

Physics through Computational Thinking
Doctor Auditya Sharma and Doctor Ambar Jain
Department of Physics
Indian Institute of Science Education and Research, Bhopal
Lecture 18
Curve Fitting

(Refer Slide Time: 0:26)

Physics through Computational Thinking

I


Data Analysis - Curve Fitting

Auditya Sharma and Ambar Jain
Dept. of Physics, IISER Bhopal

Outline

In this lecture you will

- 1. learn some curve fitting.*
- 2. apply to concrete examples.*



Hello. So we continue from this fictitious experiment we did last time involving a thought experiment of Galileo if he had access to Mathematica. Today what we will do is go a little bit further and see how we can do some data analysis on the obtained data. So, we saw how we can do statistical analysis last time to get error bars and so on.

So, if you already have the data with error bars with you then and you have a good guess available to the form that a certain function must take, the data must behave according to a certain function. And then you can go ahead and do curve fitting to extract the most optimum parameters. That is the content of the present lecture.

(Refer Slide Time: 1:20)

Galileo's *Mathematica* Experiment!


Let us generate some data from a thought experiment, which we will then analyze by the basic statistical tools we have described above. This is a recurring set of techniques that would come in handy for analyzing a variety of experimental/simulation data.

```
g = 9.8
```

```
data = Prepend[Table[Table[Round[ $\frac{1}{2}$  g i^2 (1 + 0.15 RandomReal[]), 0.01], {i, 1, 10}], {5}],  
  Range[1, 10]] // Transpose;  
data // TableForm
```

Out[1]= 9.8

```
meandata = Table[  
  set = data[[n, 2 ;;]];  
  {n, Mean[set], StandardDeviation[set]/Sqrt[Length[set] - 1]}  
  , {n, 1, 10}  
  ];  
TableForm[meandata];  
meandata[;;, 2 ;; 3];
```



Galileo's *Mathematica* Experiment!

Let us generate some data from a thought experiment, which we will then analyze by the basic statistical tools we have described above. This is a recurring set of techniques that would come in handy for analyzing a variety of experimental/simulation data.


```
g = 9.8
```

```
data = Prepend[Table[Table[Round[ $\frac{1}{2}$  g i^2 (1 + 0.15 RandomReal[]), 0.01], {i, 1, 10}], {5}],  
  Range[1, 10]] // Transpose;  
data // TableForm
```

Out[4]= 9.8

```
Out[6]//TableForm=
```

1	5.44	4.99	5.12	5.38	4.91
2	21.17	20.88	20.52	19.85	21.8
3	46.49	46.71	47.57	48.18	50.36
4	87.49	83.98	83.96	82.04	84.2
5	124.88	137.12	137.33	139.14	131.06
6	193.92	188.88	179.9	186.83	195.25
7	267.74	265.56	270.95	252.39	242.98
8	326.49	340.91	337.02	315.33	334.75
9	430.76	433.86	413.03	454.44	454.45
10	520.51	516.08	555.27	520.12	525.85



Galileo's *Mathematica* Experiment!

Let us generate some data from a thought experiment, which we will then analyze by the basic statistical tools we have described above. This is a recurring set of techniques that would come in handy for analyzing a variety of experimental/simulation data.


```
In[7]:= g = 9.8
```

```
data = Prepend[Table[Table[Round[ $\frac{1}{2} g i^2 (1 + 0.15 \text{RandomReal}[])$ , 0.01], {i, 1, 10}], {10}],
  Range[1, 10]] // Transpose;
data // TableForm
```

```
Out[7]= 9.8
```

```
Out[9]/TableForm=
```

1	5.36	5.03	5.07	5.29	5.14	5.03	5.41	5.59	5.11
2	22.	20.07	21.94	22.06	20.41	19.73	22.33	21.13	22.21
3	47.23	49.41	50.11	45.09	47.37	45.29	45.4	45.77	49.29
4	84.16	86.45	86.56	82.82	84.47	81.85	87.52	87.15	82.38
5	123.09	136.	132.79	137.98	133.12	140.18	128.87	139.40	139.1
6	198.77	178.35	178.74	188.16	193.83	198.65	177.26	189.	188.8'
7	243.87	272.67	273.41	255.2	248.57	251.95	267.54	27	272.2'
8	352.43	327.17	318.6	314.71	346.52	354.01	325.44	3	323.3;
9	436.91	443.13	406.51	397.89	426.02	409.5	407.72		94
10	400.22	515.68	522.47	518.06	510.75	40			




Galileo's *Mathematica* Experiment!

Let us generate some data from a thought experiment, which we will then analyze by the basic statistical tools we have described above. This is a recurring set of techniques that would come in handy for analyzing a variety of experimental/simulation data.

```
In[13]:= g = 9.8;
```

```
data = Prepend[Table[Table[Round[ $\frac{1}{2} g i^2 (1 + 0.15 \text{RandomReal}[])$ , 0.01], {i, 1, 10}], {10}],
  Range[1, 10]] // Transpose;
data // TableForm;
```

```
meandata = Table[
  set = data[[n, 2 ;;]];
  {n, Mean[set], StandardDeviation[set]/Sqrt[Length[set] - 1]}
, {1, 10}
];
TableForm[meandata];
meandata[;;, 2 ;; 3];
Needs["ErrorBarPlots`"];
```



Galileo's *Mathematica* Experiment!


Let us generate some data from a thought experiment, which we will then analyze by the basic statistical tools we have described above. This is a recurring set of techniques that would come in handy for analyzing a variety of experimental/simulation data.

```
In[13]:= g = 9.8;
```

```
data = Prepend[Table[Table[Round[ $\frac{1}{2} g i^2 (1 + 0.15 \text{RandomReal}[])$ , 0.01], {i, 1, 10}], {10}],
  Range[1, 10]] // Transpose;
data // TableForm;
```

```
In[16]:= meandata = Table[
  set = data[[n, 2 ;;]];
  {n, Mean[set], StandardDeviation[set]/Sqrt[Length[set] - 1]}
, {1, 10}
];
TableForm[meandata];
meandata[;;, 2 ;; 3];
```

```
In[17]:= Needs["ErrorBarPlots`"];
- General: ErrorBarPlots` is now obsolete. The legacy version being loaded may conflict with current functions. For
  updating information.
ErrorListPlot[meandata[;;, 2 ;; 3], Joined -> True]
```



```

data // TableForm;

In[16]:= meandata = Table[
  set = data[[n, 2 ;;]];
  {n, Mean[set], StandardDeviation[set] / Sqrt[Length[set] - 1]}
, {n, 1, 10}
];

TableForm[meandata];
meandata[[], 2 ;; 3];

In[17]:= Needs["ErrorBarPlots`"];
- General: ErrorBarPlots` is now obsolete. The legacy version being loaded may conflict with current functionality. See the Compatibility Guide for updating information.

In[18]:= ErrorListPlot[meandata[[], 2 ;; 3], Joined -> True]

```

Out[18]=

So, let us quickly recall what did we do last time. So, we introduce this constant g , and then we describe or we use this little piece of code to generate some spread around $\frac{1}{2}gt^2$. So, I have to use this random real and then I have data which is all these functions, which I had used in a nested form.

So, you can go back and check out the earlier video to understand all of these. And then you see that I have many pieces of data. So, I have just chosen 5 here, of course, you can have more. Maybe let me make it 10 like last time, you can even choose 100. So, it gives you so many columns of data.

So, in order to suppress all this information, then I can, of course, rerun it. Running simply involves shift enter, I can rerun it with putting this semicolon. Alright, so then we have the first step, second step, and then we figured out how to systematically collect all this data, you know, in this form here.

So, we have this n th row and all columns starting from the second column all the way to the end. So, this is the syntax to semicolon semicolon if you just do not mention anything at the other end, it is implied that it goes all the way to the end. And then you notice how we store the data for N mean of the set, standard deviation divided by $\sqrt{\text{Length}[\text{set}]-1}$.

So, once again, this is a key point from last time we emphasized, is that the error bar is contained in the standard deviation divided by square root of the number of data points you have minus 1, right. And so, if you have more data then the error bars are going to shrink,

right. So yeah, so this was the main message from last time, which we also managed to implement. And then you can go ahead and express this in a nice form using table form. And then finally, you can also plot this data.

So, ok so let us, so it says that there is a better newer version available. And so depending upon which version of Mathematica you are running, so you may need to use a different version, but this is something that you can easily figure out, it is simply a matter of what version of Mathematica you are running. So, you will see that if you do this you can extract error bars, so error bars are very tiny. So, that is why I have not exaggerated it like I did last time. So, this is all some things that you will play and learn as you go along.

So today, what we want to do is, of course, we know that this data is going to sit on $\frac{1}{2}gt^2$. After all, we have generated this data and created a spread and then we have worked backwards to extract these error bars. But imagine for a moment that you were Galileo you were just still discovering, uncovering this law. This is the first time you are seeing something like this. You have performed this experiment many, many times and then you collected all this data and you have this curve which looks like quadratic, but is there some more solid way of ascertaining this, and systematic way of extracting some fit parameters? So, that is the subject of our discussion today, right.

(Refer Slide Time: 5:16)

Basic Fitting Theory (Least Squares Minimization Criterion)


Suppose we have N data points $y_1(x_1), y_2(x_2), y_3(x_3), \dots, y_N(x_N)$ with error-bars $\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_N$ respectively. If we have reason to believe that these data points must correspond to a linear curve, our task would then be to extract the best straight line which would correspond to the available data. In other words we wish to find the best parameters a, b such that the linear curve $y = ax + b$ fits the available data as closely as possible. One standard approach for this problem is to consider the quantity

$$\chi^2(a, b) = \frac{(y_1 - ax_1 - b)^2}{\sigma_1^2} + \frac{(y_2 - ax_2 - b)^2}{\sigma_2^2} + \dots + \frac{(y_N - ax_N - b)^2}{\sigma_N^2} \quad (1)$$

and minimize it over a, b .

The above scheme can of course be generalized to more complicated functions than linear. Also several more complicated functions can be recast into a form such that linear regression holds, and this is a standard trick. For example if one is interested in fitting an exponential form, simply taking the logarithm of both the dependent and independent variables then makes it a problem of linear regression. The use of 'logged' quantities.

An enormous amount of theory exists on this topic, and it is definitely not the objective of this course to get into the details of all of this. We simply point out this as an example of the type of philosophy that goes into the fitting routines. Here, we wish to use the case of the fitting functions that are in-built in *Mathematica* to be able to carry out some quick analyses of our data.



So this is, so a slide on the basics of fitting theory. So, actually quite a vast subject and so some people spend a significant portion of their, you know, career in fact, working on these

kinds of models and figuring out optimum strategies and so on. And certainly, if you want to do this in great detail, it would be a whole course by itself and which is certainly not our aim here. So, here we will just get a quick brief glimpse of, you know, how the logic which goes into extracting parameters.

So here, for example, in our case, we know that there is a quadratic form to which we want to fit. Suppose you have some N data points y_1, y_2, y_3 so on up to y_N for x values going from x_1 to x_N . And let us say that each of these data points is associated with error bars, $\sigma_1, \sigma_2, \sigma_3$, so on, σ_N .

And if we already have some guess to make for the type of curve that must pass through this, let us say that you believe that it is a linear curve, it is a straight line that must pass through this and you ask what is the best straight line which would correspond to the available data?

So, then of course, we all know that a straight line is given by $y = ax + b$. So, we want to know what is the best a and best b , given all this information. Can we use all this information and extract these quantities? And so one approach for this is the so called least squares minimization criterion. So, what you do is you go to every point and you compute the deviation from the linear form, right. So you suppose, for the moment you do not fix a or b . So you just keep it as a variable, a and b or any number of variables, in fact, you can have which are fit parameters. So, then you form this function, which is called χ^2 .

So, you find the deviation squared, and then every deviation must be measured in some units. And so, the advantage of having all these error bars is that you have a natural scale to measure it, right. So, you take these deviation squares and divide them by the variance at each of these points or rather I should say, the error bar squared, right.

So, if the error bars are σ_1, σ_2 , so on till σ_N , which is divided by σ_i^2 so on and then you just simply minimize over this function. So, there is a lot of lines, which will look like they are decent good fits right. If you were doing it just by you are trying to sketch a line, you could do it. But this method gives you a quantitative way of obtaining an optimum line which passes through all of these points, right.

And so, of course, one can do more complicated functions. So, in general, so this is what is called a linear approach, linear regressions. It is linear in terms of the fit parameters. So, in

general, actually you can have more complicated functions sitting there, you can choose like a basis of functions, and then with parameters associated with each of them. So, just like we have $ax + b$, you could have a more complicated function involving $ax + b + cx^2$ or plus d times, even more complicated functions.

You can have any number of parameters, but it is going to be linear in all of these parameters. And so then, this approach will work but I mean, you can have more complicated cases, like for example, if you want to fit your data to a function like $e^{\lambda x}$, and λ is your parameter that you want to extract.

So, then it is no longer a linear fitting, it will become an exercise in nonlinear fitting, which is generally much harder and where there is much lesser control involved in the procedure. And so, there is, you know, a lot of discussion on this, I am just giving you a quick sort of overview of how these things go.

But there are many times it is possible to convert these nonlinear type fitting into a linear fitting by, you know, for example, the case that I just mentioned, $e^{\lambda x}$. If you have, you want to fit to $y = e^{\lambda x}$, where λ is the parameter, all you do is instead of fitting y with x , you fit $\log y$ with $\log x$ and then it becomes linear in the variable λ after this, and then you can go ahead and use basic least squares minimization type criterion, right.

So, like I said, it is not the objective here to go into any of these details, so we will just use this as a prescription. So, by the way I should also mention that lot of the times these error bars are not available, you do not have σ_1, σ_2 , so on. In such a case, the standard practice is to just put all of them to be equal to 1.

You simply minimize over the sum of these deviations and that is also a legitimate approach which is carried out, you know, in many of the in-built routines, which are there in lots of programming languages, including Mathematica, which we will come to in a moment. So, but let us work this out the hard way first. So, let us do it for our Galileo's problem.


(Refer Slide Time: 10:55)

Extracting g from fits

```
Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2 ;; 3]]], {a, 1, 10}]
```

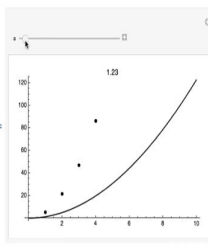
```
testfunc = Table[5 x^2, {x, 1, 10}];  
(meandata[[1, 2]] - testfunc[[1]])^2/meandata[[1, 3]]^2;  
temparray = (meandata[;;, 2] - testfunc)^2/meandata[;;, 3]^2;  
chisq = Total[temparray];  
chisqdata = Table[  
  f[x_] = a x^2;  
  chisq = Total[(meandata[;;, 2] - Table[f[t], {t, 1, 10}])^2/meandata[;;, 3]^2],  
  {a, chisq}, {a, 4, 6, 0.01}];  
ListPlot[chisqdata, Joined -> True];  
parabola[x_] = Fit[meandata[;;, 2], {1, x, x^2}, x]
```

$-0.350067 - 0.2453 x + 5.42889 x^2$




Extracting g from fits

```
In[19]= Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2 ;; 3]]], {a, 1, 10}]
```



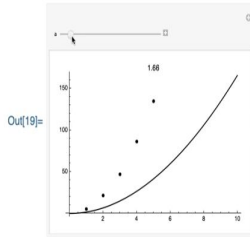
Out[19]=

```
testfunc = Table[5 x^2, {x, 1, 10}];  
(meandata[[1, 2]] - testfunc[[1]])^2/meandata[[1, 3]]^2;  
temparray = (meandata[;;, 2] - testfunc)^2/meandata[;;, 3]^2;
```



Extracting g from fits

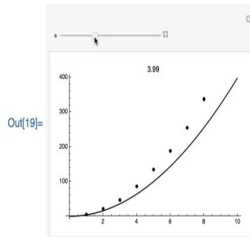
In[19]= Manipulate[Show[Plot[a x², {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2 ;; 3]]], {a, 1, 10}]



```
testfunc = Table[5 x2, {x, 1, 10}];
(meandata[[1, 2]] - testfunc[[1]])2/meandata[[1, 3]]2;
temparray = (meandata[;;, 2] - testfunc)2/meandata[;;, 3]]2;
```

Extracting g from fits

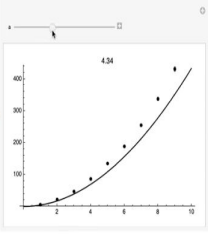
In[19]= Manipulate[Show[Plot[a x², {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2 ;; 3]]], {a, 1, 10}]



```
testfunc = Table[5 x2, {x, 1, 10}];
(meandata[[1, 2]] - testfunc[[1]])2/meandata[[1, 3]]2;
temparray = (meandata[;;, 2] - testfunc)2/meandata[;;, 3]]2;
```

Extracting g from fits

```
In[19]:= Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2]; 3]], {a, 1, 10}]
```



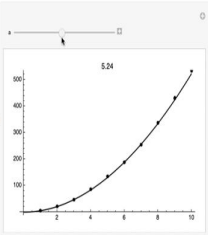
Out[19]=

```
testfunc = Table[5 x^2, {x, 1, 10}];
(meandata[[1, 2]] - testfunc[[1]])^2/meandata[[1, 3]]^2;
temparray = (meandata[;;, 2] - testfunc)^2/meandata[;;, 3]^2;
```



Extracting g from fits

```
In[19]:= Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2]; 3]], {a, 1, 10}]
```



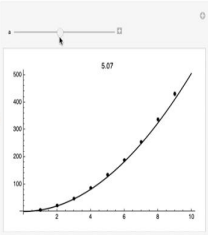
Out[19]=

```
testfunc = Table[5 x^2, {x, 1, 10}];
(meandata[[1, 2]] - testfunc[[1]])^2/meandata[[1, 3]]^2;
temparray = (meandata[;;, 2] - testfunc)^2/meandata[;;, 3]^2;
```



Extracting g from fits

```
In[19]:= Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2]; 3]], {a, 1, 10}]
```



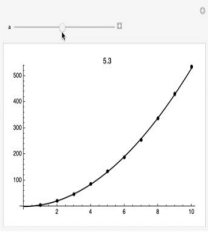
Out[19]=

```
testfunc = Table[5 x^2, {x, 1, 10}];
(meandata[[1, 2]] - testfunc[[1]])^2/meandata[[1, 3]]^2;
temparray = (meandata[;;, 2] - testfunc)^2/meandata[;;, 3]^2;
```



Extracting g from fits

```
In[19]:= Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2]; 3]], {a, 1, 10}]
```



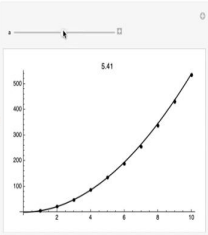
Out[19]=

```
testfunc = Table[5 x^2, {x, 1, 10}];
(meandata[[1, 2]] - testfunc[[1]])^2/meandata[[1, 3]]^2;
temparray = (meandata[;;, 2] - testfunc)^2/meandata[;;, 3]^2;
```



Extracting g from fits

```
In[19]:= Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2]; 3]], {a, 1, 10}]
```



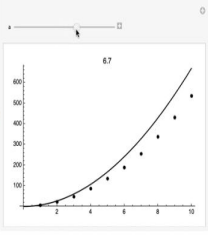
Out[19]=

```
testfunc = Table[5 x^2, {x, 1, 10}];
(meandata[[1, 2]] - testfunc[[1]])^2/meandata[[1, 3]]^2;
temparray = (meandata[;;, 2] - testfunc)^2/meandata[;;, 3]^2;
```



Extracting g from fits

```
In[19]:= Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2]; 3]], {a, 1, 10}]
```



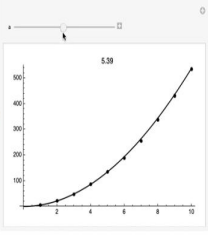
Out[19]=

```
testfunc = Table[5 x^2, {x, 1, 10}];
(meandata[[1, 2]] - testfunc[[1]])^2/meandata[[1, 3]]^2;
temparray = (meandata[;;, 2] - testfunc)^2/meandata[;;, 3]^2;
```

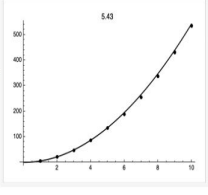


Extracting g from fits

```
In[19]:= Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2 ;; 3]]],
{a, 1, 10}]
```



```
testfunc = Table[5 x^2, {x, 1, 10}];
(meandata[[1, 2]] - testfunc[[1]])^2 / meandata[[1, 3]]^2;
temparray = (meandata[;;, 2] - testfunc)^2 / meandata[;;, 3]^2;
```



```
In[26]:=
testfunc = Table[5 x^2, {x, 1, 10}]
(meandata[[1, 2]] - testfunc[[1]])^2 / meandata[[1, 3]]^2
temparray = (meandata[;;, 2] - testfunc)^2 / meandata[;;, 3]^2
```

```
Out[26]= {5, 20, 45, 80, 125, 180, 245, 320, 405, 500}
```

```
Out[27]= 8.01373
```

```
Out[28]= {8.01373, 27.0999, 8.8178, 48.6677, 54.2252, 6.08886, 15.6089, 14.0877, 19.5}
```

```
chisq = Total[temparray];
```

So, before we do this, actually, it is very nice this function called manipulate that Mathematica has. So, what we can do is actually, so we already know that roughly this looks quadratic. So, can I take, so my goal is I do not know what is this factor a with respect to which I should multiply x square.

So, what I will do is I will on the same graph show $a x^2$, plot of $a x^2$, x going from 0 to in all this is just some syntax. And alongside, I am also going to plot the data that I have, the data which has come from my numerical experiment in this case, but in general it can be hard data.

And then what I can do is I can use this very nice feature in Mathematica to keep on changing this value of a and I see that as I increase a , it seems that my data is coming closer and

closer to the, the two of them are matching. And so, it seems that if I had chosen my a to be 5.25 maybe around 5.26 to 5.3, let us say the fit seems to be very good.

So, I could just guess that my results are consistent with the expression $5.3x^2$. So, the whole point of this exercise is to see if we can do better, if we can make this more quantitative. And if you want to do that, of course the chi-squared approach, that I just mentioned, is the way to go.

And let us see how we can do it using Mathematica, the hard way. So, what do we want? Let us say we define an array. Table command is very useful, involving so as a ballpark, I just take it as $5x^2$, it is 5.3. So, but it is, let me just take $5x^2$ and generate a bunch of data and I want to be able to compare this data with my mean data.

So, the mean data or this array has information from my raw data that I have. I also have error bars, which are sitting in the third column. So, what I can do is, I can compare my mean data first row 1,2, so the second column has all the information about, you know, all rows and second column.

So, I create an array of this console, let me actually go over this step by step. So, if I were to just take just one element, I can do mean data minus test function of 1. So, the first number in this whole list corresponds to this first element of this matrix that I have 1, 2 the first row second column corresponds to this.

So, I subtract this, square it and then I take the error bar which is located in the third column, my first row third column squared. So, it is exactly like the prescription which is described. So, this is what I want to do. And I want to be able to do it for every element of this, for all the rows here. So, I take my rows starting from the first row all the way up to the last row.

So, I put nothing to the left side or to the right side of these double semicolons, which means that I cover all the rows and I go to the second column. So, if you are a bit confused about this, go back and watch the earlier video. That is one way and another is of course, you can just search the documentation of Mathematica to figure out how to deal with these arrays, how to call these, you know, from which row to which row and so on.

So, the nice thing about Mathematica is that you can take this entire array and do these operations as if you are acting upon numbers, right. So, if you take the difference of these two

arrays, testfunc is an array and then this mean data of all the rows, comma second column is one array.

So, it will just take the difference of these two for each element. And then not only that you can directly go ahead and square this whole array.

And what does it do? Mathematica will automatically operate on every single element of this array and then it is also legit to go ahead and directly divide by sigma square, you know, of the whole array we are saying, but it will automatically take care to do this division for each element one after another in that order, right. It is very important. So, if you were using some more a language like C, where you have more control, arguably, but also it involves more lines of code.

So, Mathematica is typically not as time efficient as a language like C or Fortran, but it gives you so much more convenience and you do not necessarily need your code to be like super-efficient for all applications. So, there are certain contexts in which it is better to go to a language like C or Fortran or Python or whatever.

Or if you want something very quick and particularly if you are learning programming for the first time, if you are just getting introduced to programming, for sure this is much more hands-on. Okay so, you can, so instead of putting in a for loop like you would in a more traditional type of language, you just operate like this. Take an array, subtract two arrays, square an array everything, all those kinds of operations can be done, ok.

(Refer Slide Time: 16:51)



Basic Fitting Theory (Least Squares Minimization Criterion)

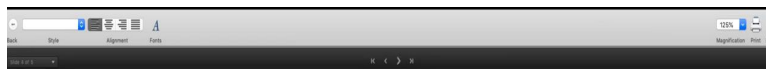
Suppose we have N data points $y_1(x_1), y_2(x_2), y_3(x_3), \dots, y_N(x_N)$ with error-bars $\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_N$ respectively. If we have reason to believe that these data points must correspond to a linear curve, our task would then be to extract the best straight line which would correspond to the available data. In other words we wish to find the best parameters a, b such that the linear curve $y = ax + b$ fits the available data as closely as possible. One standard approach for this problem is to consider the quantity

$$\chi^2(a, b) = \frac{(y_1 - ax_1 - b)^2}{\sigma_1^2} + \frac{(y_2 - ax_2 - b)^2}{\sigma_2^2} + \dots + \frac{(y_N - ax_N - b)^2}{\sigma_N^2} \quad (1)$$

and minimize it over a, b .

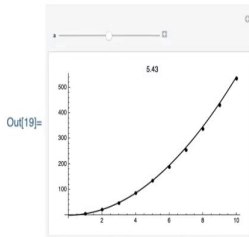
The above scheme can of course be generalized to more complicated functions than linear. Also several more complicated functions can be recast into a form such that linear regression holds, and this is a standard trick. For example if one is interested in fitting to an exponential form, simply taking the logarithm of both the dependent and independent variables then makes it a problem of linear regression of the 'logged' quantities.

An enormous amount of theory exists on this topic, and it is definitely not the objective of this course to get into the details of all of this. We simply point out this as an example of the type of philosophy that goes into the fitting routines. Here, we wish to demonstrate some of the fitting functions that are in-built in *Mathematica* to be able to carry out some quick analyses of our data.



Extracting g from fits

```
In[19]:= Manipulate[Show[Plot[a x^2, {x, 0, 10}, PlotLabel -> a], ErrorListPlot[meandata[;;, 2]; 3]], {a, 1, 10}]
```



```
Out[19]=  
  
In[26]:=  
testfunc = Table[5 x^2, {x, 1, 10}]  
(meandata[[1, 2]] - testfunc[[1]])^2 / meandata[[1, 3]]^2  
temparray = (meandata[;;, 2] - testfunc)^2 / meandata[;;, 3]^2
```



```
In[26]:=  
testfunc = Table[5 x^2, {x, 1, 10}]  
(meandata[[1, 2]] - testfunc[[1]])^2 / meandata[[1, 3]]^2  
temparray = (meandata[;;, 2] - testfunc)^2 / meandata[;;, 3]^2
```

Out[26]= {5, 20, 45, 80, 125, 180, 245, 320, 405, 500}

Out[27]= 8.01373

Out[28]= {8.01373, 27.0999, 8.8178, 48.6677, 54.2252, 6.08886, 15.6089, 14.0877, 19.553, 41.92}

In[29]:= chisq = Total[temparray]

Out[29]= 244.139

```
chisqdata = Table[  
  f[x_] = a x^2;  
  chisq = Total[(meandata[;;, 2] - Table[f[t], {t, 1, 10}])^2 / meandata[;;, 3]^2];  
{a, chisq}, {a, 4, 6, 0.01}];
```



Now, what do we want to do? To calculate. So, I have said in this previous slide, so you must, so I have shown you how to compute the first guy, the second guy, the third guy, all the end guys are stored as different members of this array, which is called temp array. So, in order to find chi square, I just need to sum this array and so simply invoke this function called total. And you are done, basically, that is it.

So now, of course, I want to carry this whole thing out in a compact way and also store all this information for more analysis. So, I will do all of this in one shot here. So, you look at this, I define this entire operation as a chi square data. And so this will create a table for me, it will first of all I will define $f[x]$, this function $f[x]$ with $a x^2$, I will take it to vary from 4 to 6.

Already having used manipulate, I know that it is somewhere around 5.3. So, just to be safe, I will allow my a to vary all the way from 4 to 6, right. So, that is what this command does. I allow my a to go from 4 all the way up to 6 in units of 0.01, then I will just carry out the same thing that I just did earlier here. But with respect to this function $f[t]$, $f[t]$ is defined as, it is the same function, but I am allowing a to vary here. Here I had just fixed a to be 5, but I want to do it for a whole range of values of a .

And I will just carry this same procedure out, it is mean data semicolon semicolon comma 2 minus table of $f[t]$ here. I have $f[t]$, where t itself will go from 1 to 10. So, I can directly do this instead of defining some testfunc called table is equal to table of this and all this, I can directly define a table of $f[t]$ comma t 1 to 10.

And then it is understood that it is an array, I can subtract these two arrays, then I can take the square of this big array which has been formed. And then I can also go ahead and divide by the error bar square. All of this is done in one step, very convenient, very compact code and then I want to store this data. Finally, I care about a and chi square, right. So, that also is done here.

So, it creates a table of, so, the first part here is doing some operations, not storing anything, but when you say put this in these flower brackets a comma chi squared gets stored and the third flower brackets tell you what is the range of values that a takes, a varies from 4 to 6 in units of 0.1.

(Refer Slide Time: 19:38)

```

Out[27]= 8.01373

Out[28]= {8.01373, 27.0999, 8.8178, 48.6677, 54.2252, 6.08886, 15.6089, 14.0877, 19.553, 41.9766}

In[29]:= chisq = Total[temparray]
Out[29]= 244.139

In[30]:= chisqdata = Table[
    f[x_] = a x^2;
    chisq = Total[(meandata[;;, 2] - Table[f[t], {t, 1, 10}])^2 / meandata[;;, 3]^2];
    {a, chisq}, {a, 4, 6, 0.01}]
Out[30]= {{4., 4144.27}, {4.01, 4081.41}, {4.02, 4019.03}, {4.03, 3957.13}, {4.04, 3895.71}, {4.05, 3834.78},
{4.06, 3774.32}, {4.07, 3714.35}, {4.08, 3654.86}, {4.09, 3595.86}, {4.1, 3537.33},
{4.11, 3479.29}, {4.12, 3421.73}, {4.13, 3364.65}, {4.14, 3308.05}, {4.15, 3251.94},
{4.16, 3196.31}, {4.17, 3141.16}, {4.18, 3086.49}, {4.19, 3032.3}, {4.2, 2978.6},
{4.21, 2925.38}, {4.22, 2872.64}, {4.23, 2820.38}, {4.24, 2768.6}, {4.25, 2717.31}, {4.26, 2666.5},
{4.27, 2616.17}, {4.28, 2566.32}, {4.29, 2516.95}, {4.3, 2468.07}, {4.31, 2419.67}, {4.32, 2371.75},
{4.33, 2324.31}, {4.34, 2277.36}, {4.35, 2230.88}, {4.36, 2184.89}, {4.37, 2138.8},
{4.38, 2094.36}, {4.39, 2049.81}, {4.4, 2005.75}, {4.41, 1962.17}, {4.42, 1919.07},
{4.43, 1876.45}, {4.44, 1834.32}, {4.45, 1792.66}, {4.46, 1751.49}, {4.47, 1710.37},
{4.48, 1670.6}, {4.49, 1630.87}, {4.5, 1591.63}, {4.51, 1552.07}, {4.52, 1512.47},

```

```

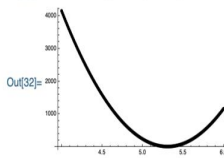
Out[27]= 8.01373
Out[28]= {8.01373, 27.0999, 8.8178, 48.6677, 54.2252, 6.08886, 15.6089, 14.0877, 19.553, 41.9766}

In[29]:= chisq = Total[temparray]
Out[29]= 244.139

In[31]:= chisqdata = Table[
  f[x_] = a x^2;
  chisq = Total[(meandata[ ;; , 2] - Table[f[t], {t, 1, 10}])^2/meandata[ ;; , 3]^2];
  {a, chisq}, {a, 4, 6, 0.01}];

In[32]:= ListPlot[chisqdata, Joined -> True]

```



```

Out[32]=
parabola[x_] = Fit[meandata[ ;; , 2]], {1, x, x^2}, x]

```

```

Out[27]= 8.01373
Out[28]= {8.01373, 27.0999, 8.8178, 48.6677, 54.2252, 6.08886, 15.6089, 14.0877, 19.553, 41.9766}

In[29]:= chisq = Total[temparray]
Out[29]= 244.139

In[31]:= chisqdata = Table[
  f[x_] = a x^2;
  chisq = Total[(meandata[ ;; , 2] - Table[f[t], {t, 1, 10}])^2/meandata[ ;; , 3]^2];
  {a, chisq}, {a, 4, 6, 0.01}];

ListPlot[chisqdata, Joined -> True];

parabola[x_] = Fit[meandata[ ;; , 2]], {1, x, x^2}, x]
-0.350067 - 0.2453 x + 5.42889 x^2

Show[Plot[parabola[x], {x, 0, 10}], ErrorListPlot[meandata[ ;; , 2]; 3], Joined

```

$$\frac{5.03 - 9.8/2}{9.8}$$

```

In[26]:=
testfunc = Table[5 x^2, {x, 1, 10}]
(meandata[[1, 2]] - testfunc[[1]])^2/meandata[[1, 3]]^2
temparray = (meandata[ ;; , 2] - testfunc)^2/meandata[ ;; , 3]^2
Out[26]= {5, 20, 45, 80, 125, 180, 245, 320, 405, 500}

Out[27]= 8.01373
Out[28]= {8.01373, 27.0999, 8.8178, 48.6677, 54.2252, 6.08886, 15.6089, 14.0877, 19.553, 41.9766}

In[29]:= chisq = Total[temparray]
Out[29]= 244.139

In[31]:= chisqdata = Table[
  f[x_] = a x^2;
  chisq = Total[(meandata[ ;; , 2] - Table[f[t], {t, 1, 10}])^2/meandata[ ;; , 3]^2];
  {a, chisq}, {a, 4, 6, 0.01}];

In[32]:= ListPlot[chisqdata, Joined -> True]

```

So, let me go ahead and run this. So, there you go. So, I have all this data has been created. So, it is just, if it were 4, then my chi squared is 4144.27, 4.01, another value and so on. So, I want to visualize this information. So, let me plot this. There you go, so there you see that it is not a surprise that around 5.3. So, I can of course, zoom in on this, I have all the data here for any value of a , I can go ahead and pick up the chi squared. So, then I would go to the minimum here and identify that as the best fit using linear regression approach.

So, there are ways of defining some notion of an error bar for a itself and that error bar would be some, you know some percentage increase in chi square about the minimum point and so on. So, that is the advantage of doing it the hard way is it also gives you some idea of not only an estimate of this parameter, but also you can get some spread around it as well, right.

So, and so the theory is quite involved so you can do more complicated fancy things like goodness of fit parameters, and so on, but let us not go into any of that here. So, our goal here is to show you that there is a simple prescription, we just told you what that prescription is. And then we told you how using Mathematica, you can actually go ahead and carry out this implementation yourself, you could play this type of a game out for some other experimental data that you have, you can just import all the data, and then you have a functional form to which you want to fit.

First, you have learned how to get error bars, do that, use all of this and make your own chi squared data and get your own error bars. This is one way of doing. First was this is also a very nice way. Manipulate is an approach where you directly get to that number. There were no error bars here, of course but here you have some more information, chi squared is perhaps the most rigorous way of doing this kind of analysis. And then the third way is to just go to Mathematica and say, you do this for me. So, there is an inbuilt function called fit. Fit also uses this kind of a linear approach.

So, I said that, when I defined chi squared, I was trying to fit it to a functional form of the kind $a + bx$. So, in general actually this linear method can work for $af_1 + bf_2 + cf_3$ so on. So, this is as general as it can get and all these basis functions can be very complicated functions, they can be highly nonlinear, they can be whatever, and it would still come under

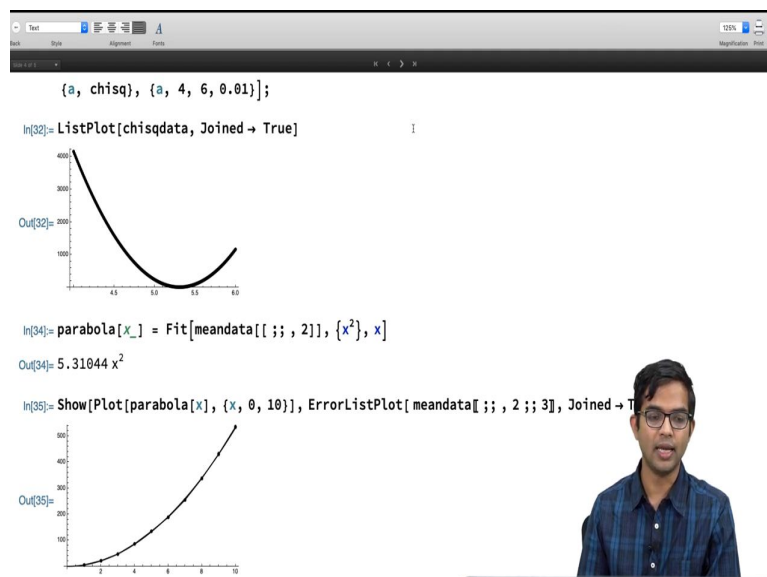
this linear fitting. So, that is what is going on here. So, we can take 1, x and x^2 as the basis functions, and then we have this whole data in here.

The first part comma separated by the basis function, it could be 1, x , x^2 , you can play with this you can put in x^3 , you can put in whatever else. And then the final thing is to just say that you are fitting with respect to. So, the independent variable is x . So, if I were to do this, I get an answer like 3.2509 minus 1.876, so plus 5.48. So, this is doing a fit and giving me some non-zero values for the first two numbers as well.

So, this also tells you that one has to be careful with fitting. You can fit your data to more than one type of function. So, purely based on fitting, one should not come to their conclusions. All the inference should not be, should not hinge purely on fitting to a form which looks like the correct form, there must be some independent way of checking this.

So, let us do this. I mean, in our case, of course, we know that this should involve only x^2 and then of course, you get some value like 5.3. So, we can go ahead and check how this, you know, if I were to use this value of this parabola of x with this fitting form and compare it against my original data, and of course, it is not a surprise that the two of them agree very, very closely and also our own, you know, eyeballing exercise using manipulate also gave us a number around 5.3.

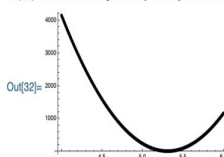
(Refer Slide Time: 24:36)



The screenshot shows a Mathematica notebook interface with the following content:

```
{a, chisq}, {a, 4, 6, 0.01}];
```

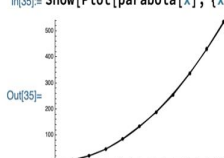
```
In[32]:= ListPlot[chisqdata, Joined -> True]
```

Out[32]= 

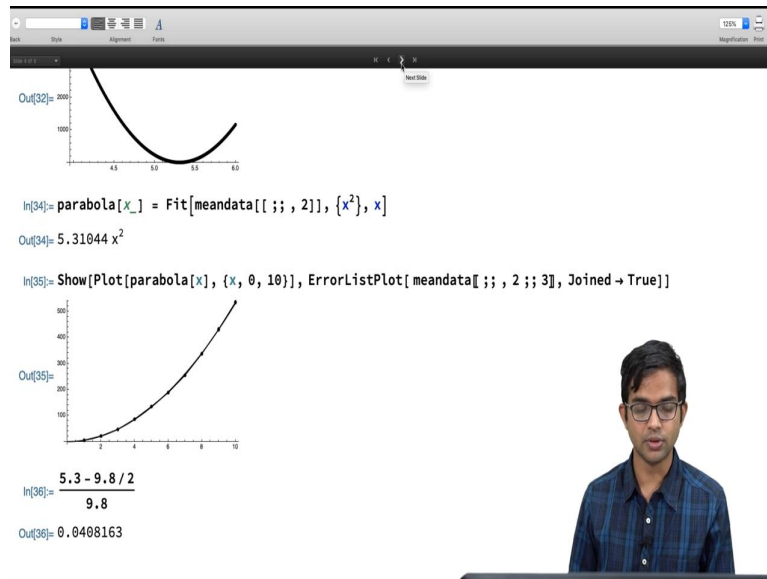
```
In[34]:= parabola[x_] = Fit[meandata[;;, 2]], {x^2, x}
```

Out[34]= 5.31044 x²

```
In[35]:= Show[Plot[parabola[x], {x, 0, 10}], ErrorListPlot[meandata[;;, 2];; 3], Joined -> True]
```

Out[35]= 

A small video inset of a man speaking is visible in the bottom right corner of the notebook window.



So, you can go ahead and check how far away so with this analysis you are really getting at g . It is g by 2. So, g by 2 we know should be 9.2 by 2, g is equal to 9.2. So, let us see, what is the percentage error in our experimental determination of g and it is not too bad and that, of course, has to do with the error bars we had chosen.

So we had chosen error bars of 0.01, 0.15, which is actually not too small. So, in the end it is a, I guess it is a decent outcome for our numerical experiment. But anyway, so the main point of this lecture or this module is to illustrate the method. So, the method is first you have the data which has come in because you already know how to compute errors.


And then you can use this alongside the chi square minimization approach. You have to come up with a sensible guess for the functional form. So, this is something which would involve some other kind of intuition, it is not purely based on how the graph looks.

Sometimes it can be but it should be buttressed or it should be supported, it should be supported by some alternate intuition or some other argumentation. And then once you have a functional form, which you can depend upon, this only tells you what is the best set of parameters which can give you this, which would fit with the data.

(Refer Slide Time: 26:31)


Nonlinear Fits

```
In[37]= fn[x_] = a + b x + c x^2;  
fit = FindFit[meandata[;;, 2], {a + b x + c x^2}, {a, b, c}, {x}];  
fn[x] /. fit;  
Show[Plot[fn[x] /. fit, {x, 0, 10}], ErrorListPlot[meandata[;;, 2]; 3], Joined -> True];
```



Nonlinear Fits

```
fn[x_] = c x^2;  
In[39]= fit = FindFit[meandata[;;, 2], {c x^2}, {c}, {x}]  
- FindFit: The function value  
{-5.255 + c x^2, -21.556 + c x^2, -46.917 + c x^2, -86.23 + c x^2, -134.68 + c x^2, -188.236 + c x^2, -254.778 + c x^2, -337.365 + c x^2, -431. + c x^2, -  
535.061 + c x^2} is not a list of real numbers with dimensions {10} at {c} = {1}.  
Out[39]= FindFit[  
{5.255, 21.556, 46.917, 86.23, 134.68, 188.236, 254.778, 337.365, 431., 535.061}, {c x^2}, {c}, {x}]  
fn[x] /. fit;  
Show[Plot[fn[x] /. fit, {x, 0, 10}], ErrorListPlot[meandata[;;, 2]; 3], Joined ->
```



Nonlinear Fits

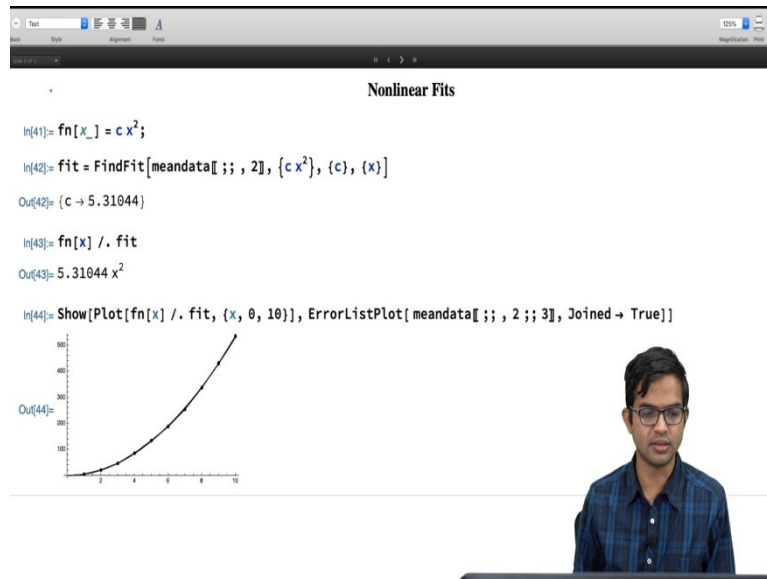
```

In[41]:= fn[x_] = c x^2;
In[42]:= fit = FindFit[meandata[;;, 2], {c x^2}, {c}, {x}]
Out[42]:= {c -> 5.31044}

In[43]:= fn[x] /. fit
Out[43]:= 5.31044 x^2

In[44]:= Show[Plot[fn[x] /. fit, {x, 0, 10}], ErrorListPlot[meandata[;;, 2]; 3], Joined -> True]

```



So, finally, I want to give you a brief description of how the same thing can be carried out using a nonlinear fit. So, if you, so you can take any function here, it is not does not have to be ax or x^2 , you can take any more much more complicated function and fit it with respect to an exponential form, a logarithmic form or whatever you want.

But, like I said, the control involved in the algorithm which goes into generating this. So, here I am going to just use a function called find fit, which is, which comes in-built in Mathematica. Here there is no notion of basis functions or anything, it is just one big complicated function you give and find fit will do it for you.

So, you can ask it to fit it to the functional form $a + bx + cx^2$ using the parameters a , b , c that is the syntax and then you have to also say what is the independent variable here x and you can just go ahead and implement this. So, it tells you that a must be around 3.25, b must be this and so on.

So, here also of course, we could have fit it to just cx^2 if you want it. So, if you can do this and then we say that. So, use of find fit for this kind of a fit would be a bit of an overkill, but it is possible. So let us check how that would. So, I guess it is not very happy. So, let us check what happens if you do bx plus.

So, I think the issue was just with the gap between c and x^2 , so this should work out. Let us also clear this. There you go. So, it is giving us 5.3, which is what we had from the earlier

method also. So yeah, so now you have to use this syntax. So, use the number which has come out of this fit and in this function, $f_n[x]$, and then there you go.

So, then finally we can plot all of this, and it is not at all surprised that the data will sit on top of the experimental data, the raw data that we have. Yeah, so, this is just a summary of this whole lecture. What did we do here? First thing is we come up with an intelligent guess for the functional form, having got the data and got the error bars using the prescription from last time.

Second is we try out the manipulate function. So, you could have more than one fitting parameters. And so you play with all of them and 0 in on some ballpark of these various parameters in which your data would, is likely to fit with the functional form and then you can either use fit, which is a Mathematica program, which can directly do it for you, which is a linear fitting program or you can do more complicated function like find fit, which is a nonlinear one and it should be avoided as far as possible because they are not so stable.

So, it is better to do a linear fitting, and it is better to convert problems which involve nonlinear fitting into linear ones if whenever it is possible by for example, doing some kind of a transformation. So, I did not go into any of that. So, there are I am just telling you the various options. So, there is fit or there is find fit. And then but of course, the most sort of controlled approach is for you to sort of directly do it with by calculating chi square because you have all the error bars. So, if you notice we did not feed in any information about the error bars to Mathematica here.

So, surely the chi squared approach has more information in it and it controls it even better. So, what does the chi squared approach involve? So, you simply compute this chi squared as a function of all these parameters a , b , c , d and how many ever you want and then make a plot of this function as a function of all these parameters and find out the point where it has a minimum. And you can define some kind of an error bar in these parameters based on the percentage increase or chi squared. There is a systematic approach to that as well.

But for our purposes, just getting at the minimum itself is a , is what we want. We have the parameters and then we have some understanding of how the chi squared increases around these points, okay, so that is this lecture. Play out these techniques with some other data that maybe you got in the lab or you can try out your similar numerical experiments come up with

other similar phenomena, extract your own data, create your own data and then analyze them.

Thank you.