is also quite simply computed from the result:

```
mean(YM)-mean(XM)*SM(2)
```

Compare this to `m(1)` and `m(2)`! It's the same thing! Powerful stuff.

The correlation coefficient is *positive* at (in my example) 0.9914, so the **linear correlation** between the variables is high. But how strong is strong, and how high is high? The **significance** depends on the number of data. Intuitively, the bar on calling a correlation **strong** should be higher and higher (closer to 1 or -1) the less data you have. With just a few observations, the chances that they correlate *by chance* are quite high. With a lot of observations, these chances go down, so even weak correlations can be significant.

Formalizing a **significance test for correlation** is hard, because you need to first **model** the chances that an observed correlation coefficient occurs *by chance* in a sample of a certain amount of data if the data in fact are *not* correlated. If you know those chances, and you have a certain observed correlation coefficient, you can then rule out that it is as high as it is completely fortuitously (when in fact the data are not correlated). Matlab's `corrcoef` has **one** such test built in, which **only** applies if the variables that you test for correlation are **normally** distributed. In our case, they were! (We made them ourselves. For your field experiments, you'll see if the assumption of normality holds).

We'll call the function with two outputs:

```
[R,P]=corrcoef(XM,YM); R(2)
```

and we get the same thing that we knew already, that the correlation coefficient is 0.9914 (in my experiment). And when we look at the second output,

```
P(2)
```

which is the calculated probability that a correlation coefficient as high as observed, or even higher, occurs by random chance under the assumption that the variables are normally distributed. In my case these chances were vanishingly small: the value of 0.9914 is deemed **significant**.

Never, ever calculate a correlation coefficient without doing and reporting the test for significance. And **never** do the test for significance without knowing the distribution of your data. (Yes, this means you will need to stay in school.) And **only** when the data are normally distributed should you use Matlab's `corrcoef` to conduct such a test: it only applies in the easy case of normality; they simply did not take the time to derive and program any other tests!

While normal distributions are pervasive in nature, *lots* of other distributions are also pervasive, and they might look *nothing* like the normal distribution. Examples are photon counts in detector systems, radioactive decay processes, the energy of ocean waves, the frequency of stock-market crashes, to name a few.

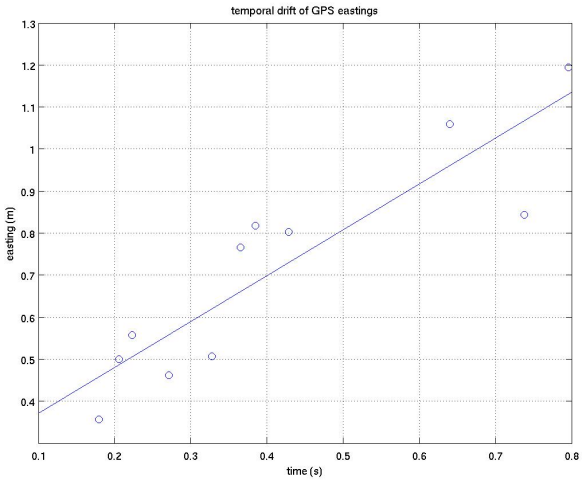
And finally here you can use your imagination — Two more figures... some ideas

Let's say you have an array variable that you call `tims` which has eleven entries, as in `tims=rand(11,1);` which contains a **uniformly distributed** random set for the purposes of this example. Let's assume you have a variable that you call `xlocs` which has the eastings of your GPS data set, as in `xlocs=tims+rand(11,1)/2`. As you can see both variables are linearly correlated: one is just the sum of the other and some other random set that has one quarter of the variance — for the purpose of this example. Type `plot(tims,xlocs,'o')` and you'll see your data appear on the screen. In this example, you'd be studying the **drift** of your eastings with respect to tims. Then `xlabel('time (s)')`, `ylabel('easting (m)')`. Let Matlab plot a **regression line** on top of this: for this you simply type `refline`. Add some grid lines by typing `grid on` and `title('temporal drift of GPS eastings')`. Print the figure to a PDF file as `print -dpdf lab2fig1` and see file `lab2fig2.pdf` appear.

Again, in your template, when you put into the editor the statements

```
\begin{figure}[h]\begin{center}
   \includegraphics[width=0.55\textwidth]{lab2fig1}
\end{center}\end{figure}
```

and then run `pdflatex` as usual, you should see a figure like this here.



Now suppose these were your data, and you have collected the data for the rest of the class in the variables (filled with random entries for this example, as in `timsrest=rand(121,1),;`) and let's assume everyone has done a better job than you, so their eastings might be something like `xlocsrest=timsrest+rand(121,1)/10+0.25`. These are also linearly correlated but with much lower variance than yours, and with a slightly different slope. Plot those data on top of yours using first `hold on` and making a different choice of symbol, e.g. `plot(timsrest,xlocsrest,'+')`. Add a legend for the three things that you've just plotted, `legend('mine','my regression','theirs')`. Here is the figure I get when I execute each of the above commands in order.

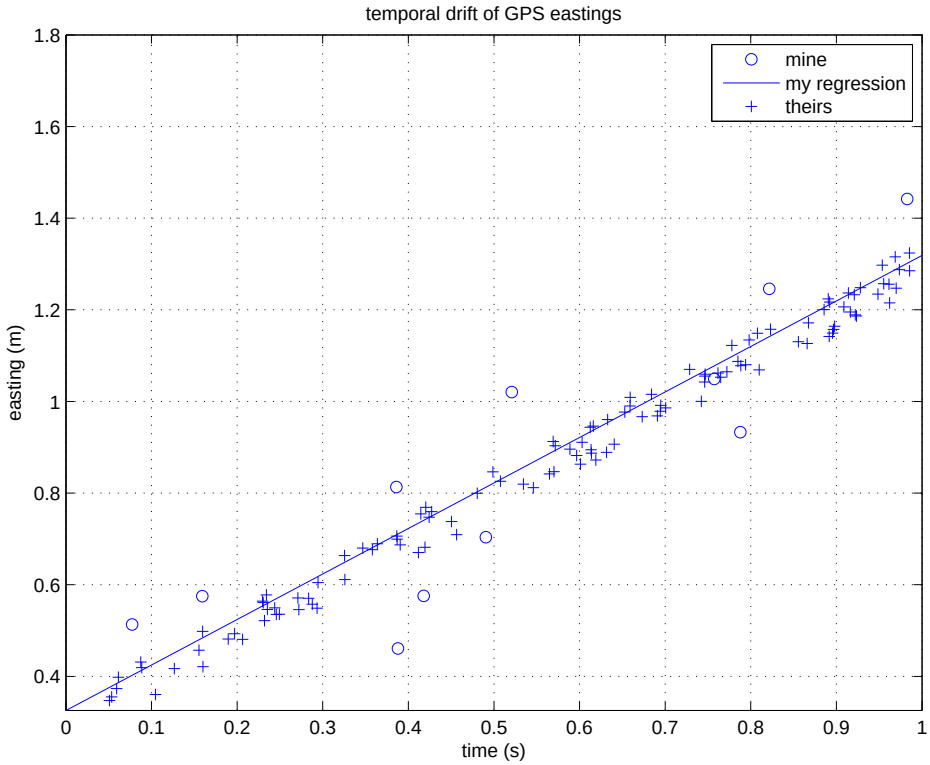


Figure 1: Temporal drift of GPS measurements: my results compared to those of the class, with a regression line for my results added. The drift is approximately 1 m s^{-1}, and this is of course totally and wholly unrealistic given that this was a made-up example using uniform random data.

Here is *verbatim* what I typed in my LaTeX document to get the above result:

```
\begin{figure}[h]
\begin{center}
   \includegraphics[width=0.75\textwidth]{lab2fig2}
\parbox{0.5\textwidth}{\caption{Temporal drift of GPS measurements: my
results compared to those of the class, with a regression line for my
results added. The drift is approximately 1~m $s^{-1}$, and this is
of course totally and wholly unrealistic given that this was a made-up
example using random data.}}
\end{center}
\end{figure}
```

I obviously only provided one caption so far, which is why LaTeX starts with number 1. In your document, you would have had a caption for every figure and you'd be up to 6 by now.

When you're done, type `hold off` to release the overlay on your graph, or when you want to start over, type `clf` to start afresh.

Here is a **final example** where you might plot a **scatterplot** of elevation versus number of sattelites, color-coded for weather. Once again, making it all up. The data are in the array `GPSdata` and the third column might contain elevation and the ninth column the weather code. I have 7 data with cloudy skies, let's say that's code 12, and 8 data with sunny skies, pretend that is code 9. Let's imagine sunny (mean 32, variance 4) beats cloudy (mean 31.5, variance 9), thus var(GPSdata(GPSdata(:,9)==9,3)) `GPSdata(:,3)=[randn(8,1)*2+32 ; randn(7,1)*3+31.5]`; will create such an array (so far only a third column). Now I add the ninth column for the weather by typing `GPSdata(:,9)=[repmat(9,8,1) ; repmat(12,7,1)];`.

Is the variance what I think it is? Type `var(GPSdata(GPSdata(:,9)==12,3))` for the cloudy elevations (I got 8.4) and `var(GPSdata(GPSdata(:,9)==9,3))` for the sunny ones (I got 3.2). Similarly, calculate the means, `mean(GPSdata(GPSdata(:,9)==12,3))` and `mean(GPSdata(GPSdata(:,9)==9,3))`. For these small numbers and the huge difference in variance but the small difference in means you'll notice the variances are much easier to distinguish than the means. But anyway, we've got our synthetic data set. No wait, we still have to make up the number of satellites, let's say that's column six. Put some random integers there for now, `GPSdata(:,6)=randi(24,8+7,1);`. Now we're done simulating.

Make it all easier to digest by reassigning to `sunnyel=GPSdata(GPSdata(:,9)==9,3)` and `cloudyel=GPSdata(GPSdata(:,9)==12,3)`, and `numsat=GPSdata(:,6)`, for example. The plot below is the result of `plot(sunnyel,numsat(GPSdata(:,9)==9),'ro')`, and then type `hold on`, `plot(cloudyel,numsat(GPSdata(:,9)==12),'b+')`. Is there a trend? Try typing `refline`, `legend('sunny','cloudy','sunny trend','cloudy trend')` and discuss. Note that `refline` gives you the *best-fitting* linear trend... but no information on whether this trend is remotely *good* or **significant**. To interpret this, you'll need to go back a few pages. Anyway, here's the figure... which you'd need to further label and annotate. Good luck!