85301 – Algorithms and Data Structures in Biology

Lab 2 – Measuring, Fitting, and Plotting

Enrico Malizia and Riccardo Treglia

DISI
University of Bologna, Italy

2nd semester, 2022/23

(parts of these slides are based on material by Prof. Ugo Dal Lago)

Measuring Program Performances at Different Input Sizes

- We have seen that a `Python` program's running time *on one specific input* can be measured effectively, e.g., via the `cProfile` module
- We have also seen that the time complexity of algorithms is usually given as a function on some of the characteristics of the algorithm input
- *Could the two approaches be reconciled?*
  - They are very different!
  - However, it would be very nice to turn our experimental data into a predictive machinery, taking advantage of the analytical knowledge about the underlying algorithm

Measuring Program Performances at Different Input Sizes

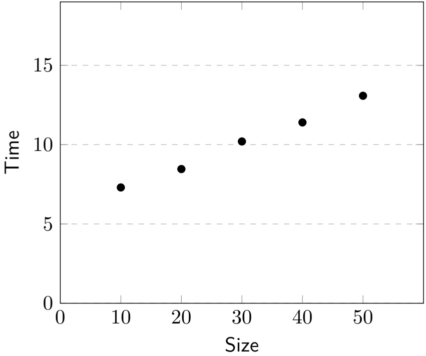
Analytical	Experimental
Asymptotic	Precise
Language and Machine Independent	Language and Machine Dependent
Worst-Case	Average-Case

Measuring Program Performances at Different Input Sizes

- Sometimes, *all runs* of a certain program (or algorithm) on inputs of a given size n take (approximately) the *same* time
  - In this case, measuring a program on few inputs of a given size is enough
  - The average-case behaviour somehow *coincides* with the worst-case behaviour
- Very often, however, the performance of a program on inputs of the same size can *vary*
  - In this case, measuring the program on many inputs of a given size, followed by averaging, is necessary
  - The average-case behaviour of the underlying algorithm can be quite different from the worst-case behaviour
- In all cases, inputs *of different sizes* should be analysed
  - This way, one can have an idea of how much the time required by the program grows with the size of the input

Data Fitting

- Suppose we have measured the performances of our program on inputs of sizes $10, 20, 30, 40, 50$, obtaining the results in the following graph:



- Moreover, suppose that the underlying algorithm has time complexity in $O(n)$, or even better in $\Theta(n)$
- How can we find a *linear function* $f : \mathbb{N} \mapsto \mathbb{N}$ fitting our data?

Data Fitting

- The **least-square method** is a suitable technique from statistics
- Given a set of points in $\mathbb{R}^2$, the technique aims at finding the curve of a certain shape (e.g., $a + bx$ or $a + bx + cx^2$, or $a + b2^x$) that *best* fits the points at hand
  - Technically, this is the curve minimizing the sum of the square residuals (i.e., of the square errors)
- A precise description of the least-square method is beyond the scope of this course
- The `Python` packages `numpy` and `scipy` provide some specific functions implementing this methodology

Data Fitting

```
import numpy
import scipy.optimize as optimization

xdata = numpy.array([10.0,20.0,30.0,40.0,50.0])
ydata = numpy.array([7.3,8.46,10.2,11.4,13.08])

def func(x, a, b):
    return a + b*x

print(optimization.curve_fit(func, xdata, ydata))
```

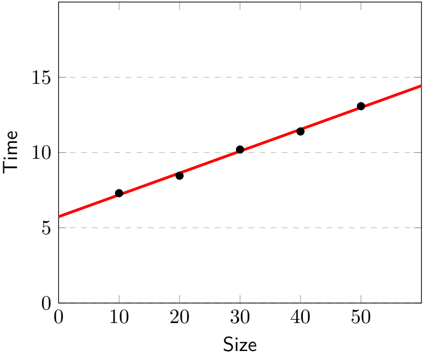
- We obtain the following:

```
(array([5.738, 0.145]),
 array([[ 3.09026665e-02, -8.42799994e-04],
        [-8.42799994e-04,  2.80933332e-05]]))
```

- The red part is the parameters estimation
- So, the linear function that best fits our data (according to the least-square method) is: $\underbrace{5.738}_{a} + \underbrace{0.145}_{b}\,x$

Data Fitting

- Once we have the estimation of the parameters for the fitting curve, we can plot the curve in LaTeX together with the data points



Plotting

- How could we *plot* data and functions in a graph, like we did in the previous slide?
- In LaTeX, there are plenty of different ways to do that, and the one we propose you to use is called `pgfplots`
  - Its syntax is quite intuitive
  - You can find plenty of resources online about how to render various kinds of graphs (see, e.g., (link))

Plotting

```
1 \begin{tikzpicture}[scale=0.7]
2   \begin{axis}[xlabel={Size},ylabel={Time},
3     xmin=0, xmax=60, ymin=0, ymax=20.000,
4     xtick={0,10,20,30,40,50}, ytick={0,5,10,15},
5     legend pos=north west,
6     ymajorgrids=true,
7     grid style=dashed]
8   \addplot[only marks, mark=*]
9     coordinates{(10,7.3)(20,8.46)(30,10.2)(40,11.4)(50,13.08)};
10  \addplot[color=red, line width=1.5]
11    coordinates{(0,5.738)(10,7.188)(20,8.638)(30,10.088)(40,11.538)
      (50,12.988)(60,14.438)};
12  \end{axis}
13 \end{tikzpicture}
```

- Instead of declaring the points of the function to plot (lines 10–11), we can provide the actual function, and the system will do the rest

```
  \addplot[domain=0:60, samples=100, color=red, line width=1.5]
    {5.738 + 0.145*x};
```