So far we've examined error propagation in terms of linear combinations of variables.

For non-linear functions, we used a truncated Taylor series to linearize it (valid for small variations/errors).

There's another way to do all this: Resampling Methods

The basic idea of all these methods is the same: That there exists some "population" of possible data measurements. The actual data measurements are drawn from this population. Thus, if we sample the data in different ways, we can get parameter estimates that give us a measure of error!

we'll look at 4 methods:

1) The Jackknife: Sequentially drop one data point and determine how parameters change

2) The Bootstrap: Draw a sample of n data points from a periodically replicated set of original data, determine how parameters change

3) Undersampling: Divide a large data set into smaller samples, calculate parameters & see how they vary

4) Monte Carlo simulation: Add random noise (of approp. mag) to data, calc parameters & see how they change.

First we look at the Jackknife
— first suggested ~1949, term coined '58

Suppose we have N meas

we wish to calculate the statistics of $f(\bar{x})$ (this can be non-linear)

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

we may define f_j s.t.

$$f_j \equiv f\left(\frac{1}{N-1}\sum_{i\neq j} x_i\right)$$

↳ avg. of x excluding jth point!

The quantity $f(\bar{x})$ is estimated by the average of f_j!

$$f(\bar{x}) \approx \frac{1}{N}\sum_{j=1}^{N} f_j$$

↳ less bias than $\overline{f(x_i)}$ for non-linear prob.

The thing is, we can also use this to get the variance!

The formula is:

$$\sigma^2_{f(\bar{x})} = \frac{N-1}{N} \sum_{j=1}^{N} \left(f_j - \bar{f}_j\right)^2$$

This looks a little strange, but it makes sense from error propagation. Consider the application to linear regression. We have: ↳ or non-linear.

$$\underset{\sim}{x} = \underset{\sim}{f}(\underset{\sim}{b}) = \underset{\approx}{K}\,\underset{\sim}{b} \quad (\text{for linear})$$

We know that

$$\underset{\approx}{\Sigma}^2_x = \underset{\sim}{\nabla}\underset{\sim}{f}\ \underset{\approx}{\Sigma}^2_b\left(\underset{\sim}{\nabla}\underset{\sim}{f}\right)^T$$

If the b_i are all indep then this is:

$$\underset{\approx}{\Sigma}^2_x = \sum_{i=1}^{N}\left(\frac{\partial \underset{\sim}{f}}{\partial b_i}\right)\left(\frac{\partial \underset{\sim}{f}}{\partial b_i}\right)^T \sigma^2_{b_i}$$

$$\approx \sum_{i=1}^{N}\left(\frac{\underset{\sim}{f}(\underset{\sim}{\mu}_b + \sigma_{b_i}\hat{\underset{\sim}{e}}_i) - \underset{\sim}{f}(\underset{\sim}{\mu}_b)}{\sigma_{b_i}}\right)\left(\quad\right)^T \sigma^2_{b_i}$$

↳ numerical derivative

$$\approx \sum_{i=1}^{N} \left(f(\underset{\sim}{\mu}_b + \sigma_{b_i} \hat{\underset{\sim}{e}}_i) - f(\underset{\sim}{\mu}_b)\right)\left(\quad\right)^T$$

which is the same formula (for large N anyway) since $f_j = \bar{f}_j$ (without the jth pt) if the curve went right through the jth data point (no change).

The $\frac{N-1}{N}$ factor comes from deg. of freedom issues.

Run example

Note that, just like other methods, it will get the error wrong if there is any data covariance ⇒ It's assuming independence!

It does not require residuals to have same magnitude – but if they aren't the same, use weighted regression!

ex 19

```
clear
format compact
echo on
%%The Jackknife
%In this example, we demonstrate the jackknife - a way of estimating the
%matrix of covariance of a set of fitted parameters without requiring
%modeling of the residuals directly.  We do this by solving the problem
%over and over, leaving out one data point each time.  We then calculate
%the matrix of covariance of the fitted values.  This has the advantage of
%not relying on the residuals being normally distributed, although if the
%residuals are not of uniform error the same issues with bias of the
%regression will occur.

%OK, we shall take as our example the "ball in air" problem we've looked at
%before:
t=[0:.1:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
%which usually contains the exact values:
xexact
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
%So the 2-sigma probability in the 90% range (depending on n-m).


pause

%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
```

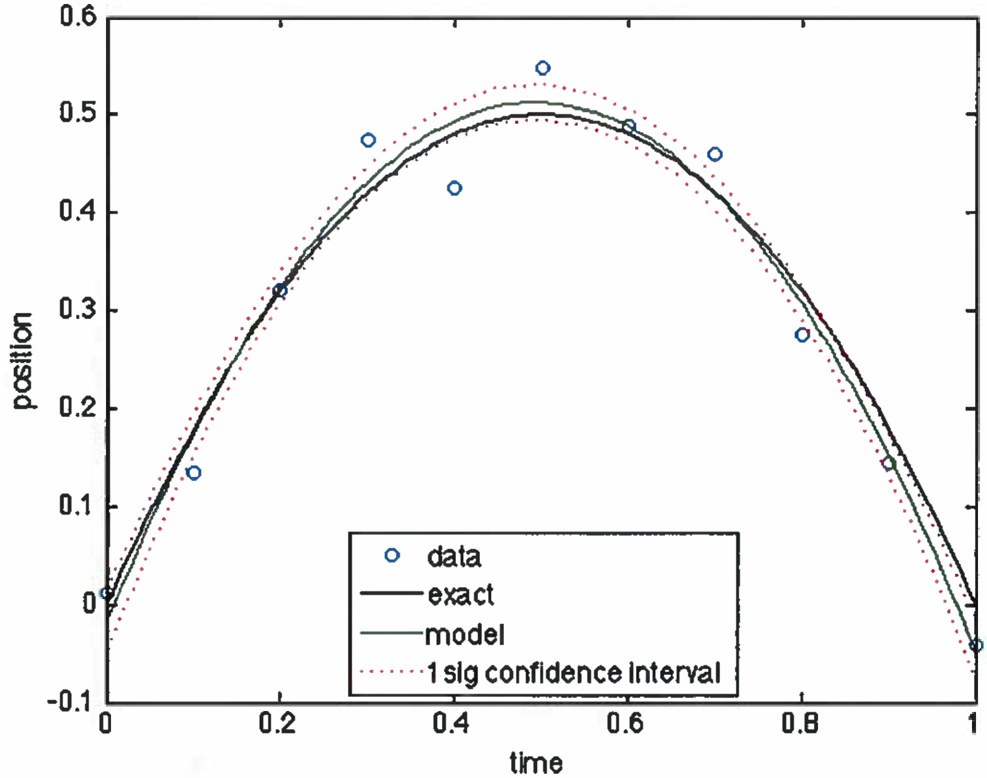
```
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%The Jackknife
%Now we look at how to estimate the matrix of covariance using the jackknife
%approach.  We simply leave off one of the data points and resolve for x.
%We then subtract it from the value obtained by looking at all the points,
%and determine the matrix of covariance:
[n m]=size(a);
xjksqall=zeros(m,m); %We initialize the array.
xjkall=zeros(m,1);
for i=1:n
    iall=[1:n];
    ikeep=find(iall~=i); %We keep all but the ith data point!
    ajk=a(ikeep,:);
    bjk=b(ikeep);
    xjk=ajk\bjk;
    xjksqall=xjksqall+xjk*xjk';
    xjkall=xjkall+xjk;
    echo off
end
echo on
%And we get the final result:
xjk=xjkall/n
%with the matrix of covariance:
varxjk=(xjksqall-n*xjk*xjk')*(n-1)/n
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.

pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxjk./varx

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.

pause

%We can also add this to our plot:
bpjk=ap*xjk;
sigbpjk=diag(ap*varxjk*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bpjk,tp,bp+sigbp,':r',tp,bpjk+sigbpjk,'--g',tp,bp-s
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','jacknife error','Location','South')
```

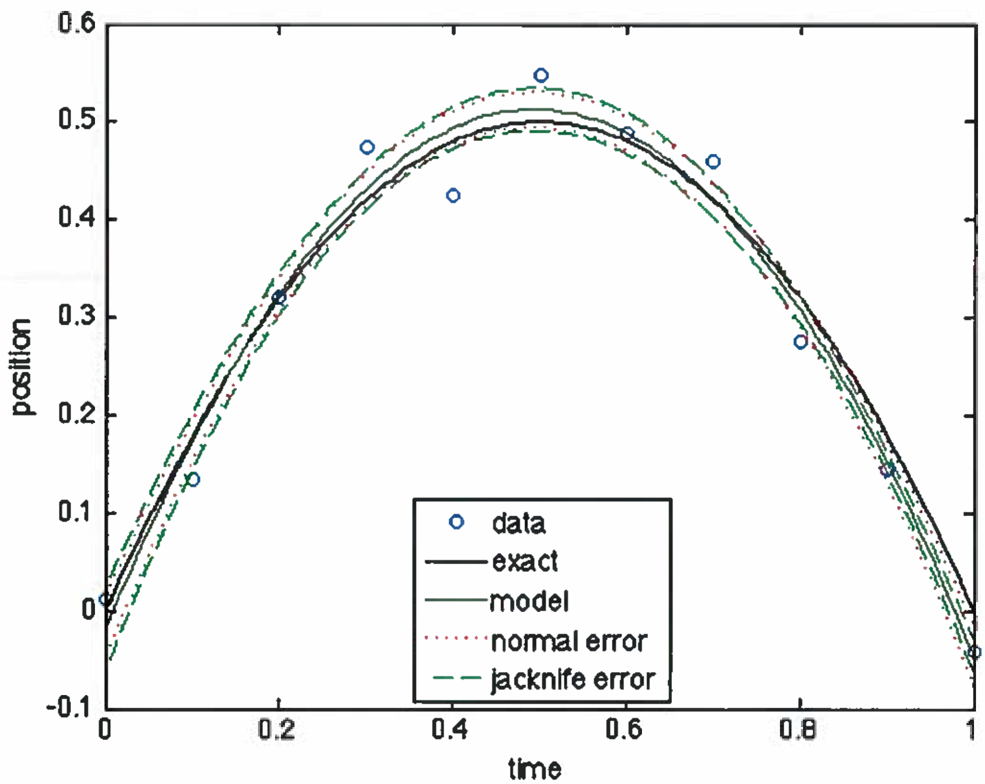
```
pause

%As a final note, the Jackknife will work both for linear and non-linear
%regression, although it does involve solving the problem n times.  It does
%not require determining the residuals (other than assuming independence),
%but you can't use it if you -require- one of the data points in the model
%fitting (such as cr0 in Tuesday's reaction engineering problem).  It also
%may be more prone to "strange results" if the number of data points is
%small, whereas the exact expressions (if the number of data points is
%still sufficient to estimate the variance in the data, or if you can get
%it in other ways) are less so.  For example, if you have a small number of
%data points and "leave off the one on the end", you will tend to get a
%much different value for fitting parameters, leading to (on average) an
%overestimate of the variance.
echo off
```

```
%%The Jackknife
%In this example, we demonstrate the jackknife - a way of estimating the
%matrix of covariance of a set of fitted parameters without requiring
%modeling of the residuals directly.  We do this by solving the problem
%over and over, leaving out one data point each time.  We then calculate
%the matrix of covariance of the fitted values.  This has the advantage of
%not relying on the residuals being normally distributed, although if the
%residuals are not of uniform error the same issues with bias of the
%regression will occur.
%OK, we shall take as our example the "ball in air" problem we've looked at
%before:
t=[0:.1:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b
x =
   -0.0174
    2.1482
   -2.1776

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
sigb =
    0.0401
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
varx =
    0.0009   -0.0035    0.0028
```

```
   -0.0035    0.0202   -0.0187
    0.0028   -0.0187    0.0187
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
xinterval =
   -0.0785    0.0437
    1.8640    2.4325
   -2.4514   -1.9038
%which usually contains the exact values:
xexact
xexact =
     0
     2
    -2
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
prob =
    0.9195
%So the 2-sigma probability in the 90% range (depending on n-m).
pause

%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%The Jackknife
%Now we look at how to estimate the matrix of covariance using the jackknife
%approach.  We simply leave off one of the data points and resolve for x.
%We then subtract it from the value obtained by looking at all the points,
%and determine the matrix of covariance:
[n m]=size(a);
xjksqall=zeros(m,m); %We initialize the array.
xjkall=zeros(m,1);
for i=1:n
    iall=[1:n];
    ikeep=find(iall~=i); %We keep all but the ith data point!
    ajk=a(ikeep,:);
    bjk=b(ikeep);
    xjk=ajk\bjk;
    xjksqall=xjksqall+xjk*xjk';
    xjkall=xjkall+xjk;
    echo off
%And we get the final result:
xjk=xjkall/n
xjk =
```

```
   -0.0189
    2.1549
   -2.1833
%with the matrix of covariance:
varxjk=(xjksqall-n*xjk*xjk')*(n-1)/n
varxjk =
    0.0018   -0.0067    0.0052
   -0.0067    0.0337   -0.0290
    0.0052   -0.0290    0.0258
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.
pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxjk./varx
var_ratio =
    1.9536    1.8994    1.8468
    1.8994    1.6686    1.5467
    1.8468    1.5467    1.3782

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.
pause

%We can also add this to our plot:
bpjk=ap*xjk;
sigbpjk=diag(ap*varxjk*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bpjk,tp,bp+sigbp,':r',tp,bpjk+sigbpjk,'--g',tp,bp-s
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','jacknife error','Location','South')

pause

%As a final note, the Jackknife will work both for linear and non-linear
%regression, although it does involve solving the problem n times.  It does
%not require determining the residuals (other than assuming independence),
%but you can't use it if you -require- one of the data points in the model
%fitting (such as cr0 in Tuesday's reaction engineering problem).  It also
%may be more prone to "strange results" if the number of data points is
%small, whereas the exact expressions (if the number of data points is
%still sufficient to estimate the variance in the data, or if you can get
%it in other ways) are less so.  For example, if you have a small number of
%data points and "leave off the one on the end", you will tend to get a
%much different value for fitting parameters, leading to (on average) an
%overestimate of the variance.
echo off
```





Published with MATLAB® 7.12

Now for the Bootstrap. This is simpler! We have N meas. Let's assume that these are rep. of population of all possible meas! Thus, if we draw N meas at random from inf. dist, can calculate parameters! We do this p times

we thus get p realizations of $\underset{\sim}{x}$!

↳ $\underset{\sim}{x}^{(k)} = k^{th}$ realization

We have:

$$\underset{\sim}{\Sigma}^2_{x_i} = \frac{1}{p-1} \sum_{k=1}^{p} \left(\underset{\sim}{x}^{(k)} \underset{\sim}{x}^{(k)T} - \bar{\underset{\sim}{x}} \bar{\underset{\sim}{x}}^T \right)$$

Note that this runs into trouble if N is small or p is too big! We could just be sampling the same point N times! — prob is $\left(\frac{1}{N}\right)^{N-1}$ though...

ex 20

```
clear
format compact
echo on
%%The Bootstrap
%In this example, we demonstrate the bootstrap technique to estimate the
%matrix of covariance.  It is similar to the jacknife, except this time we
%analyze the data by taking a random sample of data drawn from an
%infinitely replicated data set.  The assumption is that our data is drawn
%from an infinite array of possible observations, and that the actual data
%is a good representation of this data set.

%We use the "ball in air" problem again:
t=[0:.02:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
%which usually contains the exact values:
xexact
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause
```

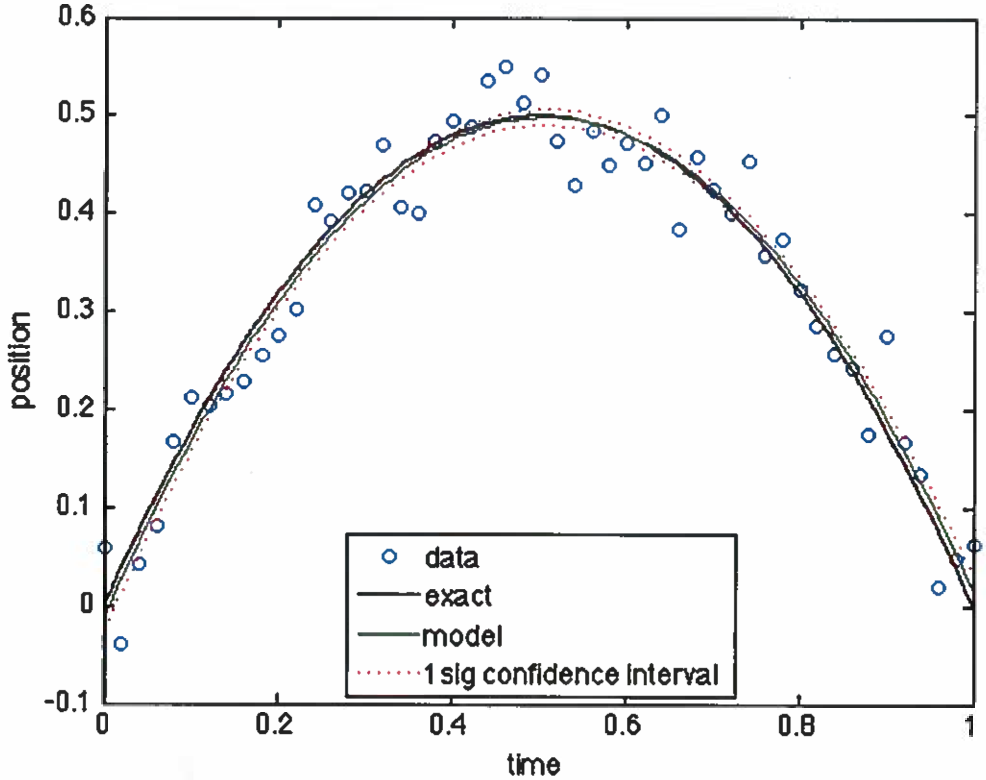
```
%%The Bootstrap
%Now we look at how to estimate the matrix of covariance using the
%bootstrap approach.  We keep our matrix a, but we "sample" it nbs times.
%For this to work, nbs has to be a pretty large number.

[n m]=size(a);
xbssqall=zeros(m,m); %We initialize the array.
xbsall=zeros(m,1);
nbs=100;
for i=1:nbs
    ikeep=ceil(rand(n,1)*n); %The indices we keep this time around
    a_bs=a(ikeep,:);
    b_bs=b(ikeep);
    xbs=a_bs\b_bs;
    xbsall=xbsall+xbs;
    xbssqall=xbssqall+xbs*xbs';
    echo off
end
echo on
%Now we calculate the mean and matrix of covariance of these values:
xbs=xbsall/nbs
%And the covariance:
varxbs=(xbssqall-nbs*xbs*xbs')/(nbs-1)
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.

pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxbs./varx

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.

pause

%We can also add this to our plot:
bpbs=ap*xbs;
sigbpbs=diag(ap*varxbs*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpbs+sigbpbs,'--g',tp,bp-si
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','bootstrap error','Location','South')

pause

%The bootstrap is an interesting approach, but it suffers from the problem
%that you have to solve the problem a fairly large number of times to get a
%reasonable estimate of the variance.  Even more than the jacknife, it runs
%into trouble with small data sets: there is a finite probability that all
%n data points it picks will be the same one, leading to a degenerate
```

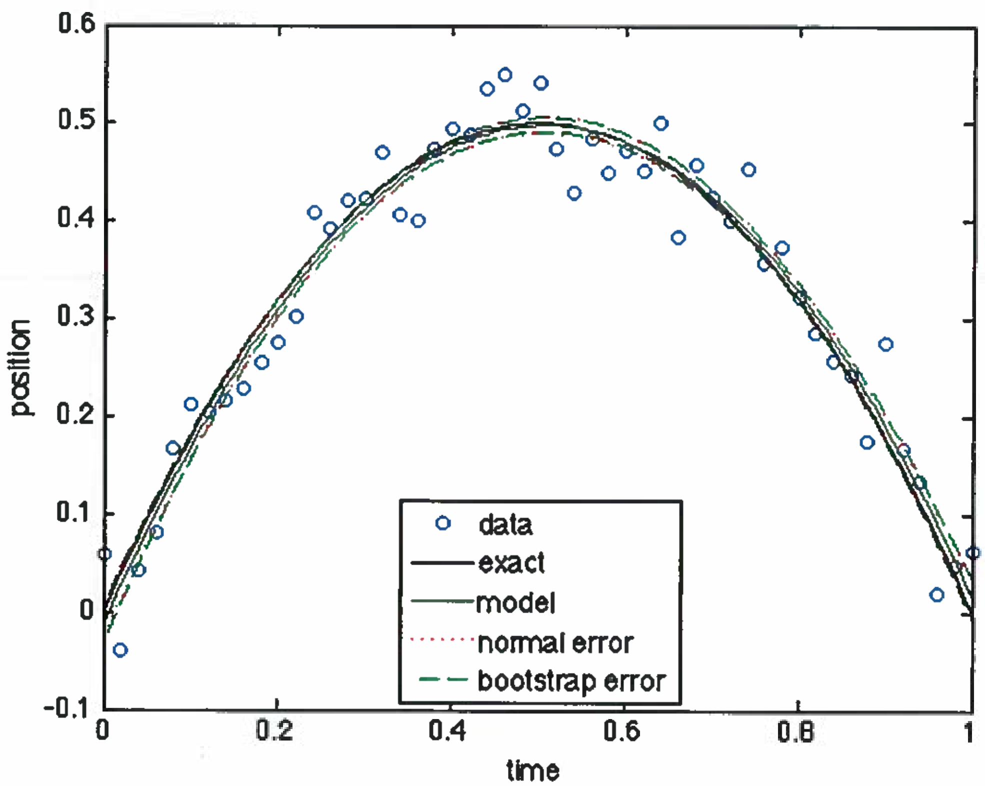
```
%matrix and lots of warning messages!  The variance calculated in this way
%is usually higher than the "correct" variance because of this effect.  If
%both n and nbs are large, however, the variances will be the same.
echo off
```

```
%%The Bootstrap
%In this example, we demonstrate the bootstrap technique to estimate the
%matrix of covariance.  It is similar to the jacknife, except this time we
%analyze the data by taking a random sample of data drawn from an
%infinitely replicated data set.  The assumption is that our data is drawn
%from an infinite array of possible observations, and that the actual data
%is a good representation of this data set.
%We use the "ball in air" problem again:
t=[0:.02:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b
x =
   -0.0121
    2.0129
   -1.9847

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
sigb =
    0.0384
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
varx =
    0.0002   -0.0010    0.0008
   -0.0010    0.0051   -0.0048
    0.0008   -0.0048    0.0048
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
xinterval =
   -0.0431    0.0189
    1.8695    2.1562
   -2.1234   -1.8461
%which usually contains the exact values:
xexact
xexact =
     0
     2
```

```
    -2
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
prob =
    0.9488
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%The Bootstrap
%Now we look at how to estimate the matrix of covariance using the
%bootstrap approach.  We keep our matrix a, but we "sample" it nbs times.
%For this to work, nbs has to be a pretty large number.
[n m]=size(a);
xbssqall=zeros(m,m); %We initialize the array.
xbsall=zeros(m,1);
nbs=100;
for i=1:nbs
    ikeep=ceil(rand(n,1)*n); %The indices we keep this time around
    a_bs=a(ikeep,:);
    b_bs=b(ikeep);
    xbs=a_bs\b_bs;
    xbsall=xbsall+xbs;
    xbssqall=xbssqall+xbs*xbs';
    echo off
%Now we calculate the mean and matrix of covariance of these values:
xbs=xbsall/nbs
xbs =
   -0.0122
    2.0178
   -1.9922
%And the covariance:
varxbs=(xbssqall-nbs*xbs*xbs')/(nbs-1)
varxbs =
    0.0004   -0.0015    0.0012
   -0.0015    0.0069   -0.0063
    0.0012   -0.0063    0.0063
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.
pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxbs./varx
```

```
var_ratio =
    1.6126    1.5491    1.5653
    1.5491    1.3368    1.3211
    1.5653    1.3211    1.3148

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.
pause

%We can also add this to our plot:
bpbs=ap*xbs;
sigbpbs=diag(ap*varxbs*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpbs+sigbpbs,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','bootstrap error','Location','South')

pause

%The bootstrap is an interesting approach, but it suffers from the problem
%that you have to solve the problem a fairly large number of times to get a
%reasonable estimate of the variance.  Even more than the jacknife, it runs
%into trouble with small data sets: there is a finite probability that all
%n data points it picks will be the same one, leading to a degenerate
%matrix and lots of warning messages!  The variance calculated in this way
%is usually higher than the "correct" variance because of this effect.  If
%both n and nbs are large, however, the variances will be the same.
echo off
```





Published with MATLAB® 7.12

For <u>really</u> large data sets, undersampling works! We have N meas.; divide into p sets & compute $\underset{\sim}{X}$ from each! (e.g., $\underset{\sim}{X}^{(k)}$ is $\underset{\sim}{x}$ from k^{th} set.

Thus:

$$\underset{\sim}{\Sigma}_x^2 = \frac{1}{p}\,\frac{1}{p-1}\sum_{k=1}^{p}\left(\underset{\sim}{X}^{(k)}\cdot\underset{\sim}{X}^{(k)T} - \bar{\underset{\sim}{X}}\,\bar{\underset{\sim}{X}}^T\right)$$

↑ extra factor of p since we want error in <u>average</u> of p subsets. We <u>didn't</u> get that for Bootstrap because they were all from the <u>same</u> data set resampled!

ex 21

```
clear
format compact
echo on
%%Undersampling
%In this example, we look at the technique of undersampling.  This is
%really only appropriate for very large data sets.  The idea is that you
%break your dataset into many subsets, calculate the parameters for each,
%and then take the mean and standard deviations of these calculated values.
%Because you need a significant number of data points in each set (for any
%reasonable statistics) and a significant number of subsets, the total
%number of data points has to be really large!

%We use the "ball in air" problem again:
t=[0:.01:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
%which usually contains the exact values:
xexact
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause
```

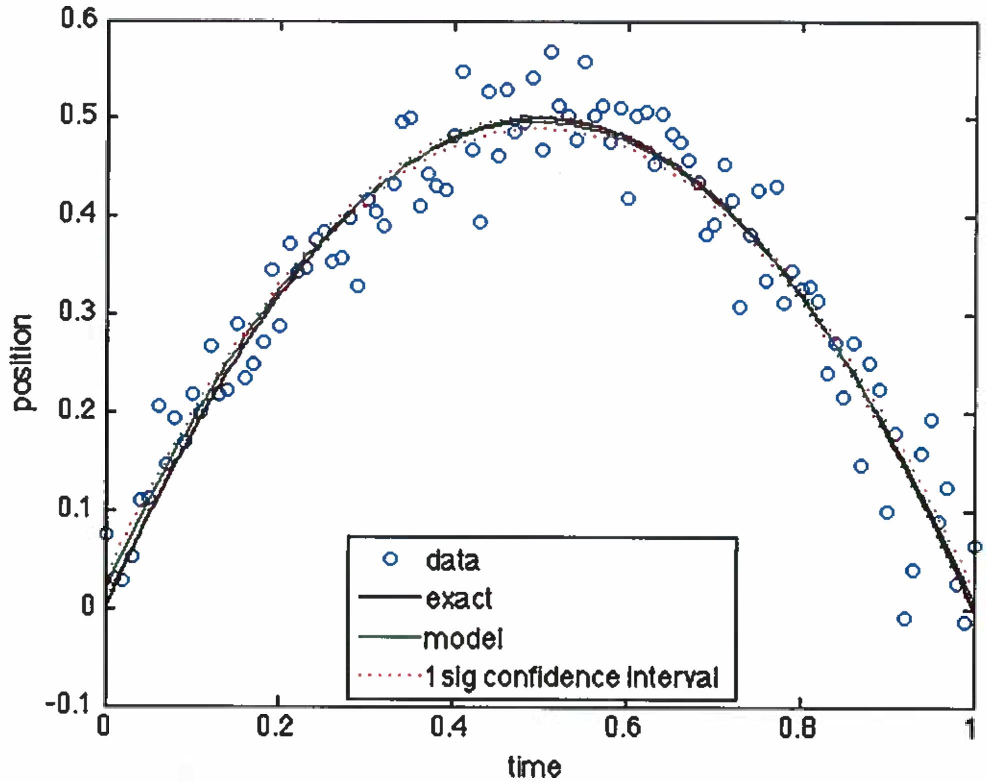
```
%%Undersampling
%Now we estimate our fitting parameters by undersampling.
[n m]=size(a);
xussqall=zeros(m,m); %We initialize the array.
xusall=zeros(m,1);
p=10
for i=1:p
    ikeep=[i:p:n]; %The indices we keep this time around
    a_us=a(ikeep,:);
    b_us=b(ikeep);
    xus=a_us\b_us;
    xusall=xusall+xus;
    xussqall=xussqall+xus*xus';
    echo off
end
echo on
%Now we calculate the mean and matrix of covariance of these values:
xus=xusall/p
%And the covariance:
varxus=(xussqall-p*xus*xus')/(p-1)/p
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.  Note that we are getting the matrix of covariance
%of the -mean- of p samples of our dataset.

pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxus./varx

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.

pause

%We can also add this to our plot:
bpus=ap*xus;
sigbpus=diag(ap*varxus*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpus+sigbpus,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','undersampling error','Location','South

pause

%Undersampling is a very simple way of getting at the variance of
%measurements, but it does require a very large number of datapoints.  You
%certainly can't do it with just a few!  Like the other resampling methods,
%it works as well for both linear and non-linear regression problems, and
%will also behave well for non-normal residuals (although it is
%questionable whether non-weighted regression is appropriate if your
```

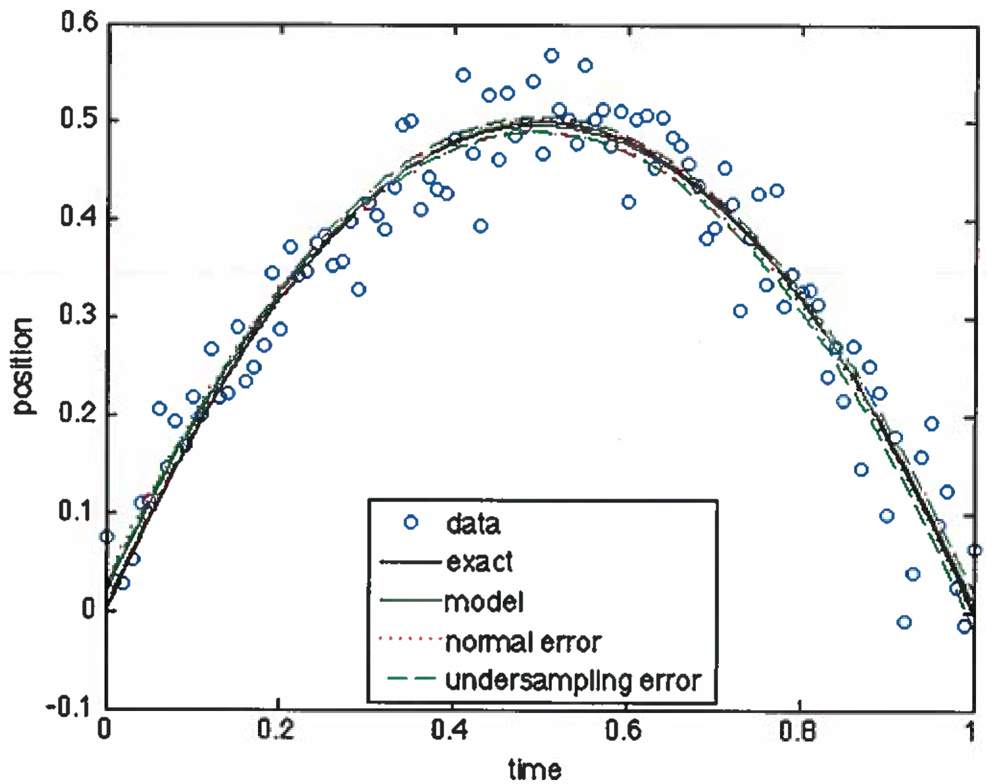
```
%residuals are not uniform).  You can also use these techniques to estimate
%the -distribution- of the fitting parameters: statistics beyond mean and
%variance, as fitting parameters are usually not normally distributed as
%well.
echo off
```

```
%%Undersampling
%In this example, we look at the technique of undersampling.  This is
%really only appropriate for very large data sets.  The idea is that you
%break your dataset into many subsets, calculate the parameters for each,
%and then take the mean and standard deviations of these calculated values.
%Because you need a significant number of data points in each set (for any
%reasonable statistics) and a significant number of subsets, the total
%number of data points has to be really large!
%We use the "ball in air" problem again:
t=[0:.01:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b
x =
    0.0191
    1.9202
   -1.9315

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
sigb =
    0.0440
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
varx =
    0.0002   -0.0007    0.0005
   -0.0007    0.0035   -0.0033
    0.0005   -0.0033    0.0033
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
xinterval =
   -0.0067    0.0448
    1.8013    2.0391
   -2.0466   -1.8165
%which usually contains the exact values:
xexact
xexact =
```

```
     0
     2
    -2
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
prob =
    0.9517
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%Undersampling
%Now we estimate our fitting parameters by undersampling.
[n m]=size(a);
xussqall=zeros(m,m); %We initialize the array.
xusall=zeros(m,1);
p=10
p =
    10
for i=1:p
    ikeep=[i:p:n]; %The indices we keep this time around
    a_us=a(ikeep,:);
    b_us=b(ikeep);
    xus=a_us\b_us;
    xusall=xusall+xus;
    xussqall=xussqall+xus*xus';
    echo off
%Now we calculate the mean and matrix of covariance of these values:
xus=xusall/p
xus =
    0.0160
    1.9405
   -1.9552
%And the covariance:
varxus=(xussqall-p*xus*xus')/(p-1)/p
varxus =
    0.0001   -0.0005    0.0005
   -0.0005    0.0031   -0.0032
    0.0005   -0.0032    0.0035
%And that generates the matrix, without having to assume anything about the
%matrix of covariance of b - other than assuming that the data points are
%independent, of course.  Note that we are getting the matrix of covariance
%of the -mean- of p samples of our dataset.
pause
```

```
%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxus./varx
var_ratio =
    0.4948    0.6851    0.8419
    0.6851    0.8844    0.9665
    0.8419    0.9665    1.0511

%Which is (usually) close to one - it will change every time you run it. If
%you have a large number of data points it will converge exactly to one, as
%expected, but it takes about a thousand or so.
pause

%We can also add this to our plot:
bpus=ap*xus;
sigbpus=diag(ap*varxus*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpus+sigbpus,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','undersampling error','Location','Soutl

pause

%Undersampling is a very simple way of getting at the variance of
%measurements, but it does require a very large number of datapoints.  You
%certainly can't do it with just a few!  Like the other resampling methods,
%it works as well for both linear and non-linear regression problems, and
%will also behave well for non-normal residuals (although it is
%questionable whether non-weighted regression is appropriate if your
%residuals are not uniform).  You can also use these techniques to estimate
%the -distribution- of the fitting parameters: statistics beyond mean and
%variance, as fitting parameters are usually not normally distributed as
%well.
echo off
```





Published with MATLAB® 7.12

Note that *none* of these methods required knowing σ_b^2 (or even if all $\sigma_{b_i}^2$ were the same! We *do* assume independence, though!
You also need decent sized data sets (or huge, for undersampling) to make it work.
For Monte Carlo, you can work w/ *small* data sets, but you need to *know* σ_b!

If we *know* σ_b, simulate by adding in *noise* $\sim \sigma_b$ & recalc. x!

We get $$\underset{\approx}{\Sigma}_x^2 = \frac{1}{p-1} \sum_{k=1}^{p} \left(\underset{\sim}{x}^{(k)} \underset{\sim}{x}^{(k)T} - \bar{\underset{\sim}{x}} \bar{\underset{\sim}{x}}^T \right)$$

(same as bootstrap)

We can do this for a really large # of trials! You can get both $\underset{\sim}{\bar{x}}$, $\underset{\approx}{\Sigma}_x^2$ but <u>also</u> the distribution! Note that if $\underset{\sim}{b}$ is not normally distributed, you can simulate just as readily! simply add same noise as distrib.

Only 2 downsides

1) Requires a <u>lot</u> of comp.

2) You <u>have</u> to know σ_b — but can est. from residual

Also — answer changes every time you run it unless ρ is huge (same prob. as w/ Bootstrap) ex22

```
clear
format compact
echo on
%%Monte Carlo Simulation
%In this example, we look at the approach of Monte Carlo simulation for
%error estimation.  This method requires knowledge of the residual (error)
%in the data set.  If we know this (via calculation or via other
%experiments) we can create "artificial" data sets with data that has an
%added error characterized by this residual.  We determine the fitting
%parameters for these, average them together, and compute the statistics.

%We use the "ball in air" problem again:
t=[0:.1:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
%which usually contains the exact values:
xexact
%Where probabilities are governed by the t-distribution:
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause
```

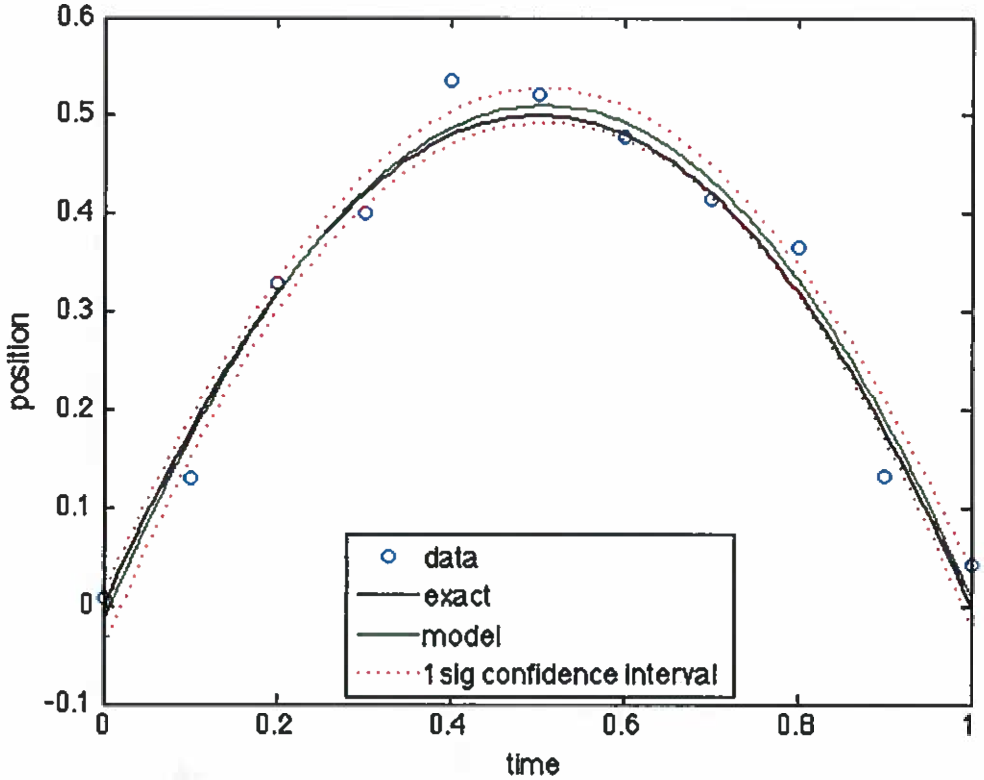
```
%%Monte Carlo
%Now we estimate our fitting parameters from Monte Carlo simulation.  We
%want to use a fair number of samples to get reasonable statistics.
[n m]=size(a);
xmcsqall=zeros(m,m); %We initialize the array.
xmcall=zeros(m,1);
nmc=100
for i=1:nmc
    bmc=b+randn(n,1)*sigb; %We add in noise based on our residuals
    xmc=k*bmc; %We use all the same times, so k doesn't change.
    xmcall=xmcall+xmc;
    xmcsqall=xmcsqall+xmc*xmc';
    echo off
end
echo on
%Now we calculate the mean and matrix of covariance of these values:
xmc=xmcall/nmc
%And the covariance:
varxmc=(xmcsqall-nmc*xmc*xmc')/(nmc-1)
%And that generates the matrix.

pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxmc./varx

%Which is (usually) close to one - it will change every time you run it.
%You don't need a large number of data points, but you do need to have a
%large number of monte carlo simulation runs!

pause

%We can also add this to our plot:
bpmc=ap*xmc;
sigbpmc=diag(ap*varxmc*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpmc+sigbpmc,'--g',tp,bp-sig
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','monte carlo error','Location','South')

pause

%Monte Carlo Simulation is an easy technique for estimating the statistics
%of the fitting parameters.  Unlike other resampling techniques, however,
%it does require knowledge of the residual: exactly the same information
%required of the normal error propagation formulas.  The computational
%requirement is much higher than that of other methods (at least 100
%simulations for decent statistics), and yields the exact same matrix
%of covariance (if the number of simulations is high enough).  There are
%two advantages: it does not require taking any gradients (this may be
%significant in non-linear regression, but is irrelevant in linear
%regression), and it can yield the distribution of the fitting parameters
%(more than just the covariance).  Of the resampling approaches, it is the
```

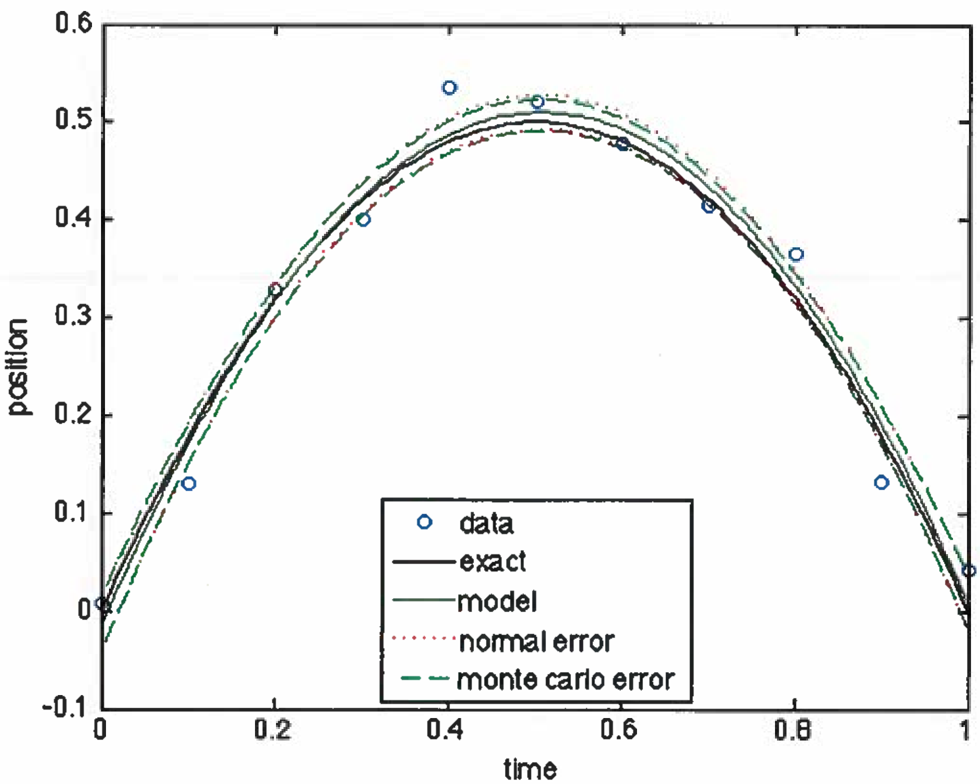
```
%only one which is suitable for small data sets.
echo off
```

```
%%Monte Carlo Simulation
%In this example, we look at the approach of Monte Carlo simulation for
%error estimation.  This method requires knowledge of the residual (error)
%in the data set.  If we know this (via calculation or via other
%experiments) we can create "artificial" data sets with data that has an
%added error characterized by this residual.  We determine the fitting
%parameters for these, average them together, and compute the statistics.
%We use the "ball in air" problem again:
t=[0:.1:1]';
%and we generate some "data":
xexact=[0,2,-2]'; %The exact values
a=[ones(size(t)),t,t.^2];
noise=0.05; %The amplitude of the noise
b=a*xexact+noise*randn(size(t)); %Our artificial data set.

pause

%%Basic Regression:
%We solve this using the usual regression formula:
k=inv(a'*a)*a';
x=k*b
x =
   -0.0145
    2.0749
   -2.0510

%We calculate the error in the usual way:
r=a*x-b;
varb=r'*r/(length(r)-3); %Three degrees of freedom lost
sigb=varb^.5
sigb =
    0.0382
%Which is (usually) close to the value of noise we put in
varx=k*varb*k'
varx =
    0.0008   -0.0032    0.0026
   -0.0032    0.0183   -0.0170
    0.0026   -0.0170    0.0170
%and in particular we have the 2-sigma confidence intervals of the x
%values:
xinterval=[x-2*diag(varx).^.5,x+2*diag(varx).^.5]
xinterval =
   -0.0728    0.0437
    1.8041    2.3458
   -2.3119   -1.7901
%which usually contains the exact values:
xexact
xexact =
     0
     2
    -2
%Where probabilities are governed by the t-distribution:
```

```
prob=tcdf(2,length(b)-3)-tcdf(-2,length(b)-3)
prob =
    0.9195
%So the 2-sigma probability in the 90% range (depending on n-m).
%Now let's plot this up. We want some smooth plotting:
tp=[0:.01:1]';
ap=[ones(size(tp)),tp,tp.^2];
bp=ap*x;
sigbp=diag(ap*varx*ap').^.5;
figure(1)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bp-sigbp,':r')
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','1sig confidence interval','Location','South')

pause

%%Monte Carlo
%Now we estimate our fitting parameters from Monte Carlo simulation.  We
%want to use a fair number of samples to get reasonable statistics.
[n m]=size(a);
xmcsqall=zeros(m,m); %We initialize the array.
xmcall=zeros(m,1);
nmc=100
nmc =
   100
for i=1:nmc
    bmc=b+randn(n,1)*sigb; %We add in noise based on our residuals
    xmc=k*bmc; %We use all the same times, so k doesn't change.
    xmcall=xmcall+xmc;
    xmcsqall=xmcsqall+xmc*xmc';
    echo off
%Now we calculate the mean and matrix of covariance of these values:
xmc=xmcall/nmc
xmc =
   -0.0121
    2.0532
   -2.0312
%And the covariance:
varxmc=(xmcsqall-nmc*xmc*xmc')/(nmc-1)
varxmc =
    0.0008   -0.0026    0.0019
   -0.0026    0.0143   -0.0131
    0.0019   -0.0131    0.0133
%And that generates the matrix.
pause

%Let's compare this to the values we obtained using the error propagation
%formula:
var_ratio=varxmc./varx
var_ratio =
    0.9054    0.7977    0.7268
    0.7977    0.7782    0.7676
    0.7268    0.7676    0.7792
```

```
%Which is (usually) close to one - it will change every time you run it.
%You don't need a large number of data points, but you do need to have a
%large number of monte carlo simulation runs!
pause

%We can also add this to our plot:
bpmc=ap*xmc;
sigbpmc=diag(ap*varxmc*ap').^.5;
figure(2)
plot(t,b,'o',tp,ap*xexact,'k',tp,bp,tp,bp+sigbp,':r',tp,bpmc+sigbpmc,'--g',tp,bp-sic
set(gca,'FontSize',14)
xlabel('time')
ylabel('position')
legend('data','exact','model','normal error','monte carlo error','Location','South')

pause

%Monte Carlo Simulation is an easy technique for estimating the statistics
%of the fitting parameters.  Unlike other resampling techniques, however,
%it does require knowledge of the residual: exactly the same information
%required of the normal error propagation formulas.  The computational
%requirement is much higher than that of other methods (at least 100
%simulations for decent statistics), and yields the exact same matrix
%of covariance (if the number of simulations is high enough).  There are
%two advantages: it does not require taking any gradients (this may be
%significant in non-linear regression, but is irrelevant in linear
%regression), and it can yield the distribution of the fitting parameters
%(more than just the covariance).  Of the resampling approaches, it is the
%only one which is suitable for small data sets.
echo off
```


position
time
data
exact
model
1 sig confidence interval


position
time
data
exact
model
normal error
monte carlo error

As a final note on statistics, we have assumed <u>throughout</u> that the deviation between the observations and the model is <u>random</u>! This is, in general,

Not True!

Thus, you tend to underestimate your error by ignoring the non-zero covariance in your data.

It is <u>important</u> to <u>always</u> plot your residuals to see if there is a <u>systematic</u> deviation.

A systematic deviation means that something is missing from your model and/or there is a systematic error in your data.

<u>Never</u> <u>forget</u> that for large N parameter error is <u>dominated</u> by systematic error!!