Programming, Data Analysis & Machine Learning

Lecture 8: Minimisation & Chi-squared fitting

Peter Clarke

THE UNIVERSITY
of EDINBURGH

Random number generators & Numerical Integration

Aim:

- To write a simple minimiser
- To introduce the use of χ^2
- To use your minimiser to find the best values for parameters “m” and “c” in a straight line fit to data
- To show how to estimate errors on parameters from a minimiser
- To compare to the use of package minimisers

Minimisation

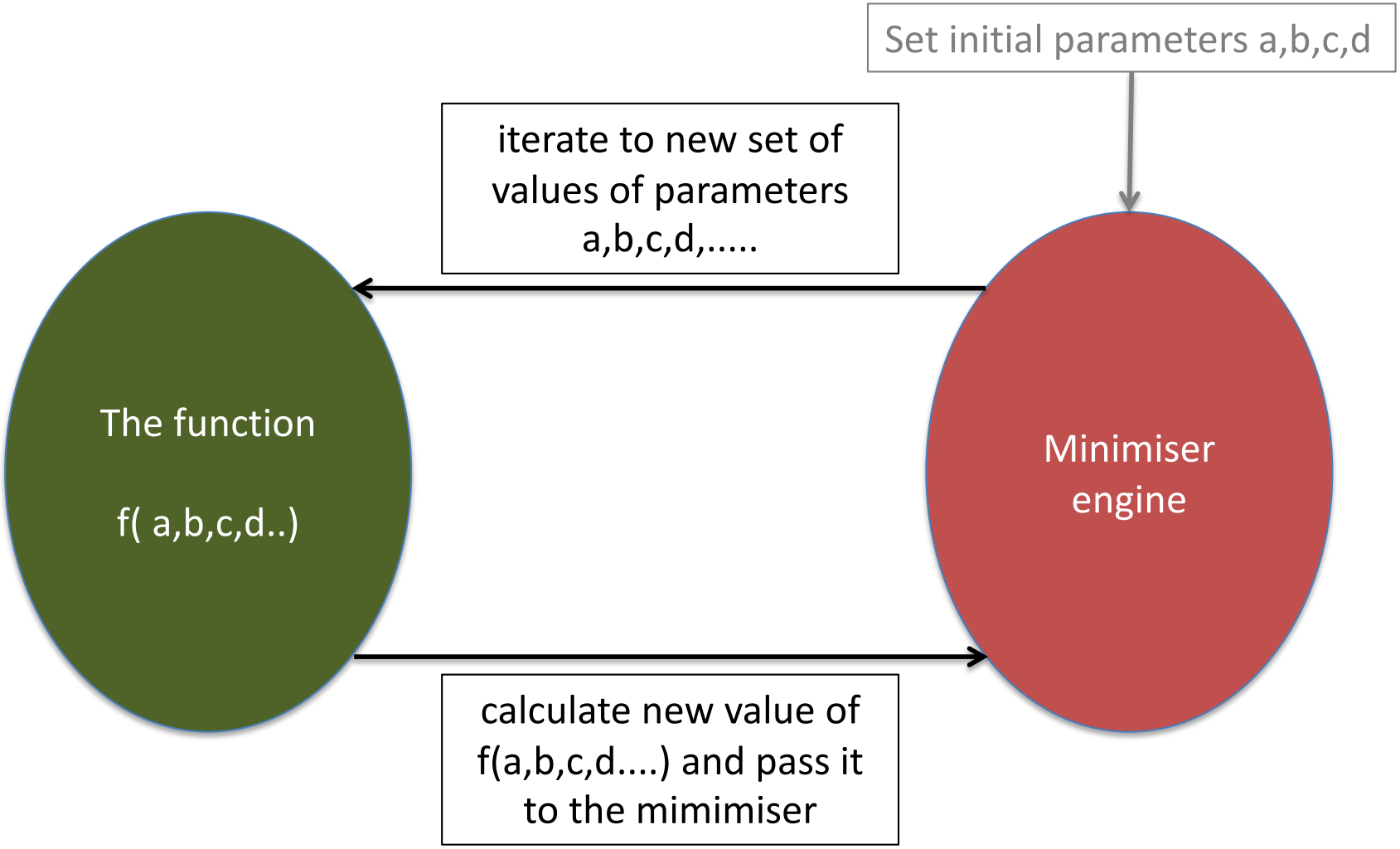
Minimisation

Motivation:

- Situations arise where one needs to find the minimum of some function
  - You have a function f(a,b,c,d.....)
  - You want to find the values of a,b,c,d... for which the function is a minimum
- The situation is such that you cannot do it analytically
- So you can only do it by a “search”
- This is called “minimisation”

Minimisation

- A simple scheme is:



- Continue in a loop until some threshold for convergence at minimum is reached

Minimiser : Interface

Here are some methods you might think of for performing such a minimisation as shown in the picture.

```
Class Minimiser:

    #To set convergence criteria
    setConvergenceLimit( conv)
    setMaxIterations( max )
    setStartParameters( params )

    #Main minimising method
    minimise( valueOfFunction )

    #Methods to report if finished
    isFinished()
```

Minimiser : Interface

```
#Main minimising method
minimise( valueOfFunction )
```

The idea is that you give the minimiser the value of **`valueOfFunction`** (which you calculate) for a given set of parameters.

The minimiser works out where it wants to change the parameters to and returns these to you in a list (this is the return argument of minimise).

You then use these new parameters to calculate a new **`valueOfFunction`**

You use this method in an iterative loop until the minimiser tells you to stop. This is what implements the picture 2 slides ago.

In reality we would probably do something more sophisticated like pass a function pointer so that the minimiser can make the call to calculate valueOfFunction itself.

- Here is a code fragment which does the minimisation loop as depicted earlier

```
function = MyFunction(....)

minim = Minimiser(...)

.........

params = [........]

while( ! minim.isFinished() ):

    valueOfFunction = function.evaluate( params )

    params = minim.minimise( valueOfFunction )

}
```

Minimiser : Interface

Consider the other methods. You will need these or some equivalent.

```
Class Minimiser:

    #To set convergence criteria
    setConvergenceLimit( conv)
    setMaxIterations( max )
    setStartParameters( params )

    #Main minimising method
    minimise( valueOfFunction )

    #Methods to report if finished
    isFinished()
```

Minimisation : Implementation

- It needs to deal with starting the process – and this is typical of the problems you face in writing this sort of code - once its going it is easy, but setting it up at the start needs thinking about

- What are the limits of the parameters should it use ? Its easier if you tell the minimiser the upper and lower limits of each parameter so you might want to add methods for this.

- If you don't give it limits then it is going to have to perform some rough search first to determine a reasonable space to vary the parameters within.

- It is also probably a good idea to first find the approximate minimum using a simple grid search.
  - Divide the space into a regular grid
  - calculate the valueOfFunction for each point
  - find which point is the approximate "best guess" minimum.

Minimisation : Implementation

- Once set up then it has to change the parameters on each iteration to try to get a better lower value for the function returned to it

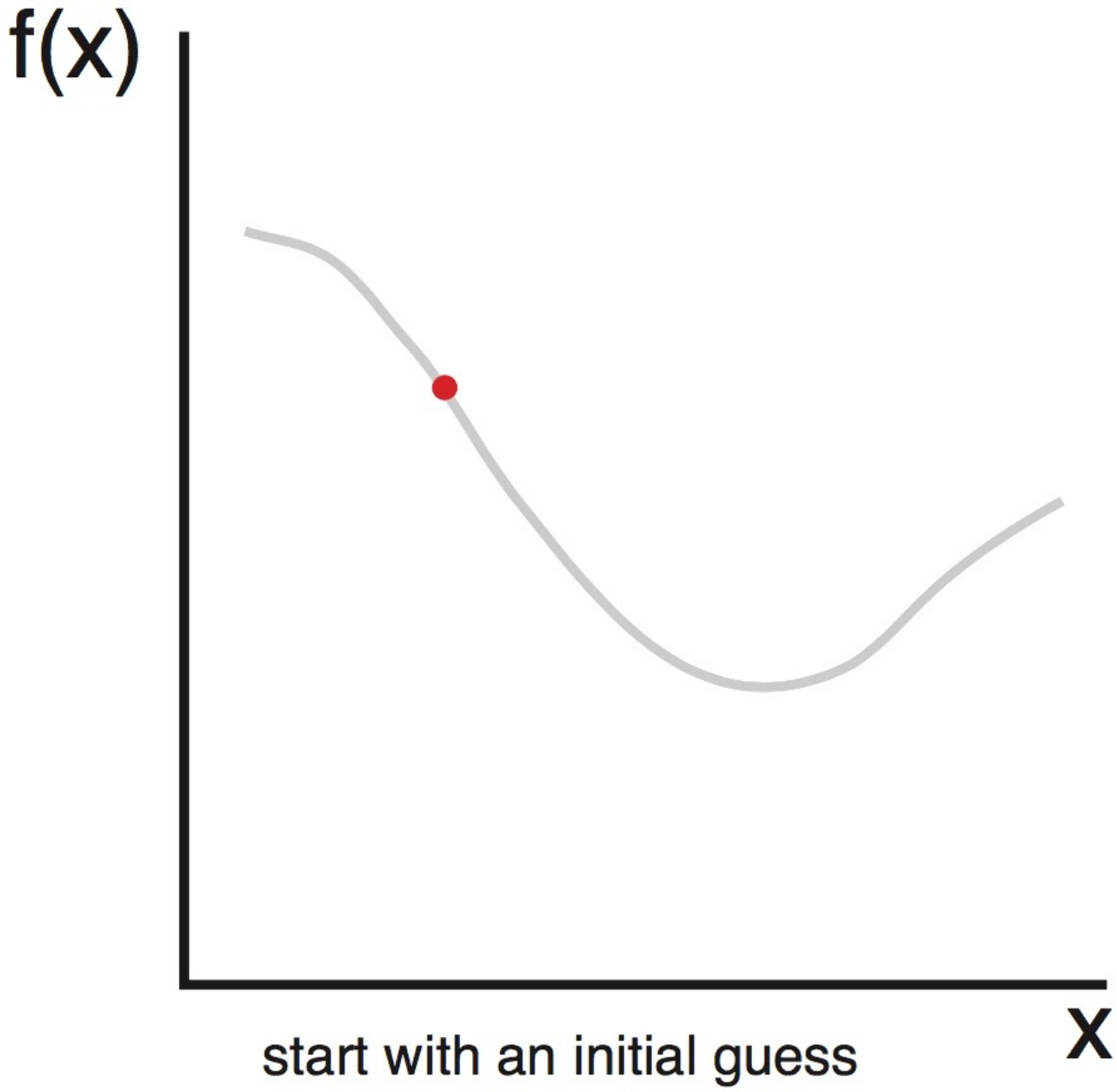
- A simple way to do this is go in one direction from the current “best guess” at the minimum by some interval.

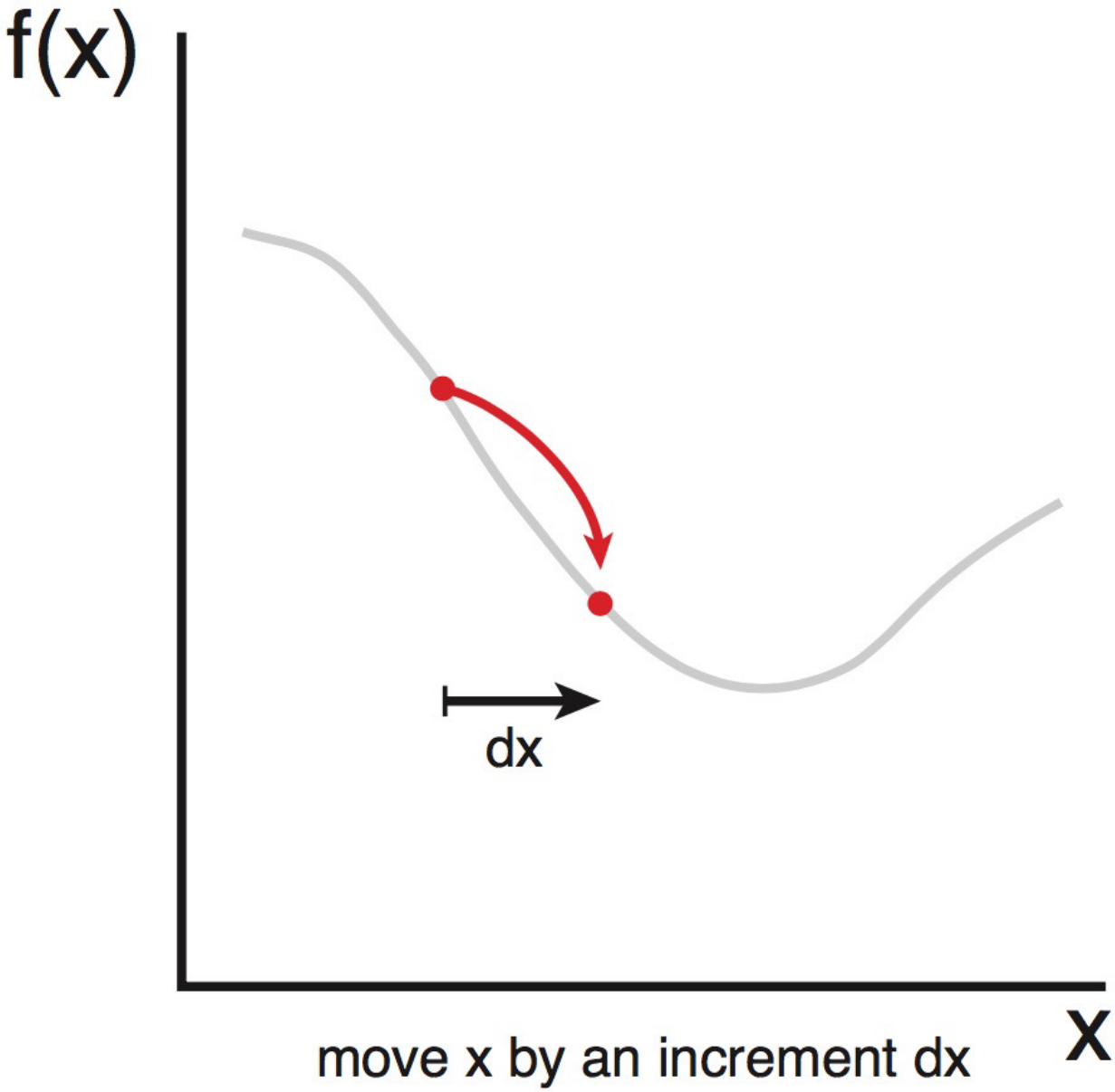
  - If the new point is lower it becomes the next best guess and you repeat the process without changing the direction or interval

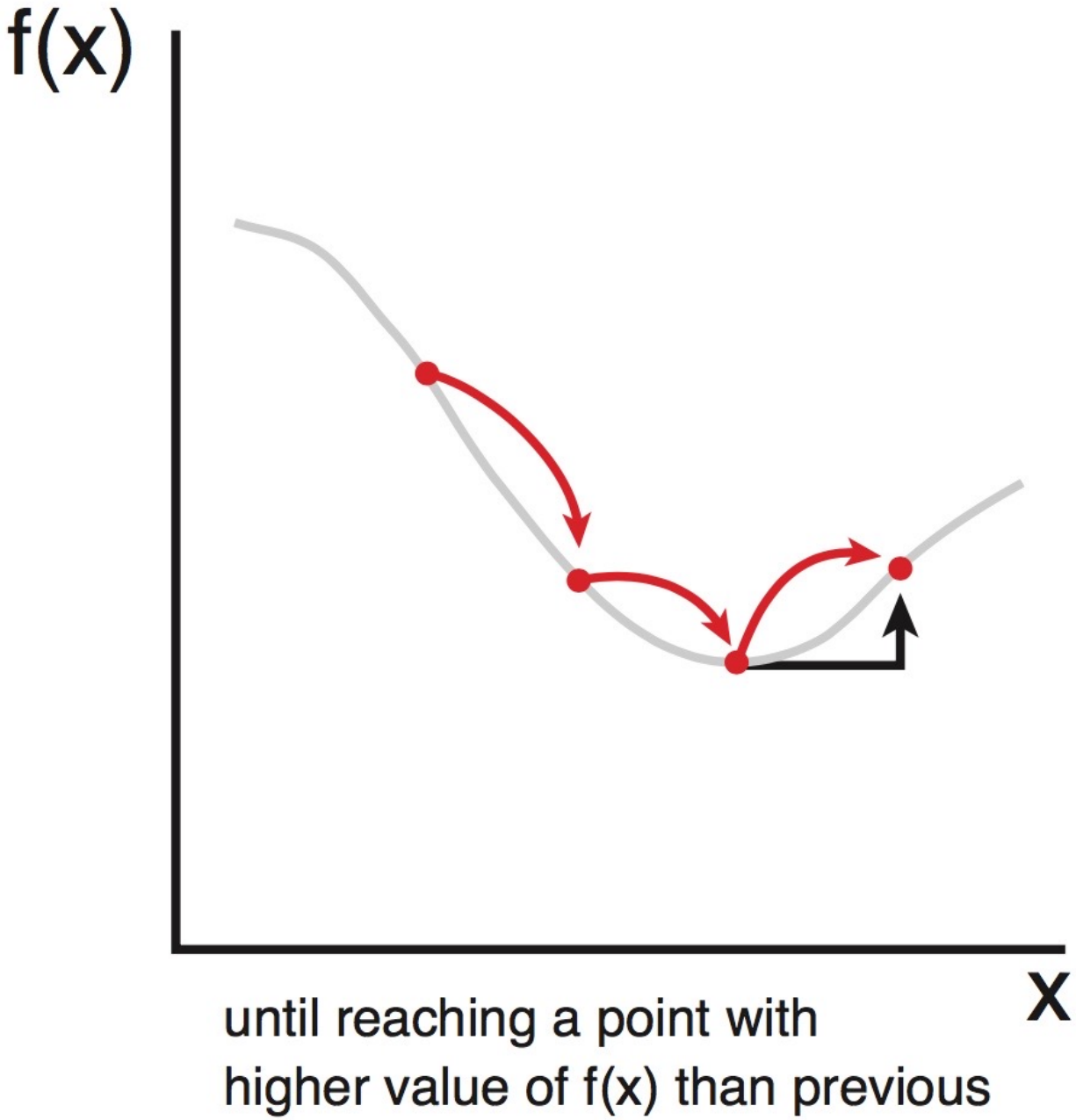
  - If the new point is not lower then halve the interval, reverse direction, and repeat

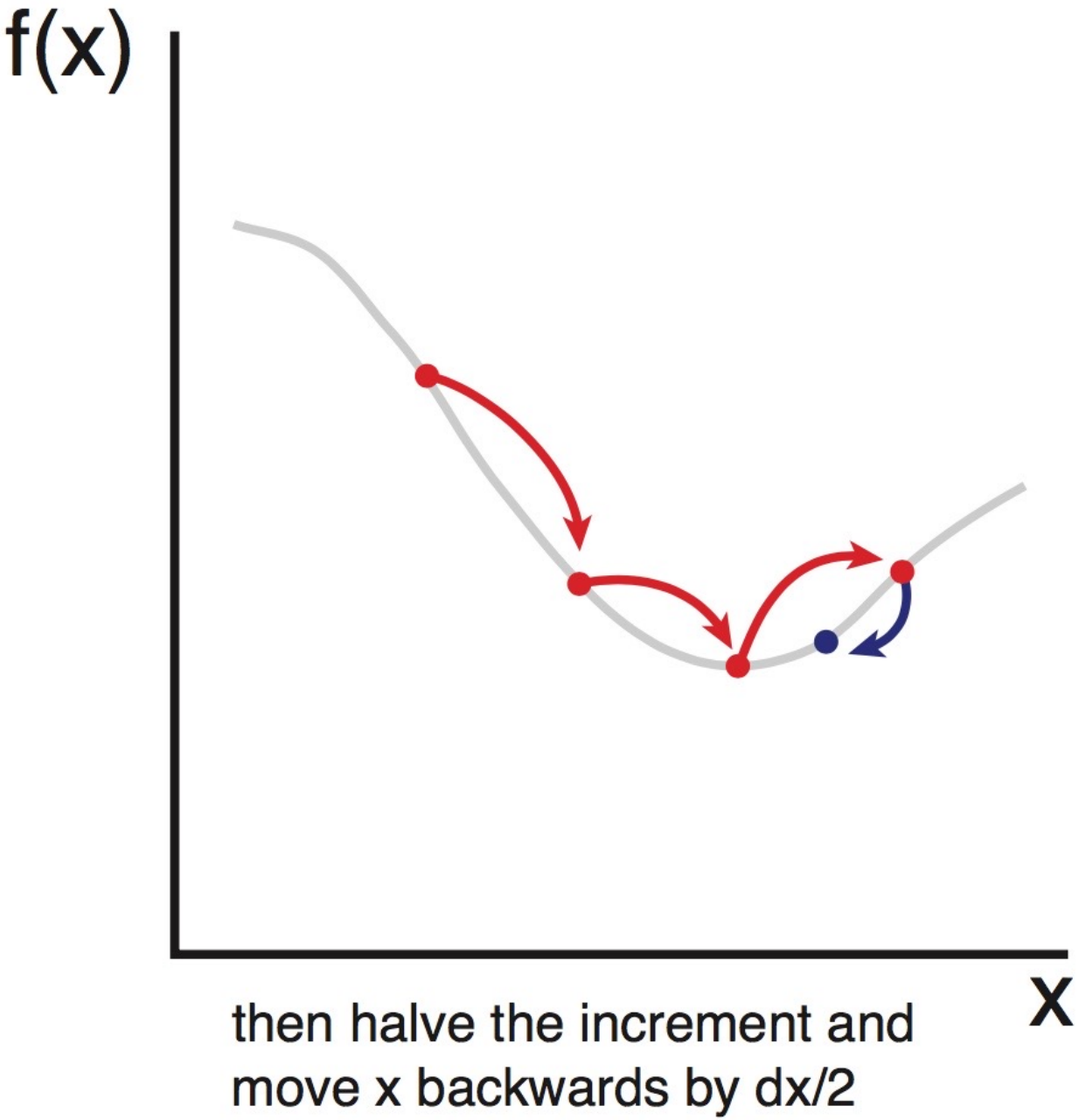
  - keep going until the changes are below some threshold

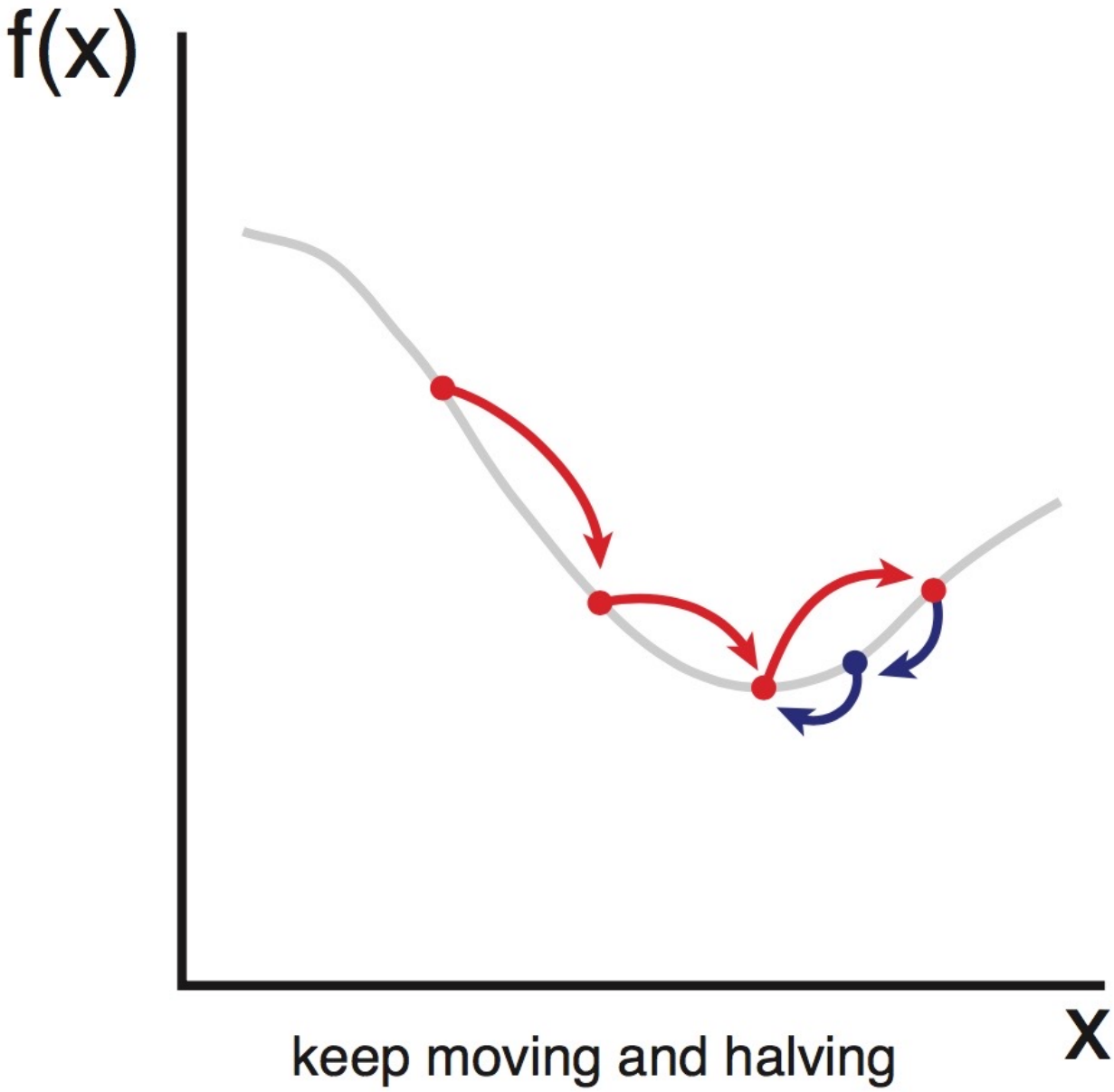
(view pdf as single page to see animation)

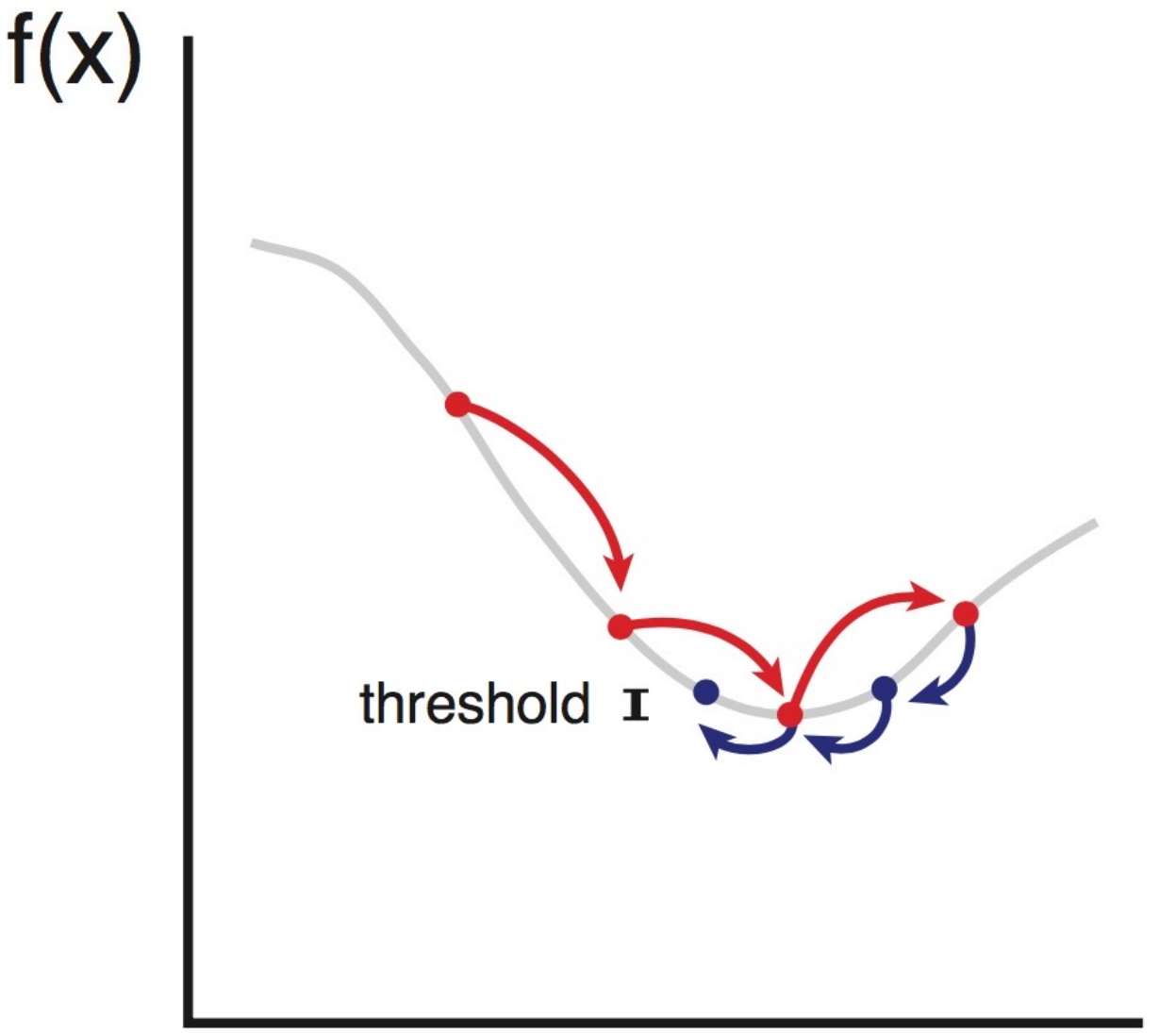



f(x)
dx
move x by an increment dx
X


f(x)
x
until reaching a point with
higher value of f(x) than previous


f(x)
x
then halve the increment and
move x backwards by dx/2


f(x)
keep moving and halving
x



until difference between f(x) at
two consecutive steps smaller than threshold

Minimisation : Packages

You would in general use an external minimiser package because (i) its not a new problem (ii) there is a lot of clever technology to use to optimise the process and make it fast, and you wont write this.

Examples are:

- Use is made of calculating the first derivatives (gradient at each point in a multidimensional space). Using this you can make a more intelligent guess at which way to go:

- Since a well behaved parameter has a parabolic “minimisation curve” (see next section), then it follows that its second derivative is a constant. In which case if you calculate this you can use it to make a projection of where the first derivative becomes zero (i.e. the minimum) and go straight there and then refine the guess .

- Errors are calculated in a way which estimates the correlations between parameters.

Minimisation : Packages

- In python use scipy.optimize
  - I find it hard as it does not return an error estimate easily

- Minuit is a very good and widely used minimiser. It pervades particle physics.
  - Comprehensive - returns errors and error matrices !
  - In C++ Minuit is native.
  - In python you can also use Minuit - its been wrapped in iMINUIT

- I recommend you try to use iMINUIT
  - More configurable
  - Will give you experience of particle physics

Exercise 1

Estimating parameters by minimising a χ^2

Minimisation of χ^2 to obtain a best fit straight line

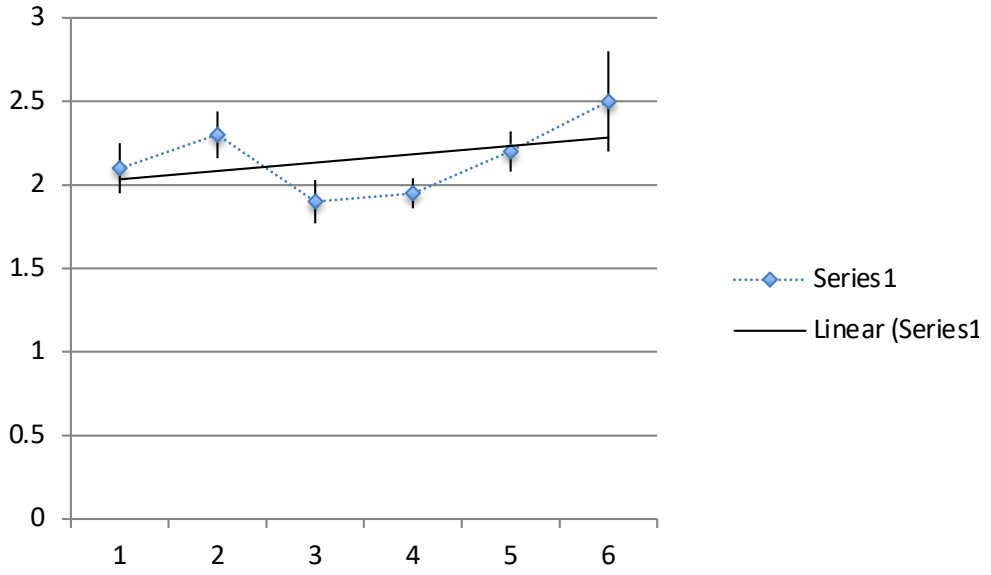
1. For each measured data point you form the quantity d^2:

$$d^2_i = [(y_i^{meas} - y_i^{pred}) / \varepsilon_i]^2$$

where

y_i^{meas} is the value of the measured data point "i"
ε_i is the error on the measured data point
y_i^{pred} is the prediction for that point from y=mx+c

Note that the "difference" is scaled by the error – which correctly accounts for the true significance of the difference: hence better than simple least squares



2. You then sum d^2_i over all data points, i, to form χ^2

$$\chi^2 = \Sigma_i \; d^2_i$$

3. You then vary "m" and "c" until χ^2 takes its **minimum** value

Where to get the Input Data

For data use:

x	y	err
1.0	1.05	0.03
2.0	0.93	0.04
3.0	0.99	0.02
4.0	1.06	0.02
5.0	0.94	0.01
6.0	1.08	0.03
7.0	1.01	0.02
8.0	0.89	0.04
9.0	0.95	0.025
10.0	1.03	0.03

These data are in a file:

`testData.txt`

another one in :

`testDataII.txt`

- Implement a minimiser of your own
- Form the χ^2 for the straight line hypothesis
- Run this to find the minimum value of χ^2 as you vary m and c
- This formally gives you the best estimates of the m and c parameters
- Run the same fit to get m and c using a module (scipy.optimise or Minuit)

Some things to think about

- How to minimise 2 parameters ? Vary one first, then the other ?
- But this is not good enough to just do once, you have to iterate this process until some threshold is reached
- Perhaps calculate derivate vector and go on that direction

Exercise 2

Estimating errors as per last slides

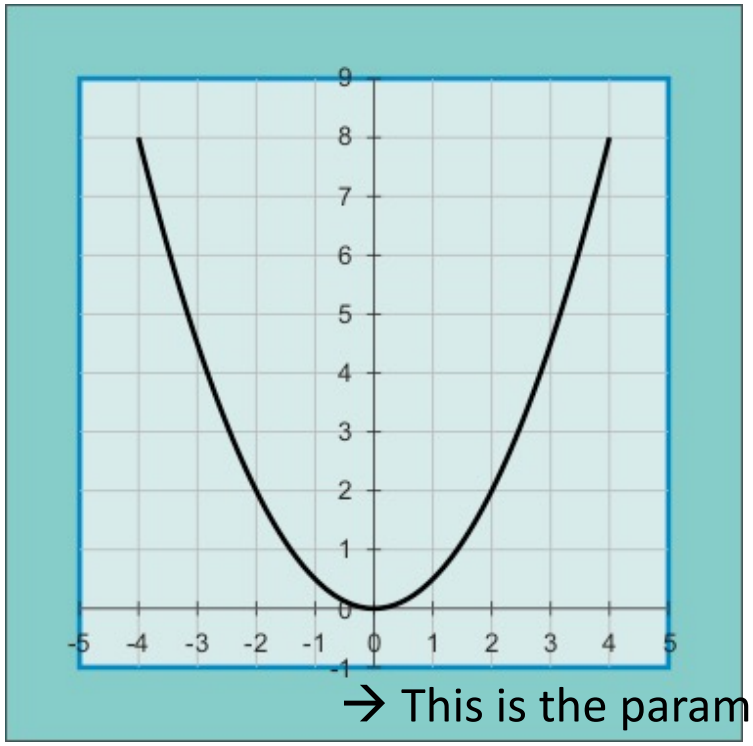
How to obtain the error on a measurement

- We have already covered one possible way: repeat the measurement many times then the standard deviation on the measured values of the parameter is its error, σ
- It is likely it will be difficult to repeat the experiment though. Often one only has one shot at it
- As an approximation one can run many so called "toy" measurements. This is where you simulate the entire measurement and repeat it many times. **This was exactly what you did in Checkpoint 1**
- However there is a much simpler way which mostly gives you the correct error .....

Plot the value of χ^2 around the minimum, here assumed to be at zero

➔ For a well behaved parameter, it is parabolic

This is the value of χ^2

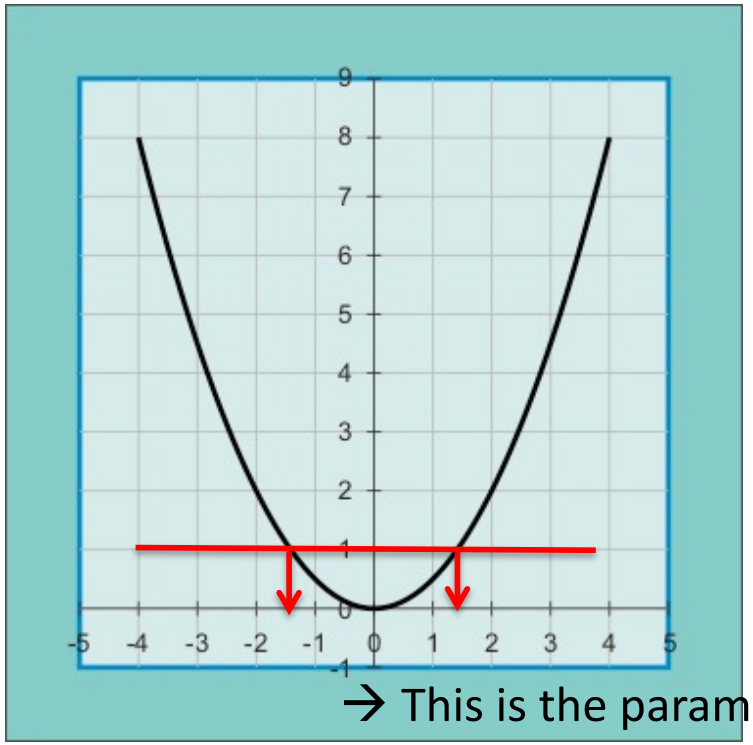


→ This is the parameter you vary

It is the case that if you go to either side of the minimum by one unit of the error on the parameter, then the χ^2 increases by a value of 1 unit

How to obtain the error on a measurement

This is the value of χ^2



→ This is the parameter you vary

The red line is drawn at +1 above the minimum on the y-axis

➔Draw a horizontal line at the +1 point

➔Read down to the x-axis and you will get the +- 1-standard deviation errors (should be the same) here this is about +- 1.5

You could easily implement this in the minimiser code you have written

This is the simple way in which MINUIT calculates its error

Random number generators & Numerical Integration

Summary of lecture & exercises:

- Writing a simple minimiser
- Minimisation of χ^2 as a way to find the best values for parameters
- Estimate errors on parameters from a minimiser using the χ^2 +-1 points
- Introduction to package minimisers: optimize and Minuit