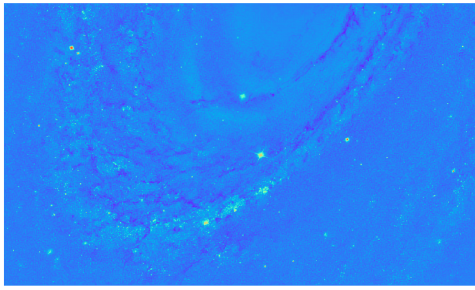
Lecture 7



- **Markets, Numerical Calculus, Zeros**

Scripts discussed in this lecture are downloadable from http://planets.utsc.utoronto.ca/~pawel/pyth. Please learn Python from them.

- Solutions to midterm problems
- Random walk through the markets
- Using real world data: Yahoo Finance historic quotes
- Data smoothing: convolution, boxcar kernels. Oxford weather
- Numerical Calculus: Differentiation formulae
  - applications to image processing and diffusion
- Zero finding of a real function: efficient methods
  - Bisection method
  - Secant method
  - Newton's method - what is 'quadratic convergence'?

Gamblers ruin in 1D walk with two absorbing boundaries at n=M & n=0 can easily be solved analytically:

Let P_n be the probability of a ruin starting from value n. No limit on number of steps
Boundary values are: $P_0 = 1$ and $P_M = 0$. (Why?)

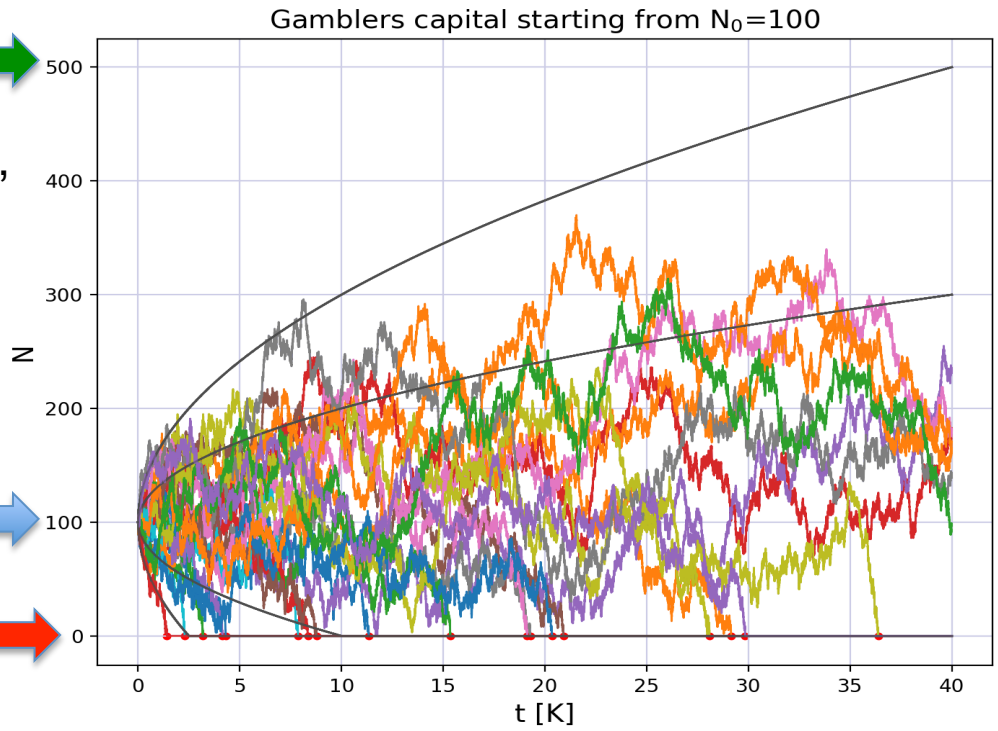
From value n we can either continue via n-1 with probability q, or via n+1 with probability 1-q. Therefore the probability of ruin can be expressed as:

$$\mathbf{P_n = q\, P_{n-1} + (1-q)\, P_{n+1}}$$

In case of q=1/2 (unbiased random walk in 1D with step +-1), we have

$\mathbf{P_{n-1} - 2\, P_n + P_{n+1} = 0}$ i.e., $\mathbf{P_{n-1} - P_n = P_n - P_{n+1}}$.

WIN at M



The slopes to the left and right are equal, therefore the function P(n) = P_n is a straight line joining the two boundary values 0 & 1,

$\mathbf{P_n = (M-n)/M.}$

In our case $P_{100} = 4/5$.

start at n

RUIN

See also the case M=+∞

rnd_wlk-ret-1.py

on our coding page.

Random Walks in Stock Market?

Full Randomness in Stock Market? The efficient market hypothesis says Yes. See N. Taleb's book "Fooled by Randomness".

Example: IBM stock price history, last two years:

- **Econometrics of markets:**

There is evidence that people (for instance Mutual Fund, retirement fund managers are not taking right decisions and perform *worse* than the stock index (70% funds lose money + they charge fees).

Is this true? You may analyze the data yourself with Python.

Data are freely downloadable from, e.g., Yahoo Finance site.

The so-called 'technical analysis' purports to be able to take good decisions by algorithmic trading.

Personal and corporate finance greatly benefit from computing.

1999: 90% of trades in markets by people, 10% by algorithms
2019: 10% ---,,--- --,,-- 90% ---,,----

One good book to read about computerized trading and its dangers is "Flash Boys", by Mikael Levis (there is also a Canadian movie loosely based on one chapter, but the book is very different).

Stock market: What can we do quantitatively with the data freely available from Yahoo Finance?

Date,Open,High,Low,Close,Adj Close,Volume [.csv format]

1980-09-18,0.492188,0.493490,0.492188,0.492188,0.312819,14227200

1980-09-19,0.496094,0.497396,0.496094,0.496094,0.315301,6086400

1980-09-22,0.490885,0.490885,0.488281,0.488281,0.310336,8313600

1980-09-23,0.471354,0.471354,0.470052,0.470052,0.298750,7142400

1980-09-24,0.467448,0.467448,0.464844,0.464844,0.295440,1002240

(thousands of lines here....)

2019-09-09,51.060001,52.029999,51.020000,51.590000,51.590000,20749700

2019-09-10,51.330002,51.840000,50.830002,51.820000,51.820000,18532000

2019-09-11,51.599998,52.790001,51.380001,52.790001,52.790001,18968900

2019-09-12,53.000000,53.330002,52.070000,53.009998,53.009998,23308700

2019-09-13,52.759998,53.000000,52.230000,52.540001,52.540001,18010800

2019-09-16,51.900002,52.290001,51.700001,52.200001,52.200001,13354600

2019-09-17,52.049999,52.139999,51.349998,51.950001,51.950001,19641100

Stock market: What can we do quantitatively with the data freely available from Yahoo Finance?

1. We can do simple histogram of daily increments of prices p(t), or increases of ln(p), i.e., log-ratios of prices, or logarithmic returns

$$X = d\ \ln(p) = \ln p(t+dt) - \ln p(t) = = \ln [p(t+dt)/p(t)],$$

in order to find out whether they resemble the symmetric, Gaussian normal distributions, which result from combination (addition) of very many small random disturbances.

2. We can take two price (increase) histories and correlate them

Definitions: We often write

$$\text{Variance} = \sigma^2 = < (X - <X>)^2 >$$

Numpy provides methods: X.mean(), X.std().

The second method does exactly what the above equation says: a mean of squared deviations from the mean. Check it!

But .std doesn't quite work as it should in statistics of data uncertainty

$$\text{Variance} = \sigma^2 = N/(N-1) < (X - <X>)^2 >$$

Iff N >> 1, then the two definitions are in agreement. For N=1, not really 😐

- What can we do quantitatively with the price data available from Yahoo Finance?

Variance = Var(x) = $\sigma_x^2 := \langle (x - \langle x \rangle)(x - \langle x \rangle) \rangle$

measures volatility of time series.

Covariance = Cov(x,y) = $\sigma_{xy}^2 := \langle (x - \langle x \rangle)(y - \langle y \rangle) \rangle$,

[where $\langle x \rangle := (1/N) \sum_n^N x_n$ is an arithmetic average; time aver.]

measures interdependence of two variables (financial instr.)

Coefficient of correlation = $r_{xy} := \sigma_{xy}^2 / \sigma_x \sigma_y = Cov/\sigma_x \sigma_y$

$r_{xy} = \sum_n (x_n - \langle x \rangle)(y_n - \langle y \rangle) / [\sum_n (x_n - \langle x \rangle)^2 \sum_n (x_n - \langle x \rangle)^2]^{1/2}$.

Q: Can you show that coeff. of correlation is $r = +-1$, if $y = ax+b$? (a,b=const.)

- Stock market: Price data on Yahoo Finance

There is a more specialized indicator called Beta, defined as:

$$\beta = \mathrm{Cov}(x,y) / \mathrm{Var}(y) = \sigma_{xy}^2 / \sigma_y^2 = r_{xy} \, \sigma_x / \sigma_y ,$$

where y is a financial instrument used as a benchmark, for instance risk-free government bond or market as a whole (market average).

Beta measures sensitivity of x to y, relative to the market's volatility.

Some of these definitions will be useful for an assignment problem in set #3

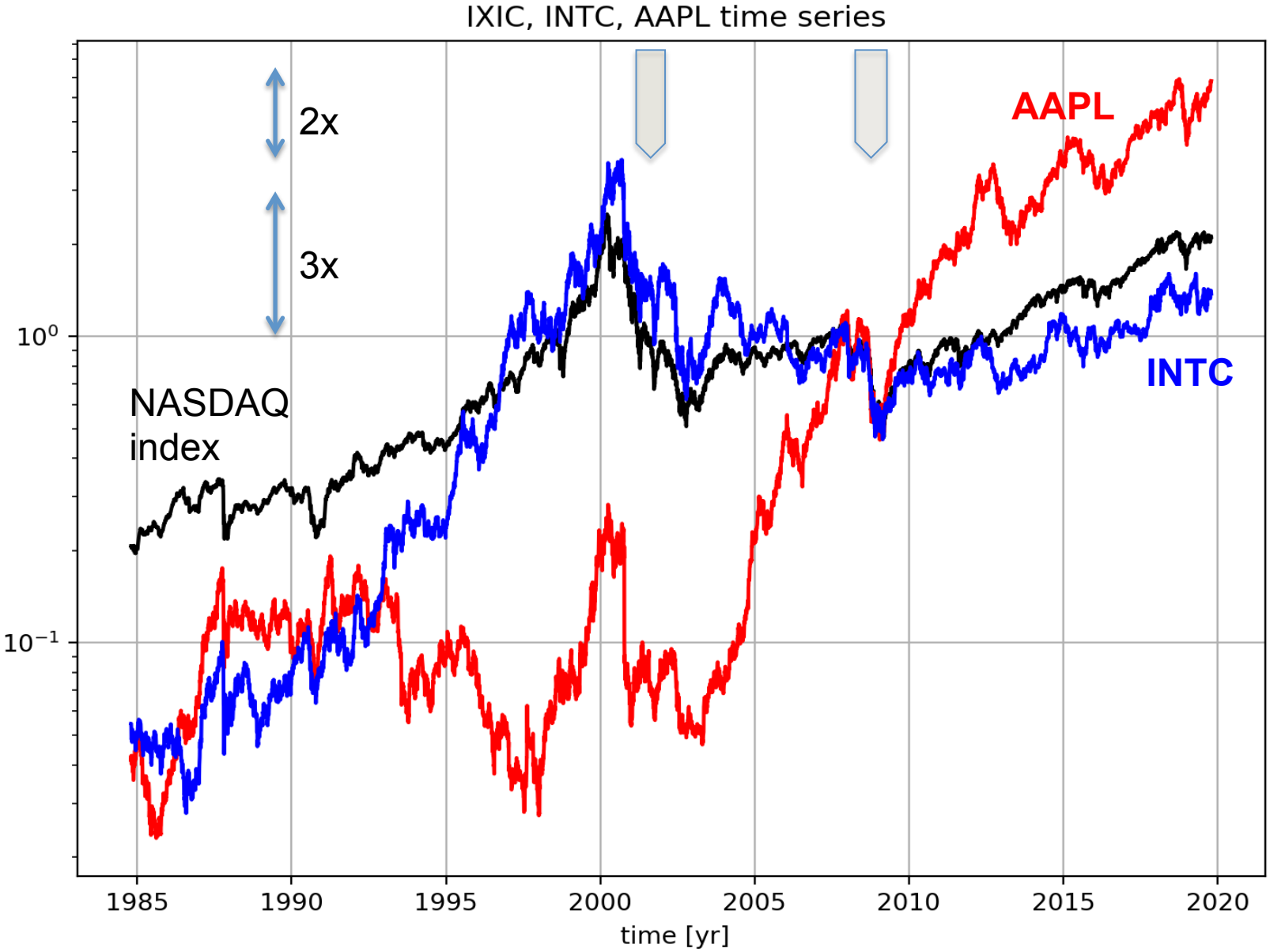
- Financial data processing from Yahoo Finance simple_fin-3.py

```
import numpy as np;  import pandas as pd   #  Pandas module, statistics
def  dataset(nr):
    if(nr==1):    name = "AAPL-84-19.csv"  # in our python code dir
    elif(nr==2):  name = "INTC-84-19.csv"  # ---,,---
    elif(nr==3):  name = "IXIC-84-19.csv"  # ---,,---

    x = pd.read_csv(name)      #  pandas can read CSV data from YF
    print('dataset: ', name, ' read ok. ', type(x))

    y = x.as_matrix()          # turn Pandas Data Frame into np.array
    Nd = np.size(y)//7
    print('Nd =',Nd)
    po = y[:,1];  pc = y[:,4]     # Open, Close
    ph = y[:,2];  pl = y[:,3]     # High, Low
    vol = y[:,6]                  # Volume
    return(Nd, y)       #  explicitly return var's to main prog.

# main program starts here:
NA, yA = dataset(1)            # read AAPL data
(...)
```

Financial data processing from Yahoo Finance

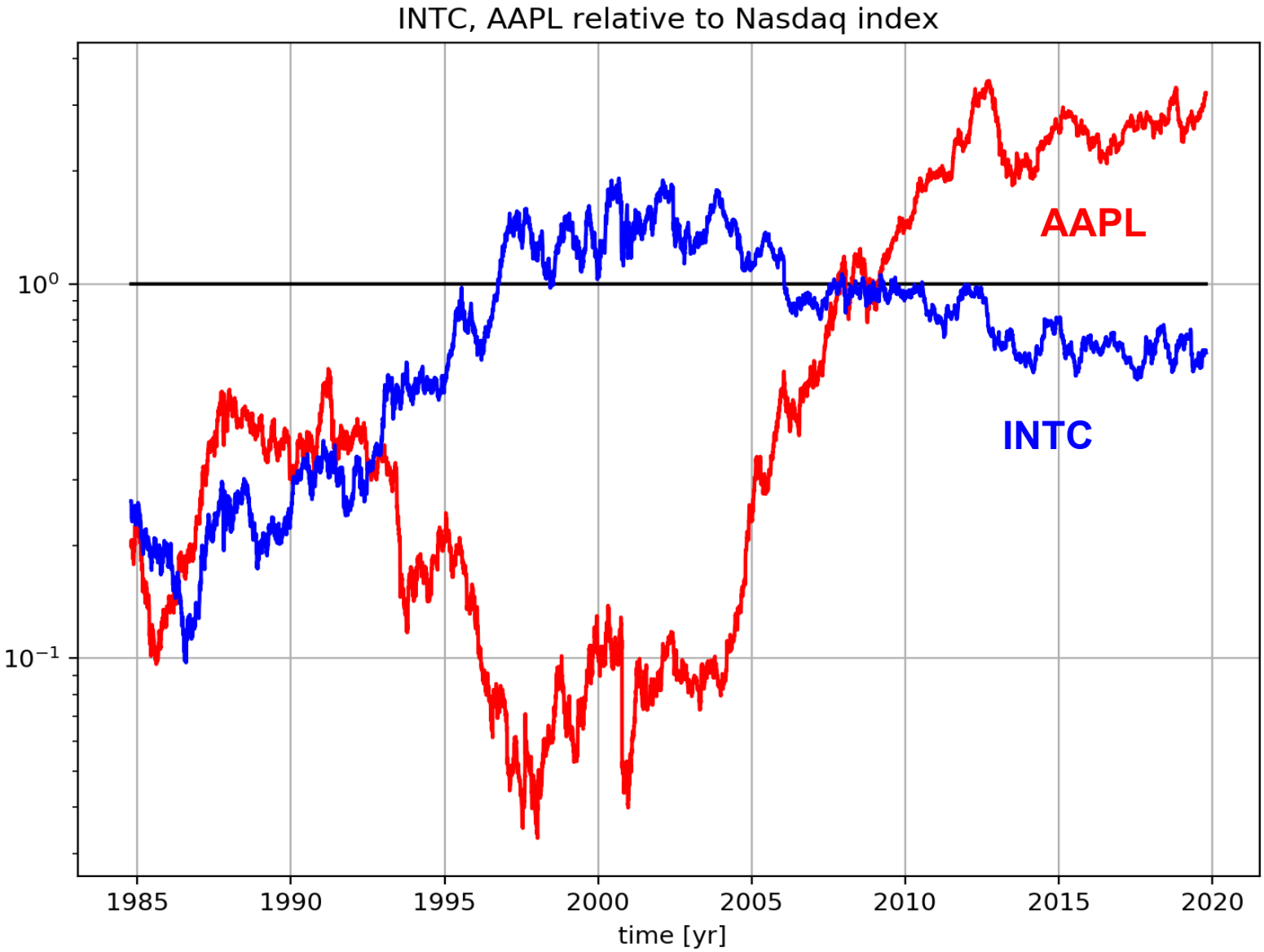
simple_fin-3.py

Notice the global crises of 2000 & 2008

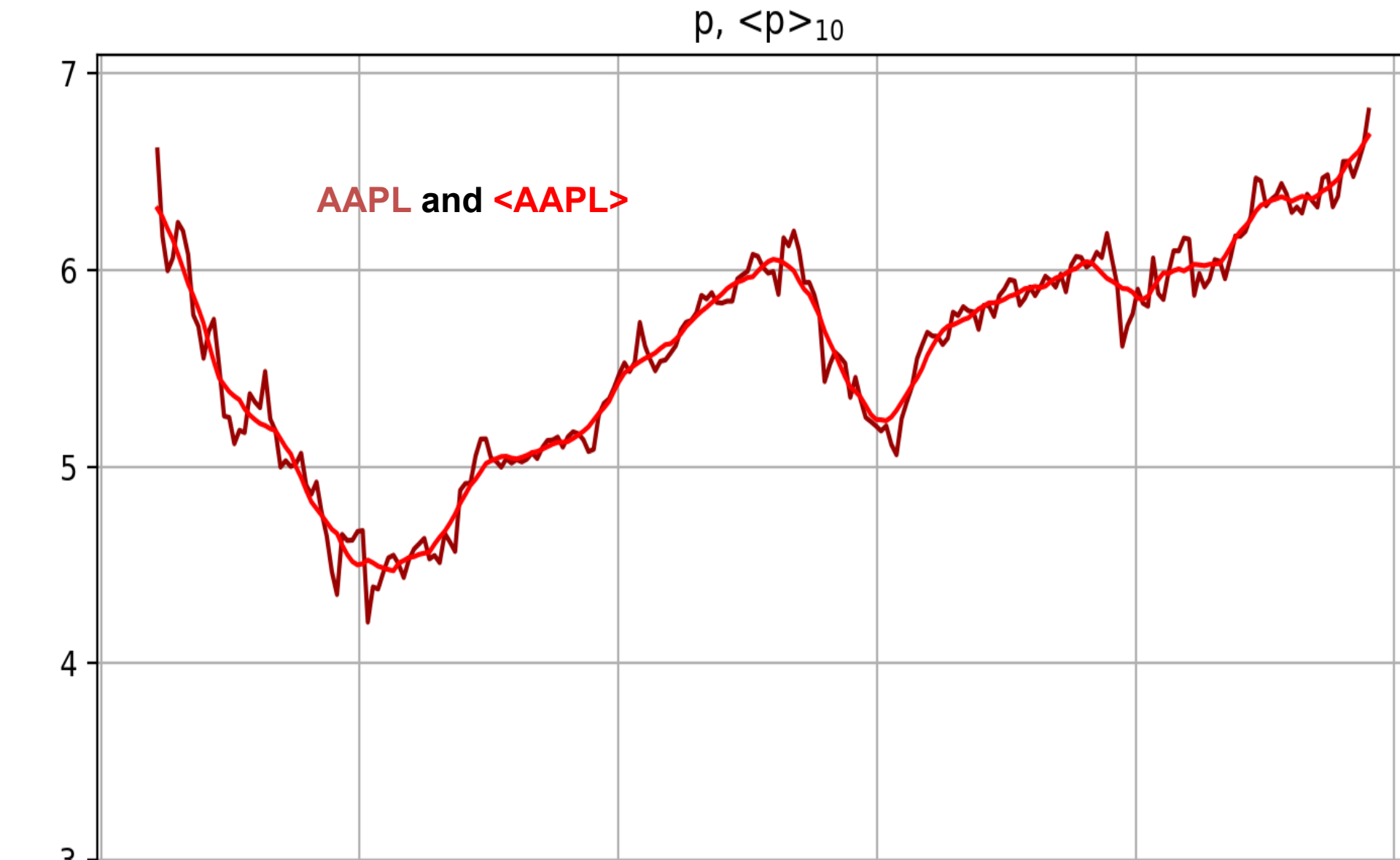


Apple, Intel pricelines divided by Nasdaq index

simple_fin-3.py show performance relative to market



Data smoothing by boxcar average, 11 days wide.



Data averaging by boxcar convolution

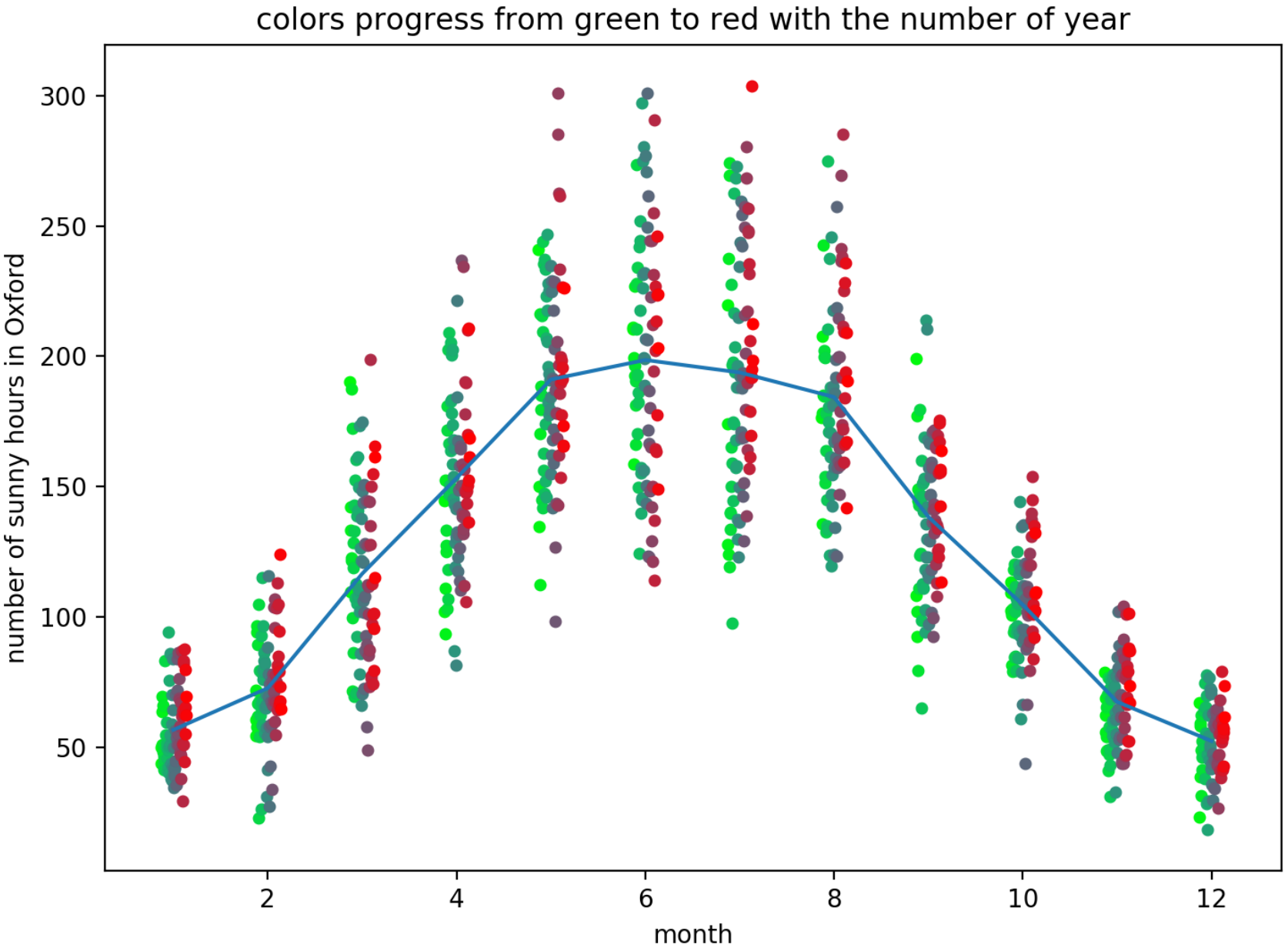
- **Sunny days in Oxford**

  oxford-IO-0.py oxford-IO-4.py

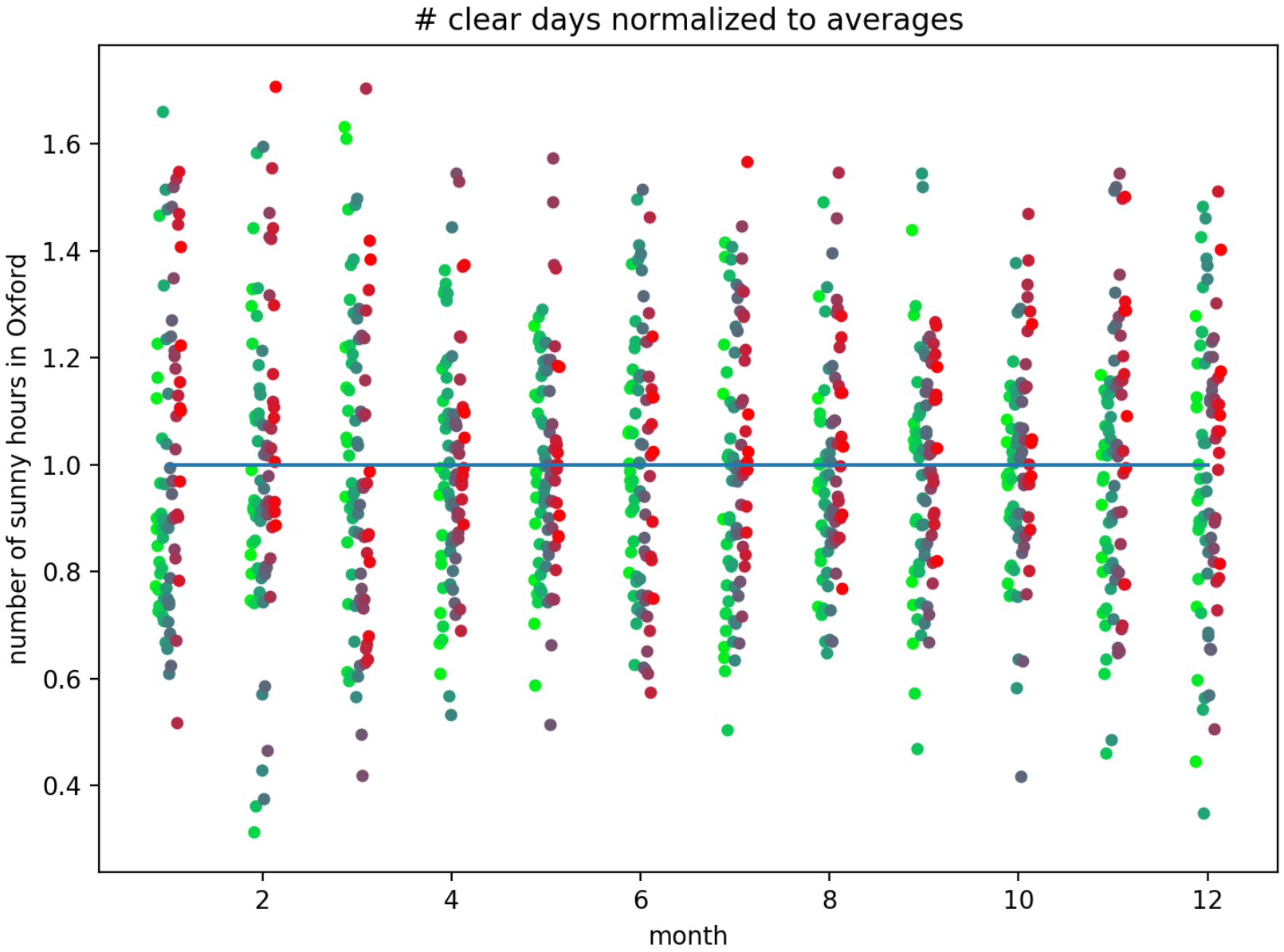
- Data on number of sunny hours/month in Oxford, UK, 1929-2010
- cf. H. P. Langtangen 'A primer on Scientific Progr. with Python', 3rd ed., p 78
- Meteorological archival data from British office
- http://tinyurl.com/pwyasaa/misc/Oxford_sun_hours.txt
- or in a simpler form of ASCII blank-separated columns
- in our code depository
- http://planets.utsc.utoronto.ca/~pawel/pyth/oxford_sunny.dat

Sunny days in Oxford

oxford-IO-0.py

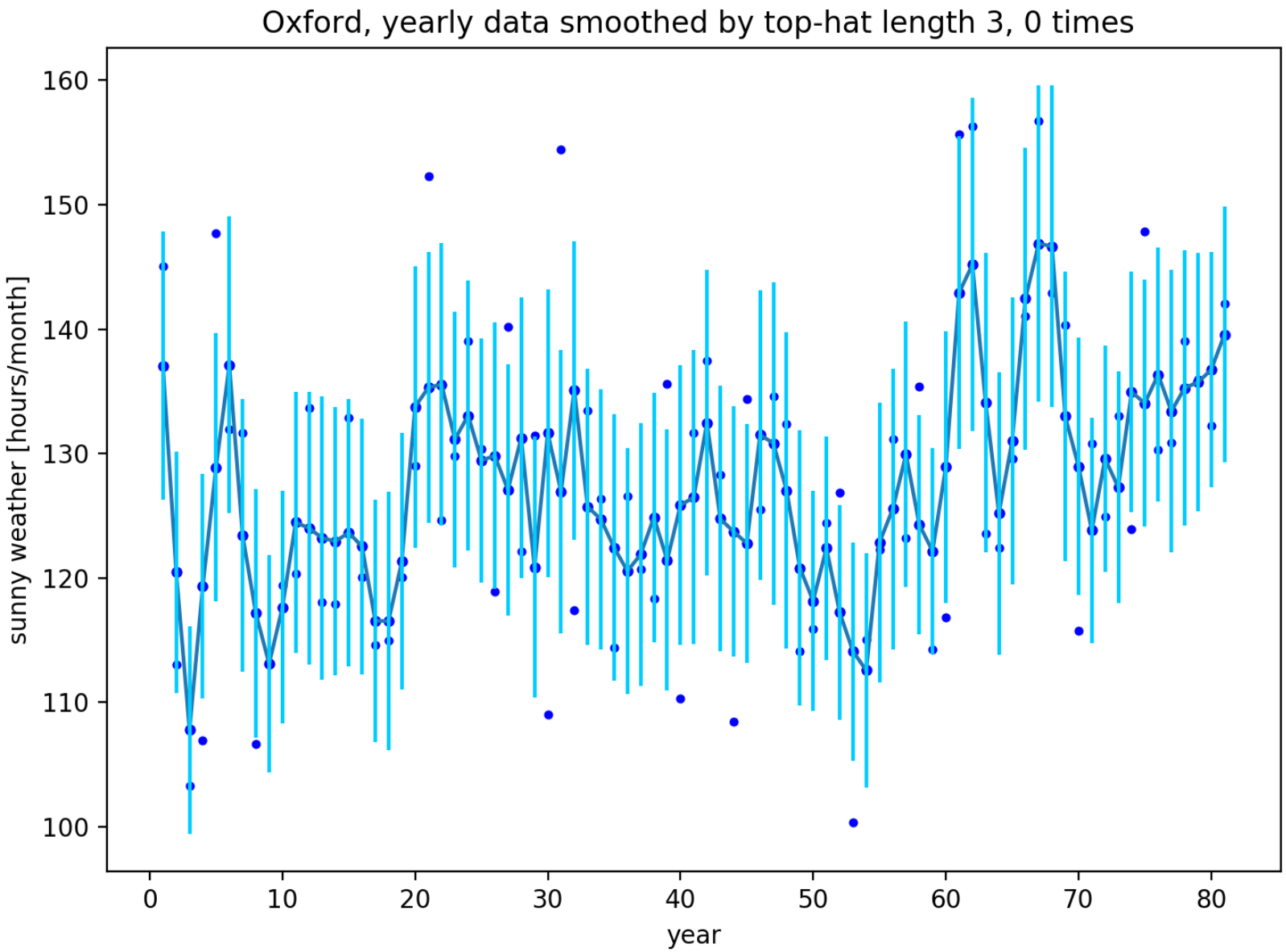


Sunny days in Oxford oxford-IO-0.py



Sunny days in Oxford

oxford-IO-4.py



- I/O and data analysis - averaging

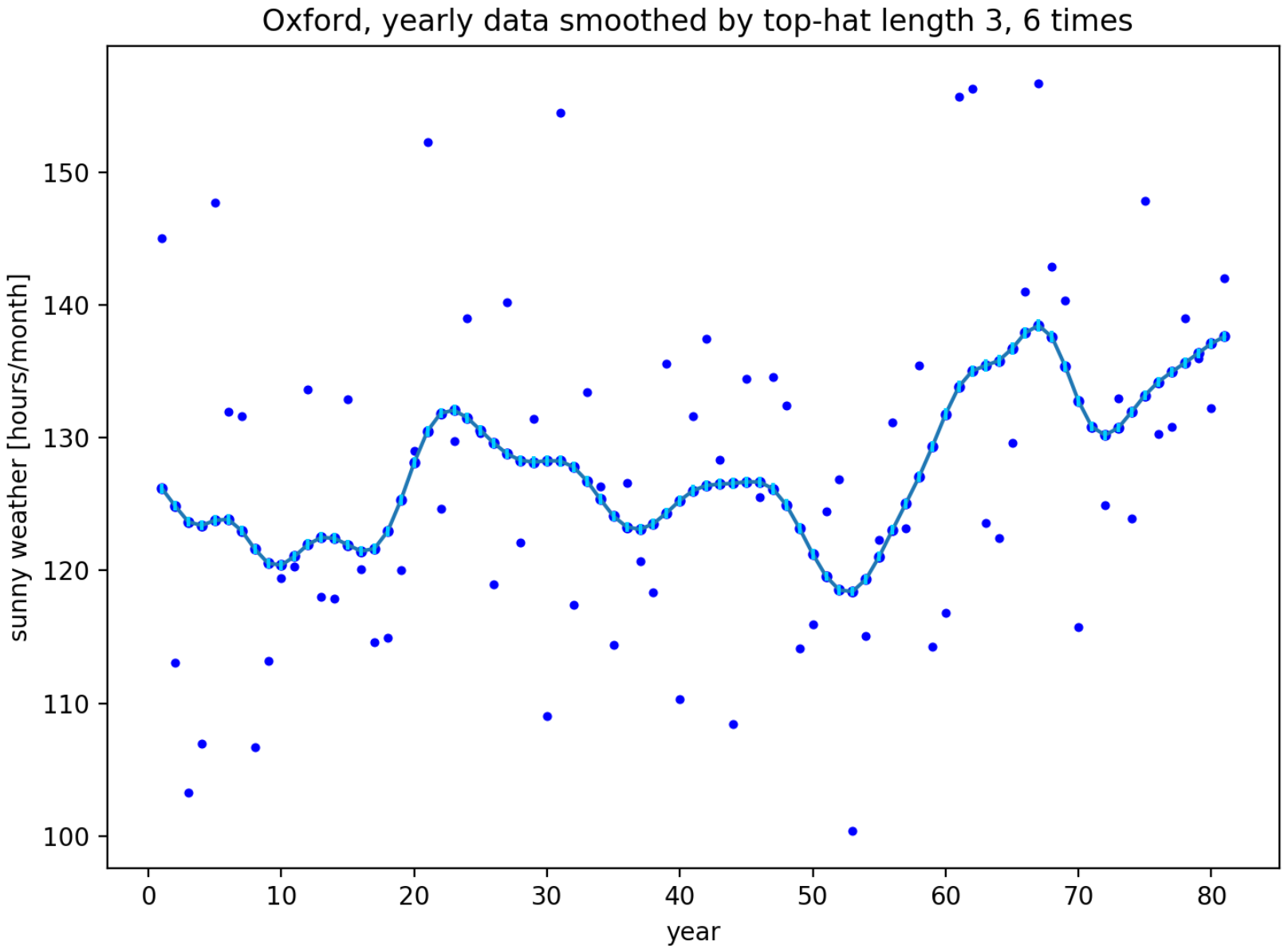
oxford-IO-0.py oxford-IO-4.py

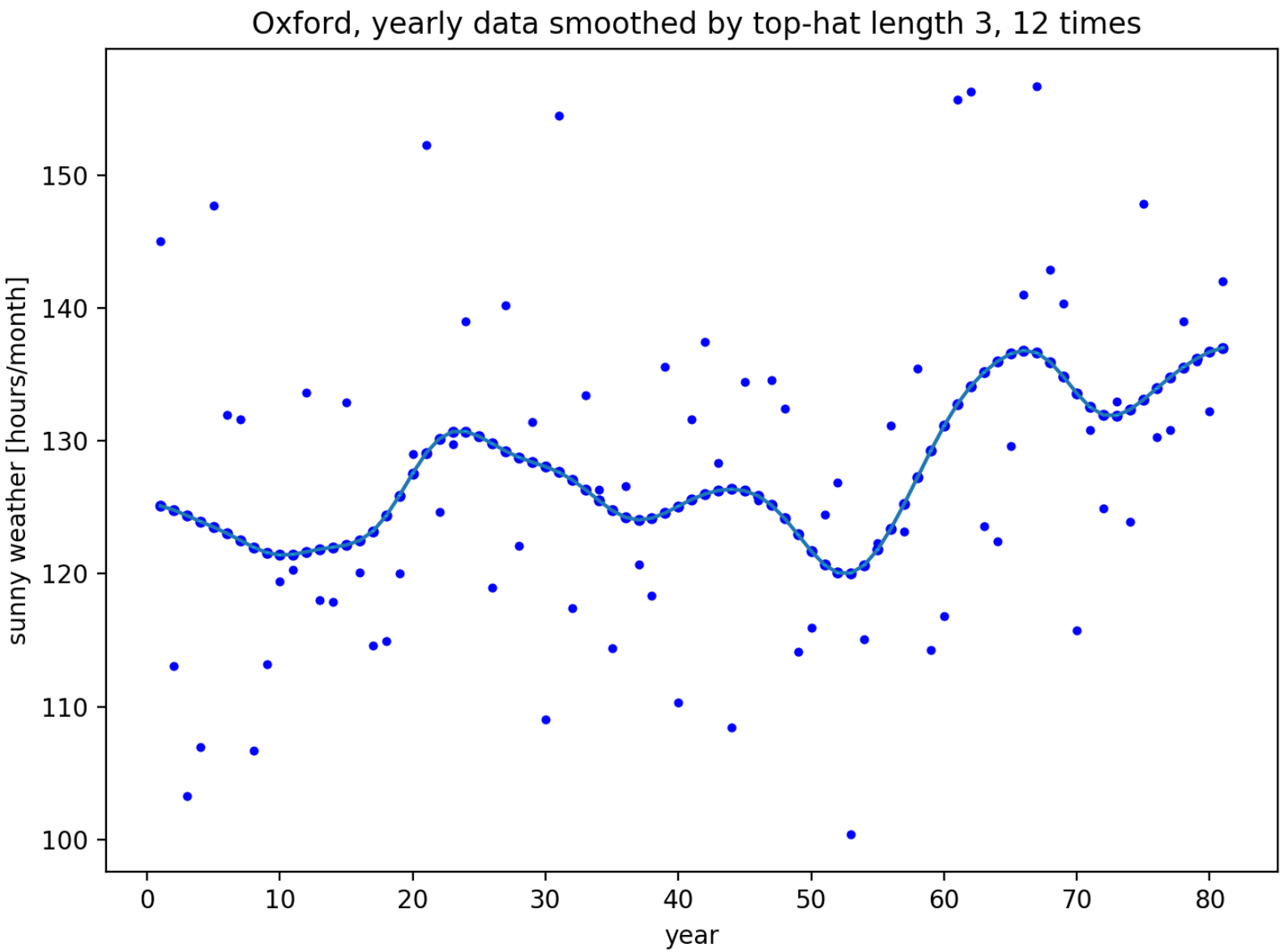
Boxcar average = convolution (smoothing)

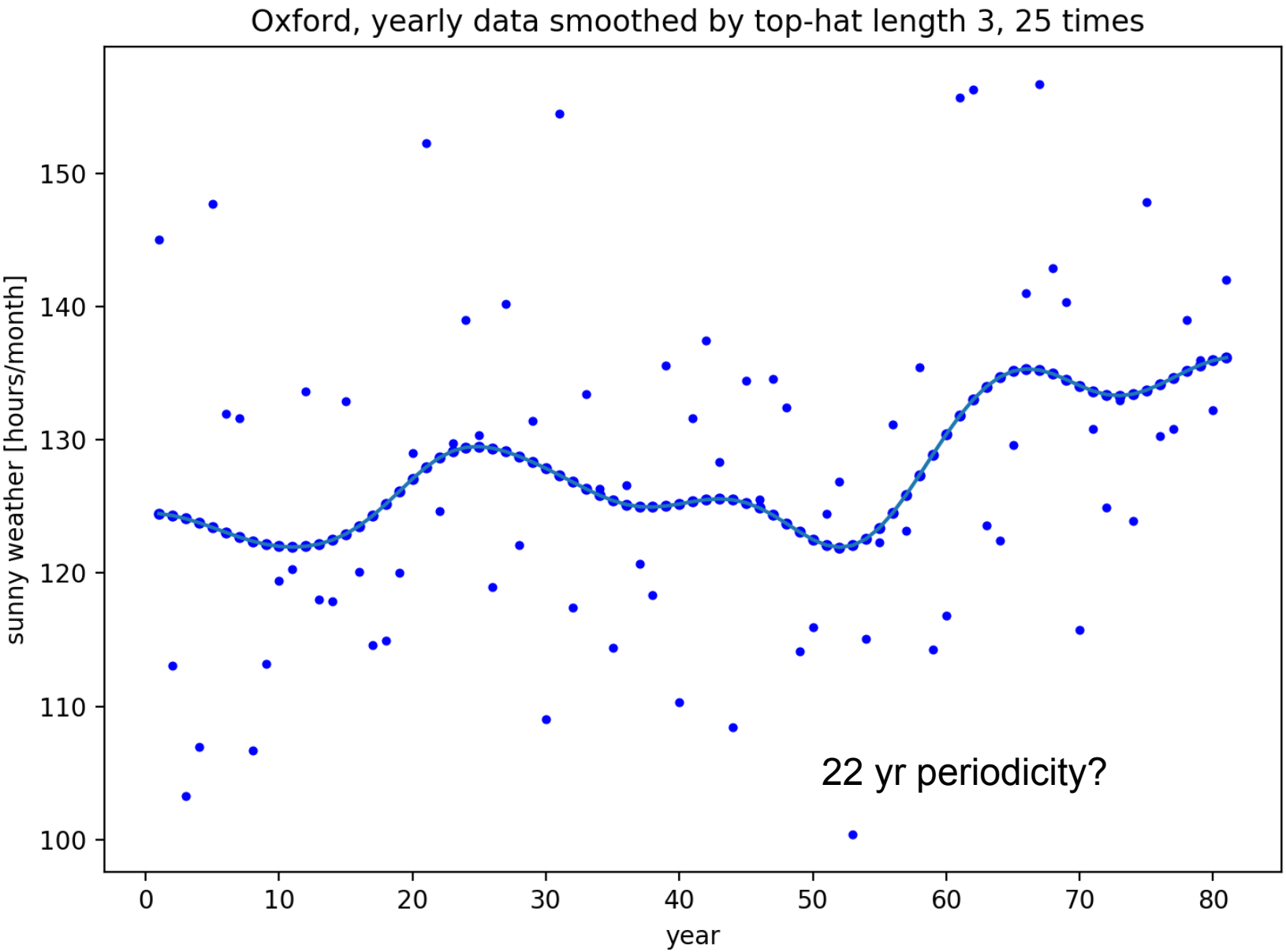
of data with top-hat or bell-shaped, symmetric kernel function

Boxcar averages all data above which it stands at the moment. It moves over all data set, producing its smoothed version.

- oxford-IO-4.py




Oxford, yearly data smoothed by top-hat length 3, 12 times
sunny weather [hours/month]
year
150
140
130
120
110
100
0
10
20
30
40
50
60
70
80


Oxford, yearly data smoothed by top-hat length 3, 25 times
sunny weather [hours/month]
year
100
110
120
130
140
150
0
10
20
30
40
50
60
70
80
22 yr periodicity?

Data smoothing by kernels:

- Weather in Oxford, England. Simple iterative smoothing.
- Convolution $$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$
- Convolution y = f * g is an integral operation,

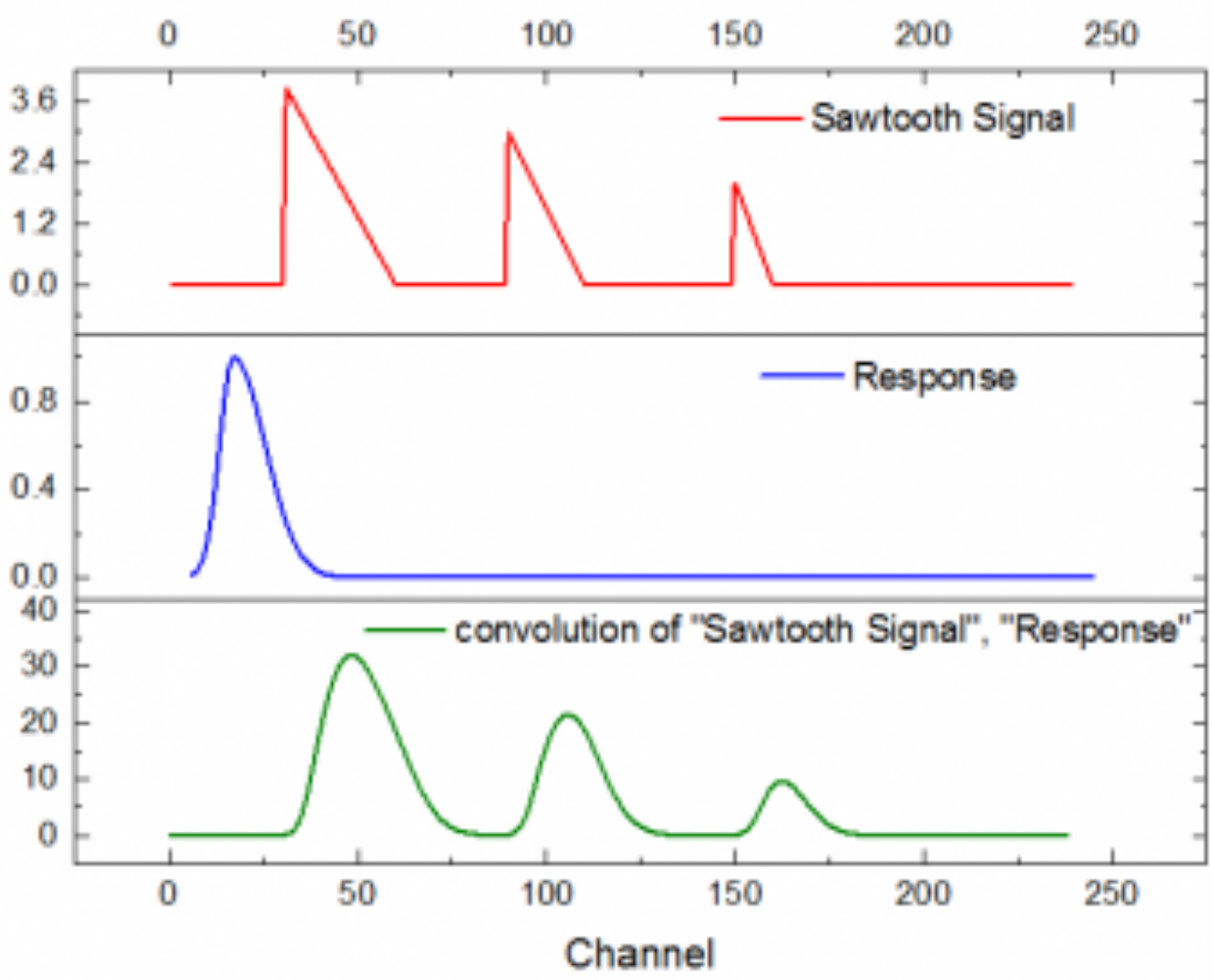
corresponding to shifted integration of a product of functions. Most often integration is over the whole domain on which functions are defined, e.g. –∞ to +∞. The integrand is one function (f) multiplied by a shifted & reversed copy of another (g). The amount of shift is the argument of the resultant function

f*g (t) = g*f (t) .

The meaning of convolution is smearing each point of one function with the pattern provided by the other function.

Data smoothing by convolution kernels:

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$



Data smoothing by convolution kernels:

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

- An example of convolution is the PDF (probability density function) of a sum of two random events: e.g., sum of points on 2 dice: Z = X + Y, or two coin throws ea. resulting in +-1.
- For instance, a blurry image through foggy eye glasses, or a telescope adding diffraction and internal scattering effects, is a convolution of a sharp image with the fuzzy PSF image (Point Spread Function, image of one single point on black background).
- Physics knows many systems governed by superposition principle. Convolution is how we compute the gravity or electrostatic potentials and forces from arbitrary distribution of matter.
- Another example is the image of a license plate of a moving car, smeared beyond recognition by the too long an exposure of the picture. Deconvolution allows restoration of readability.
- Yet another is a reverb effect added by the room on the played music. In this case, reverb may be similar to one-sided exponential function. Reverberation of a sharp peak of sound does not spread it into the past, only into the future.
- Convolution is an essential ingredient of pattern-recognizing NNs.

Numerical calculus: differentiation

see http://planets.utsc.utoronto.ca/~pawel/pyth

err_d1f.py

- Theory discussed in texbook#2
- Turner et al., Springer 2018
- Section 3.2, p. 38

Peter R. Turner · Thomas Arildsen
Kathleen Kavanagh

Applied
Scientific Computing

With Python

3 Numerical Calculus 35
3.1 Introduction 35
3.2 Numerical Differentiation 38
3.3 Numerical Integration 48
3.4 Composite Formulas 58
3.5 Practical Numerical Integration 68
3.6 Conclusions and Connections: Numerical Calculus 78
3.7 Python Functions for Numerical Calculus 79

4 Linear Equations 81
4.1 Introduction 81
4.2 Gauss Elimination 84

Numerical differentiation err_d1f.py

Take a function, any function (with enough non-zero derivatives)

- **def f(x): return((x-1)**3 -(x-1)**6/6)**

```
def  f1_exact(x):  return(3*(x-1)**2   -(x-1)**5)
def  f2_exact(x):  return(6*(x-1)   -5*(x-1)**4)
def  f3_exact(x):  return(6     -20*(x-1)**3)
def  f4_exact(x):  return(-60*(x-1)**2)
def  f5_exact(x):  return(-120*(x-1))
def  f6_exact(x):  return(-120)
```

and differentiate it once numerically (dy/dx)
with different displacements h (same as dx).

Plot error (w.r.t. exact derivative) as a function of h.

Numerical differentiation err_d1f.py

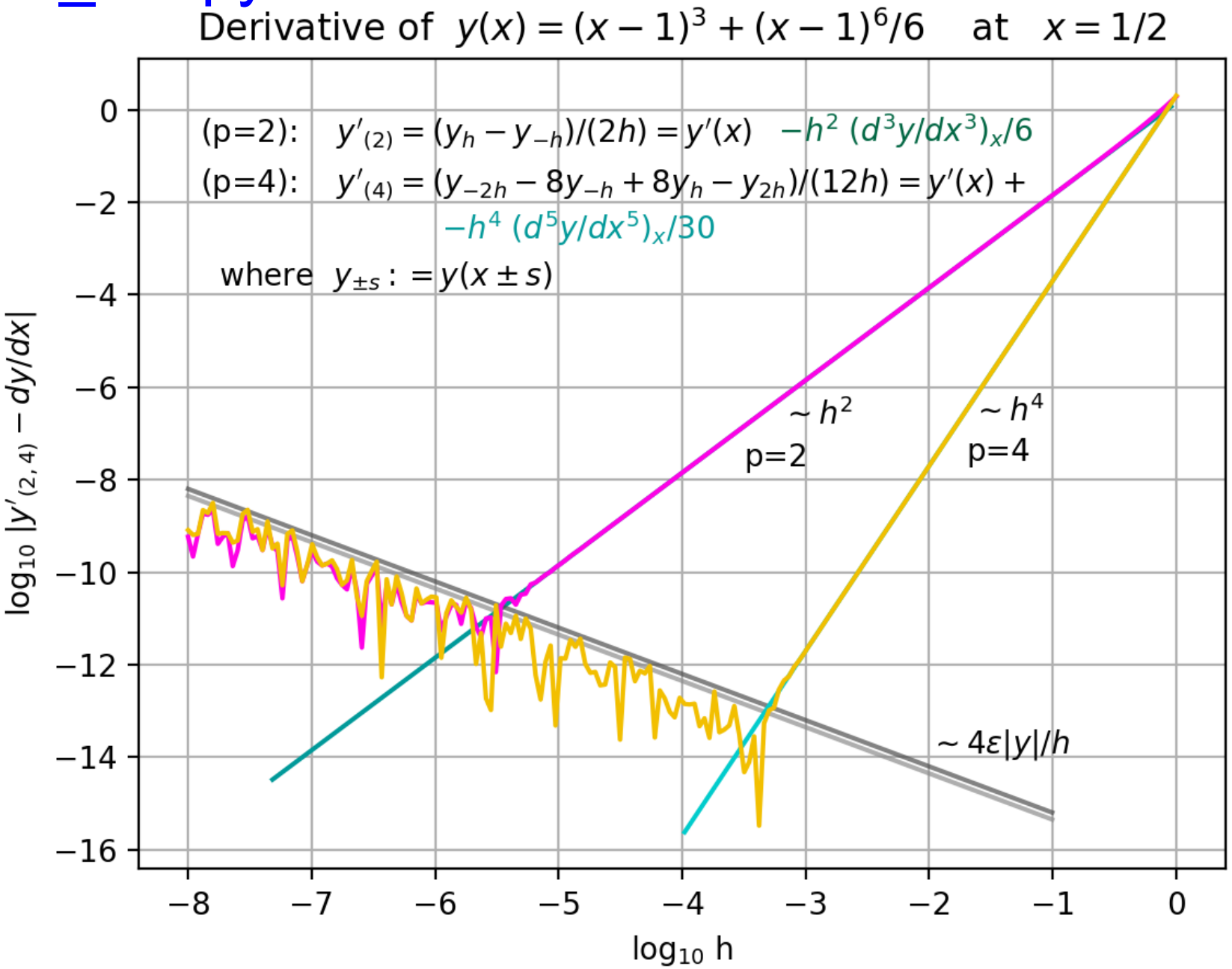
Take a function, any function (with enough non-zero derivatives)

```
def  f(x):
        return((x-1)**3 -(x-1)**6/6)
# main program
# different differentiation schemes
logh = -np.linspace(8,0,200)    # log h = -8...0
h = 10.**logh
x = 0.5
# numerical differentiation
# p = 2 diff. formula
d1f_2 = (f(x+h)-f(x-h))/(2*h)
# p = 4 diff. formula
d1f_4 = (8*(f(x+h)-f(x-h))+f(x-2*h)-f(x+2*h))/
(12*h)
```

Numerical differentiation err_d1f.py

```
# theoretical error bounds: roundoff and truncation errors
for p in [2,4]:
    E_round = np.array(  (5*p)**0.5/2* eps * abs(f(x))/h  )
    logE = np.log10(E_round)
    c = 0.8-p/8
    plt.plot(logh,logE,color=(c,c,c),alpha=0.7)
    if(p==2):
        E_trunc = abs(f3_exact(x))/6*h**2
#                 |d^3f/dx^3(x)|/6   h^2
        plt.plot(logh,np.log10(E_trunc),color=(0,.6,.6))
    elif(p==4):
        E_trunc=abs(f5_exact(x))/30*h**4
#                 |d^5f/dx^5(x)|/30  h^4
        plt.plot(logh,np.log10(E_trunc),color=(0,.8,.8))

# plotting of numerical error
plt.plot(logh,np.log10(abs(d1f_2-f1_exact(x))),
       color=(1,0,.9)) # magenta
plt.plot(logh,np.log10(abs(d1f_4-f1_exact(x))),
       color=(.95,.75,0)) # gold
```

err_d1f.py


Derivative of $y(x) = (x-1)^3 + (x-1)^6/6$ at $x = 1/2$
(p=2): $y'_{(2)} = (y_h - y_{-h})/(2h) = y'(x) - h^2\,(d^3y/dx^3)_x/6$
(p=4): $y'_{(4)} = (y_{-2h} - 8y_{-h} + 8y_h - y_{2h})/(12h) = y'(x) + -h^4\,(d^5y/dx^5)_x/30$
where $y_{\pm s} := y(x \pm s)$
$\sim h^2$
p=2
$\sim h^4$
p=4
$\sim 4\varepsilon|y|/h$
$\log_{10}|y'_{(2,4)} - dy/dx|$
$\log_{10} h$
0
−2
−4
−6
−8
−10
−12
−14
−16
−8
−7
−6
−5
−4
−3
−2
−1
0

Read textbook 2 carefully, there is an assignment that asks you to analytically repeat the derivation of error term in order p=2,4 **second differentiation.** Second derivative is the first derivative of the first derivative, so one stencil is this:

$$y'' \sim [\ (y_{n+1} - y_n)/h + (y_{n+1} - y_n)/h\ ]\ /h\ =\ (y_{n-1} - 2\,y_n + y_{n+1})/h^2$$

Since we know that symmetric first derivative stencil has error $\sim h^2$, applying it twice (forming difference of two symmetrically displaced first derivatives) we cannot lower the accuracy to ~h: the error will be some combination of two truncation errors $\sim h^2$, and thus will be of the second order ($\sim h^2$).

It's up to you to write Taylor expansions and do the algebra, to derive the full error term with correct derivatives and coefficients.

Then, using program err_d1f.py as a template, you will change it to study the numerical errors (truncation and roundoff) of two second derivative stencils,

$$y'' \sim\ \ (y_{n-1} - 2\,y_n + y_{n+1})/h^2$$

and

$$y'' \sim (-y_{n-2} + 16\,y_{n-1} - 30\,y_n + 16\,y_{n+1} - y_{n+2})/12h^2$$

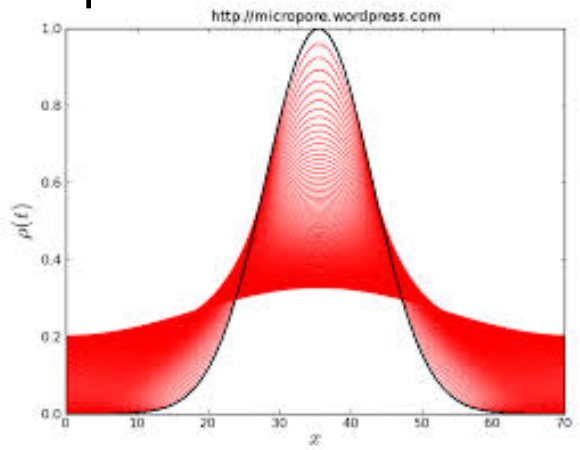
[cf. problem set #3]

Application of numerical calculus:

Differentiation formulae provide ways to compute n-dimensional stencils for 1st and 2nd derivatives. They include Laplacian operators such as

$(d^2/dx^2 + d^2/dy^2 + d^2/dz^2)\, f(x,y,z)$

$(d^2/dx^2 + d^2/dy^2)\, f(x,y)$



This enables us to find curvature of functions that change in time.
In fact, time evolution of thing that diffuse, such as thermal energy (=temperature T), or concentration of fragrance in a room, or molecules in a container, is governed by diffusion equation, which says that:

$$\frac{\partial C(x,t)}{\partial t} = D\frac{\partial^2 C(x,t)}{\partial x^2},$$

or

The rate of change is equal to the divergence of the product of the diffusivity and the gradient

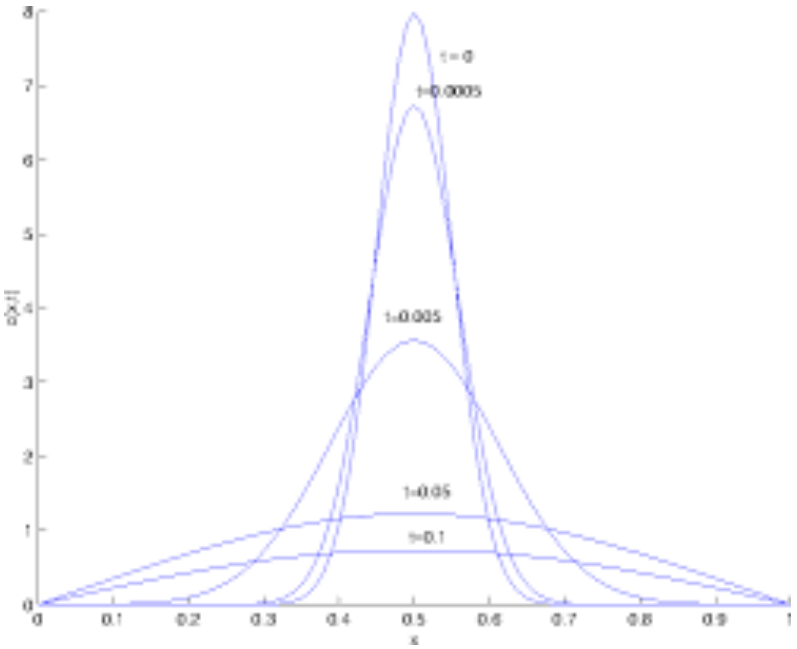
$$\frac{\partial u}{\partial t} = \nabla \cdot (D \nabla u)$$

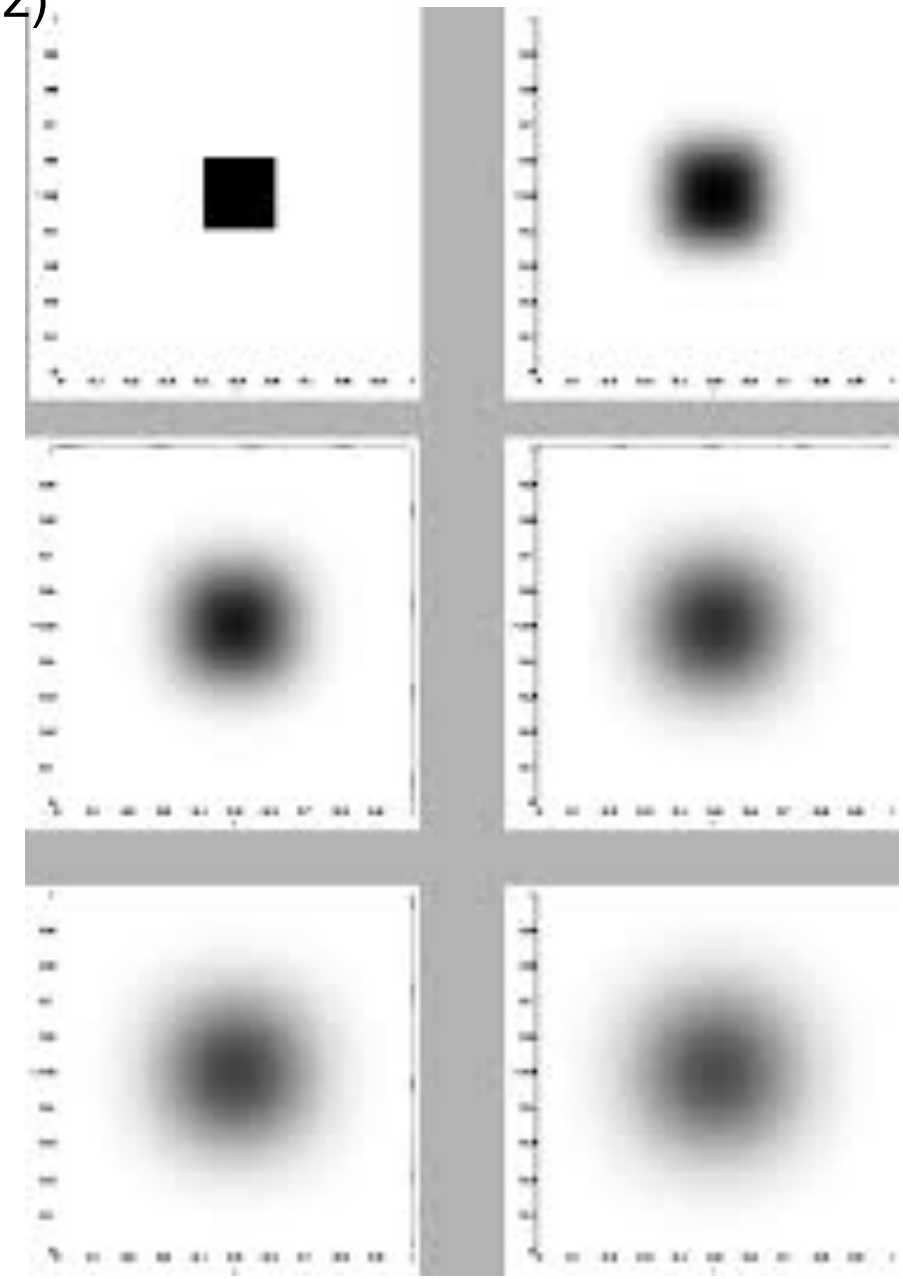
Application of numerical calculus: Diffusion equations

$df/dt = D\,(d^2/dx^2 + d^2/dy^2 + d^2/dz^2)\,f(x,y,z)$

$df/dt = D\,(d^2/dx^2 + d^2/dy^2)\,f(x,y)$

$$\frac{\partial C}{\partial t} = D\nabla^2 C,$$





Application of numerical calculus: Image processing, blurring images

laplacian-4.py in our code repository

Used the basic second derivative stencil to do
$(d^2/dx^2 + d^2/dy^2)\ F(x,y)$ on an image $F(x,y)$.

d^2/dx^2

+1 -2 +1

+

+1

-2

+1

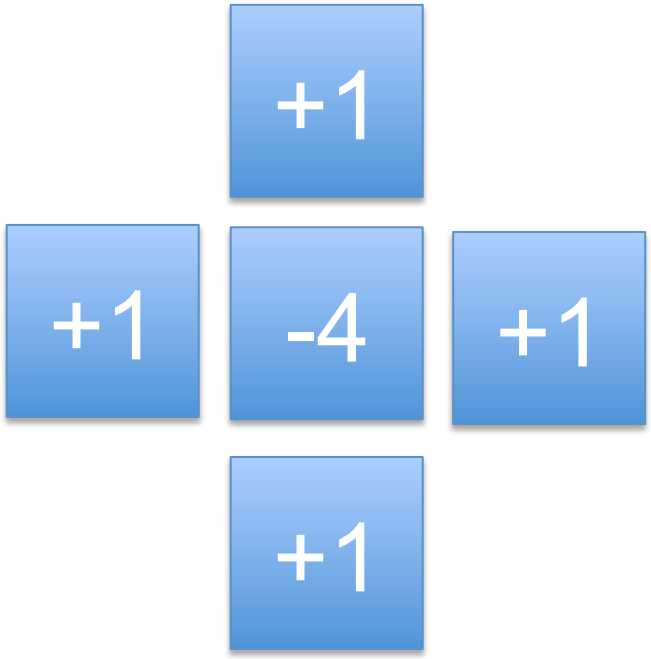
$+ d^2/dy^2$

Application of numerical calculus: Image processing, blurring images

laplacian-4.py

Used the basic second derivative stencil to do
$(d^2/dx^2 + d^2/dy^2)\ F(x,y)$ on an image $F(x,y)$.

=



Stencil for Laplacian operator in 2D

$$F(x,y,z,\ t+dt) = F(x,y,z,t) + dt\ Dh^{-2}\ [F(x+dx) + F(x-dx) + F(y+dy) + F(y-dy) - 4\ F(x,y,z,t)]$$

Application of numerical calculus: Image processing, blurring images

$dF/dt = (d^2/dx^2 + d^2/dy^2)\, F(x,y)$ on an image $F(x,y)$.
Diffusion Equation discretized as

$$F(x,y,z,\, t+dt) = F(x,y,z,t) + dt\, Dh^{-2} [F(x+dx) + F(x-dx) + F(y+dy) + F(y-dy) - 4\, F(x,y,z,t)]$$

Let $q := dt\, Dh^{-2}$

$$F(x,y,z,\, t+dt) = (1-4q)\, F(x,y,z,t) + q\, [F(x+dx) + F(x-dx) + F(y+dy) + F(y-dy)]$$

For stability of calculation, $q < 0.25$, otherwise negative $F(x,y,z,t+dt)$ can appear, which for temprature of concentration of particlaes or images is non-physical.

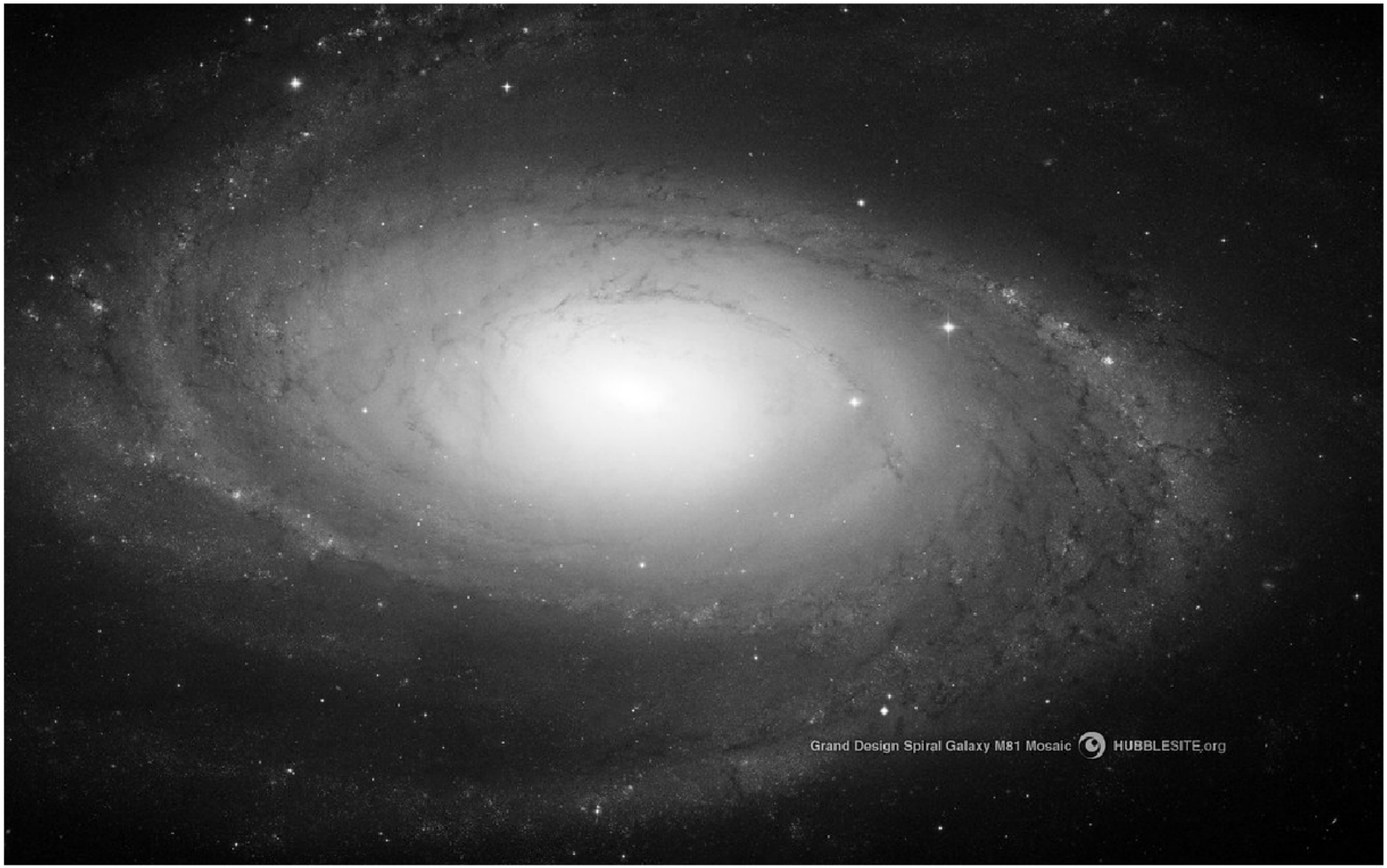
Maximum *reasonable* q is however 1/5 not ¼, and results in spreading of one single peak (1 pixel) to neighboring 5 pixels, each getting value 1/5:

$1 - 4q = 1 - 4/5 = 1/5 = q$

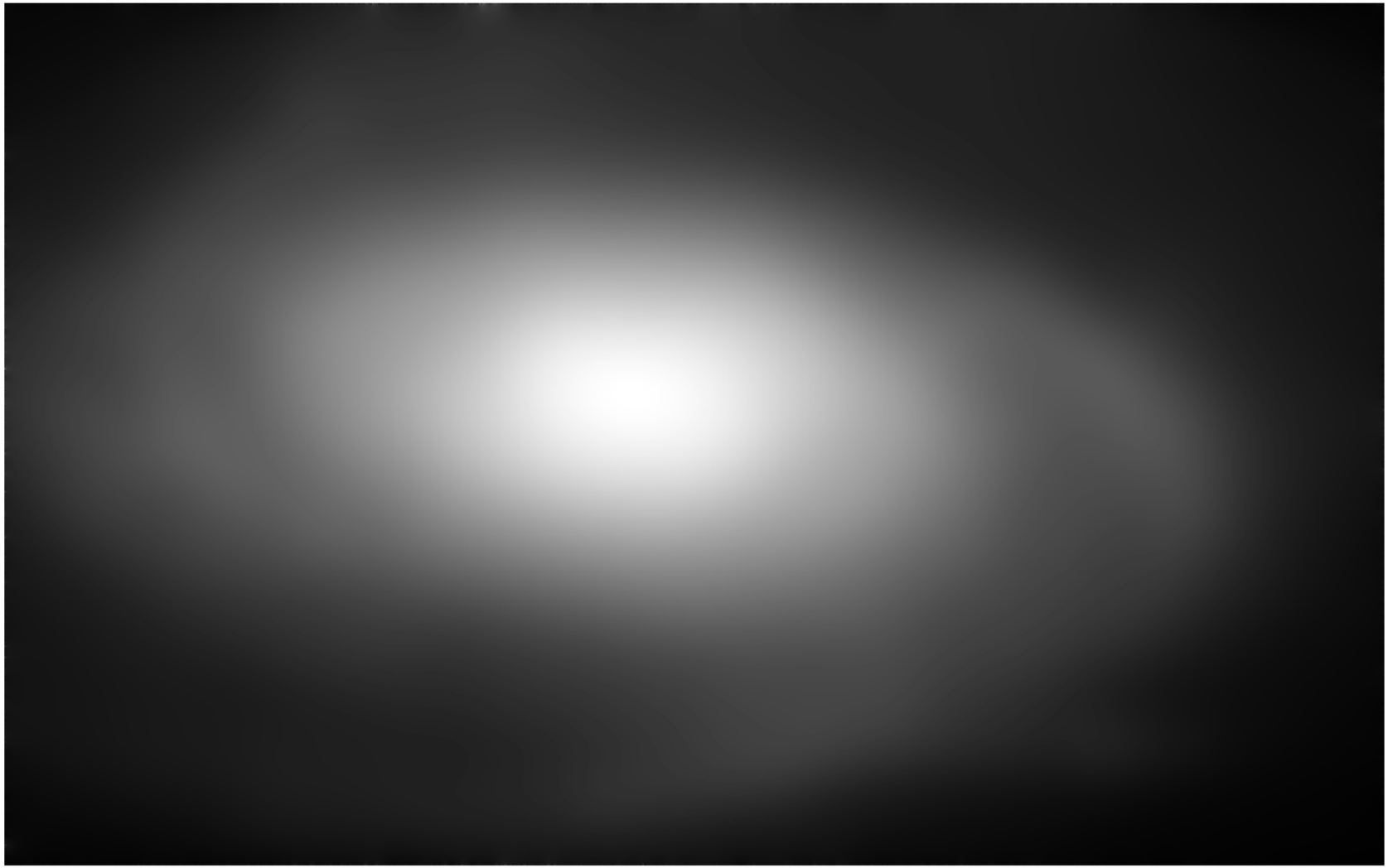
Let's apply such a blurring procedure to astronomical image of grand-design spiral galaxy M81.

laplacian-4.py

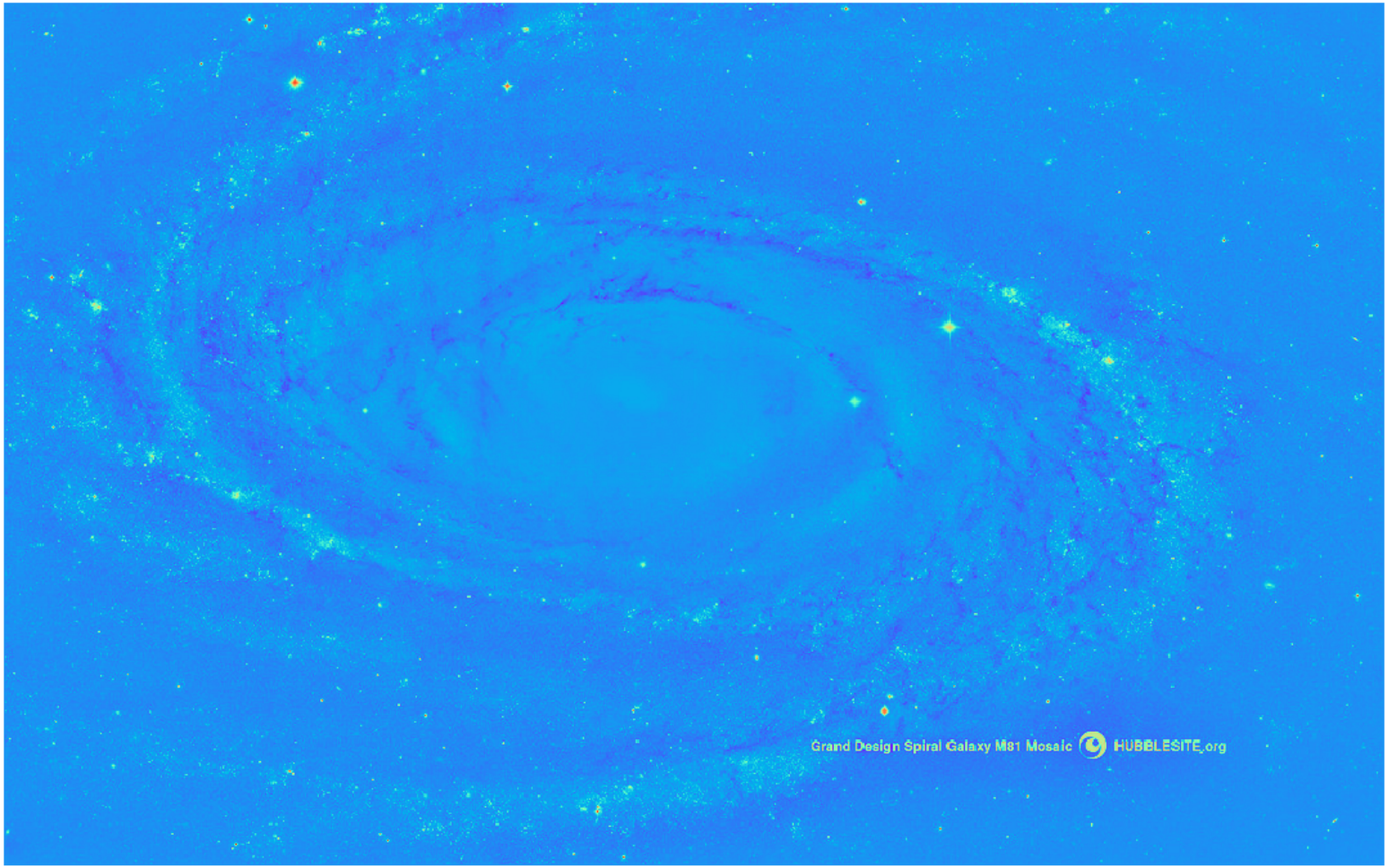
M81 galaxy

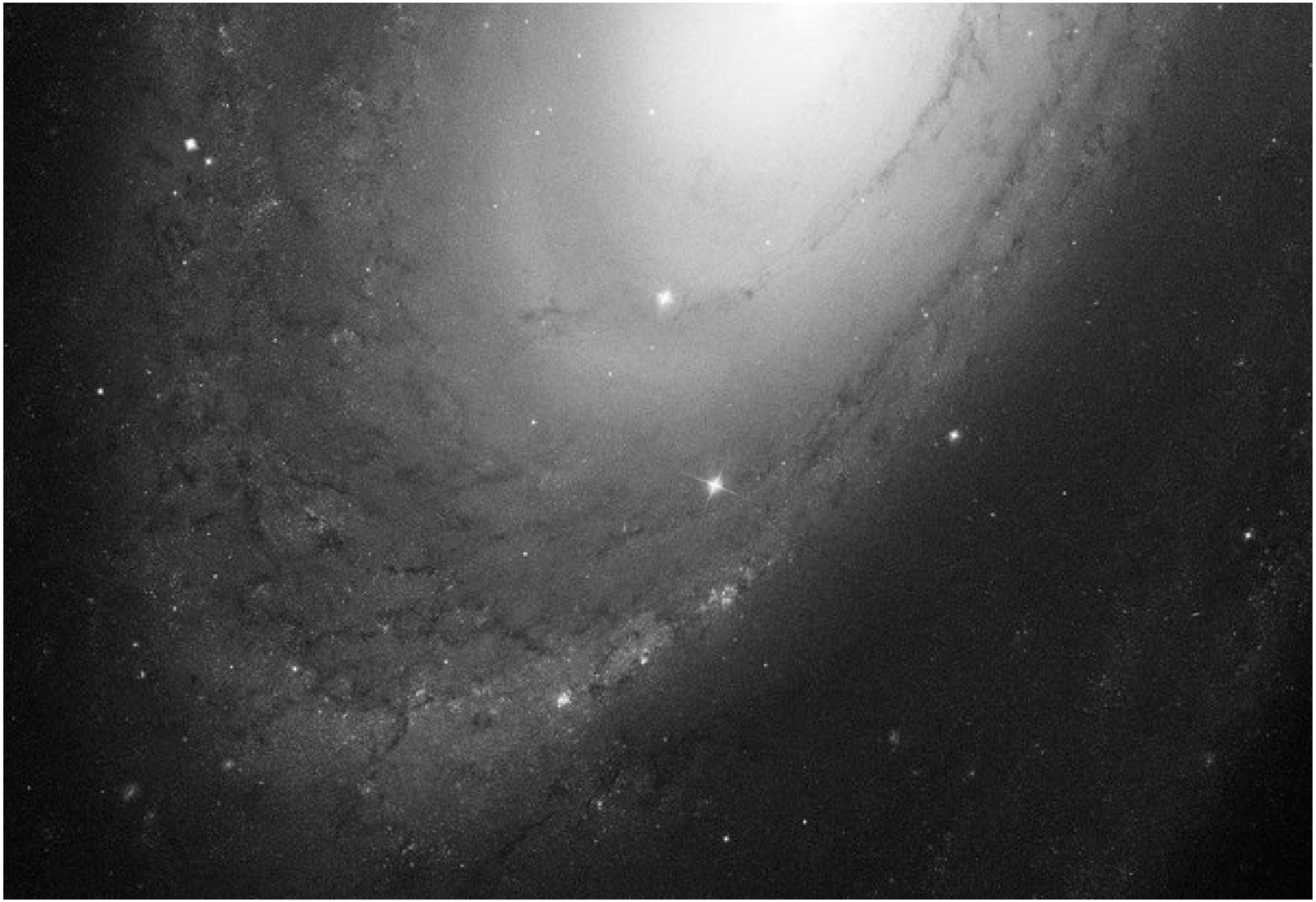


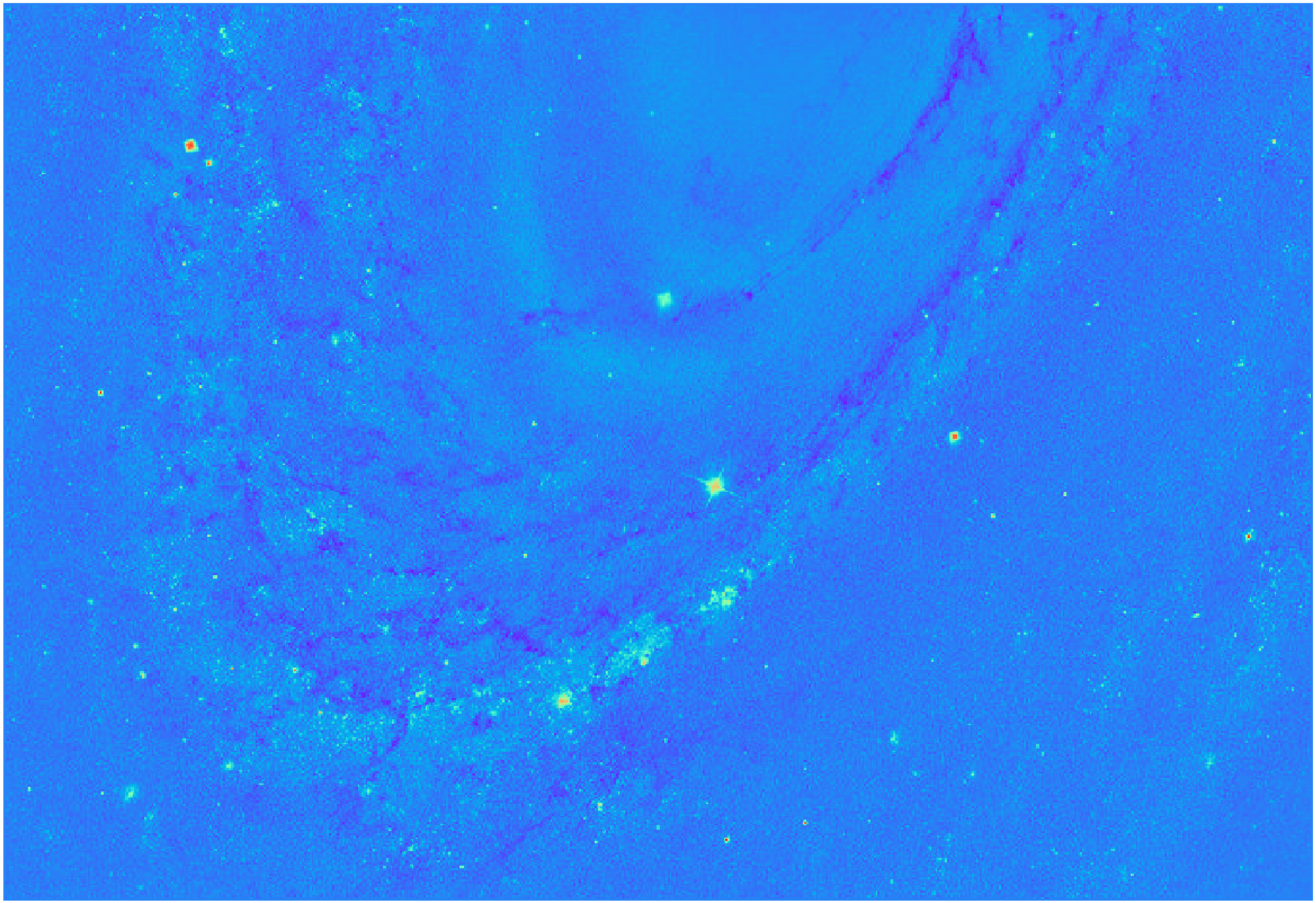
M81 after N=1200 Laplace iter. q=0.2



Unsharp masking technique:
Subtract blurred image from the original image
to remove the background, level the background gradients
and enhance visibility of overexposed features.







Finding Zeros of Real Functions

see http://planets.utsc.utoronto.ca/~pawel/pyth

- Bisection method simple_bisec.py
- Secant method simple_secant-2.py
- Newton's method simple_newton.py

- We will try these methods on a polynomial of the form

```
def fun(x):
    return((x-0.43)*(x-0.52)*(x-0.56))
```

We know in advance the three real roots: {0.43, 0.52, 0.56}

Also, in Newton's method, we can either analytically or numerically (which is less elegant/efficient but more general in applications) derive the first derivative df/dx of the function:

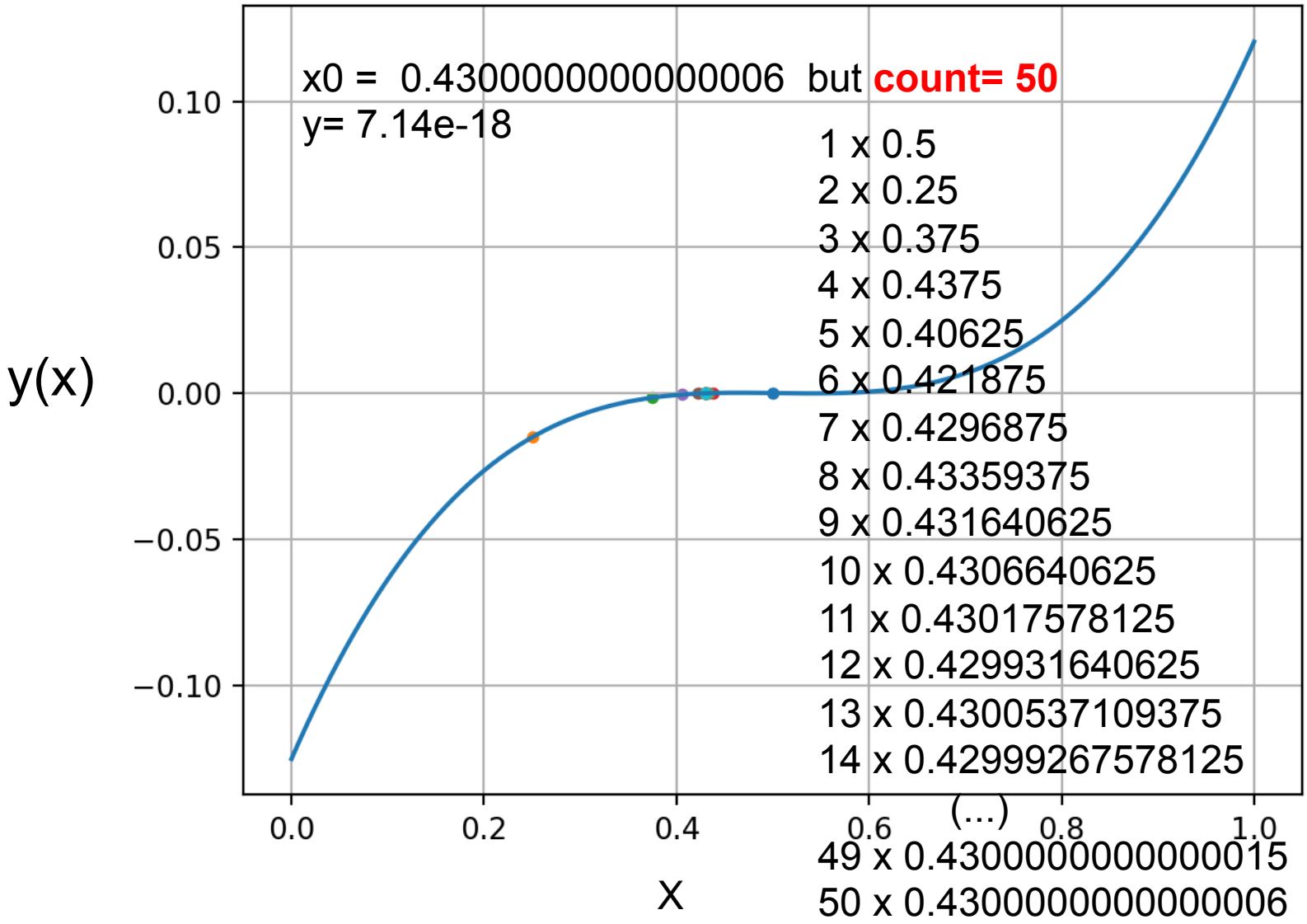
```
def dfdx(x):
    return ( (fun(x +1e-6) - fun(x -1e-6)) /2e-6 )
```

Zero Finding

see http://planets.utsc.utoronto.ca/~pawel/pyth

Bisection method

simple_bisec.py

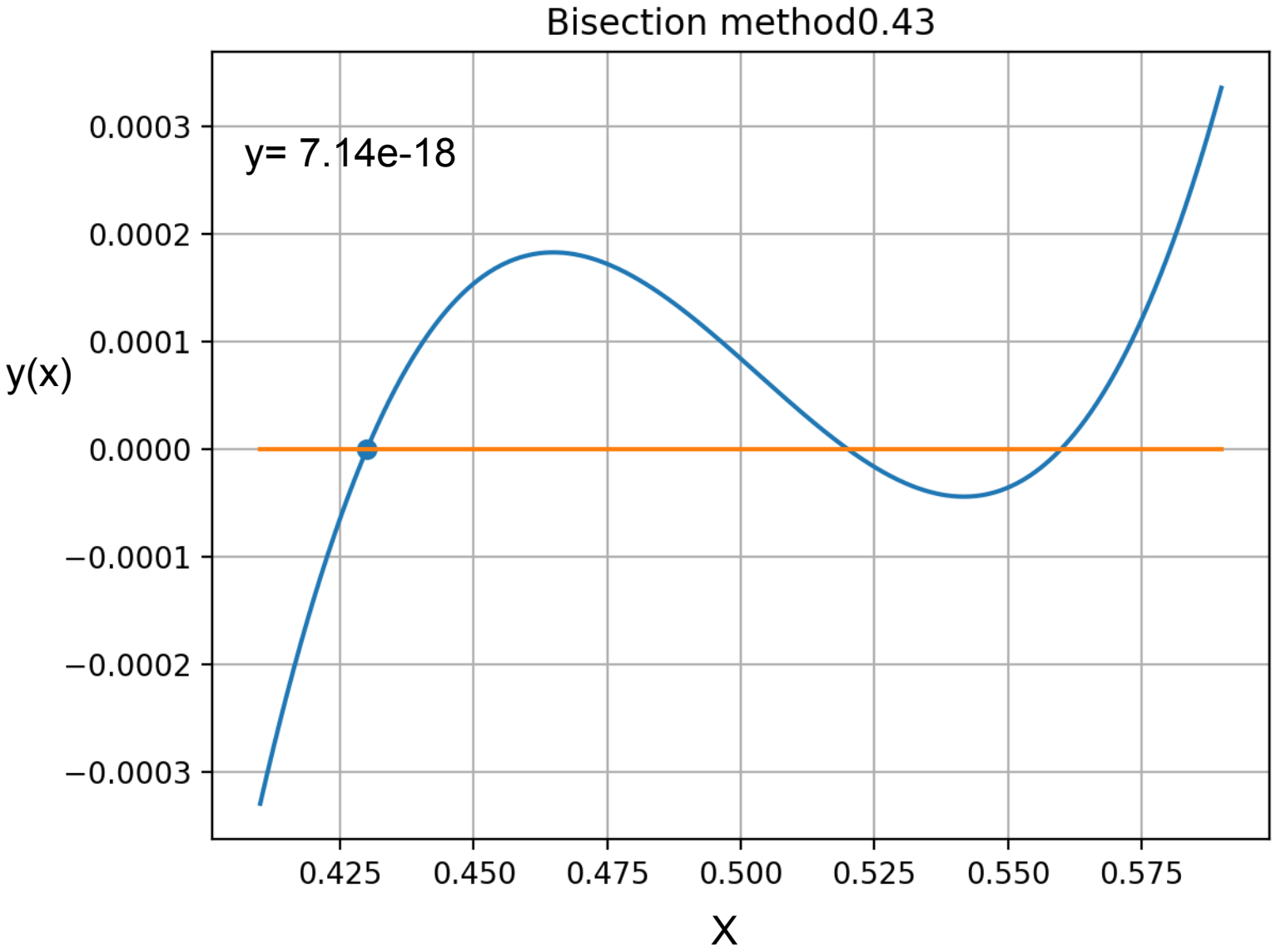


Zero Finding

see http://planets.utsc.utoronto.ca/~pawel/pyth

Bisection method

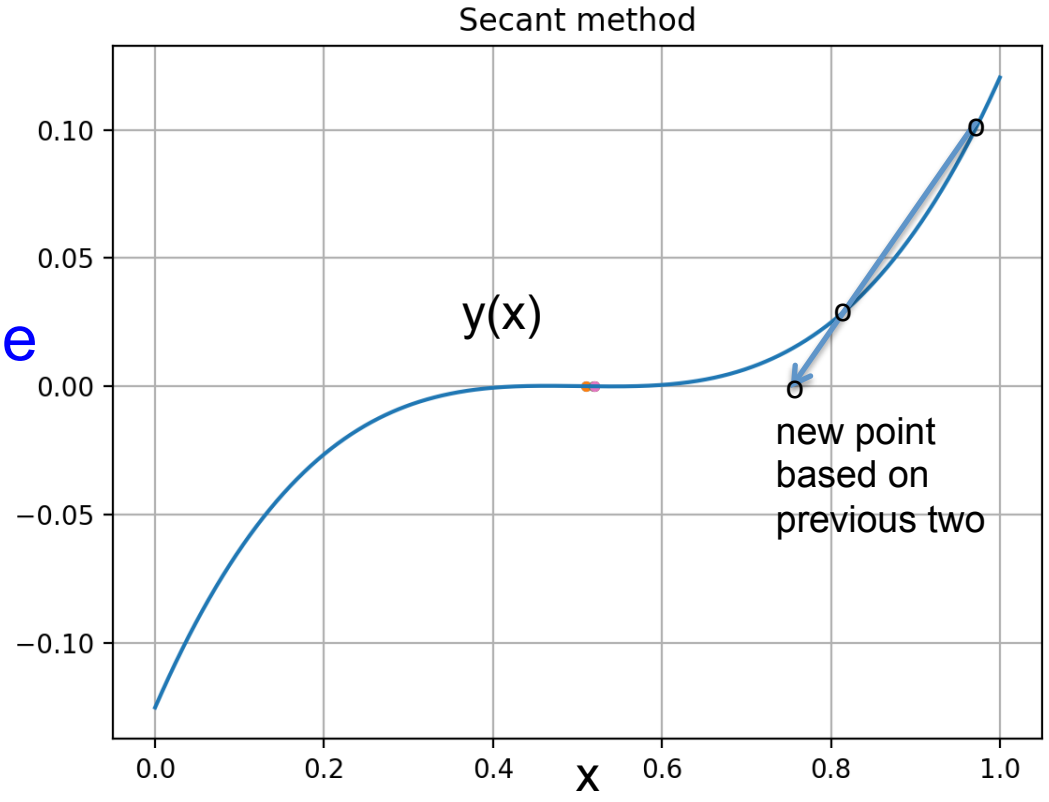
simple_bisec.py



Zero Finding see http://planets.utsc.utoronto.ca/~pawel/pyth

- Secant method

simple_secant.py



- $x_{n+1} = x_n - f(x_n)/S$

  $S = (f_n - f_{n-1})/(x_n - x_{n-1})$

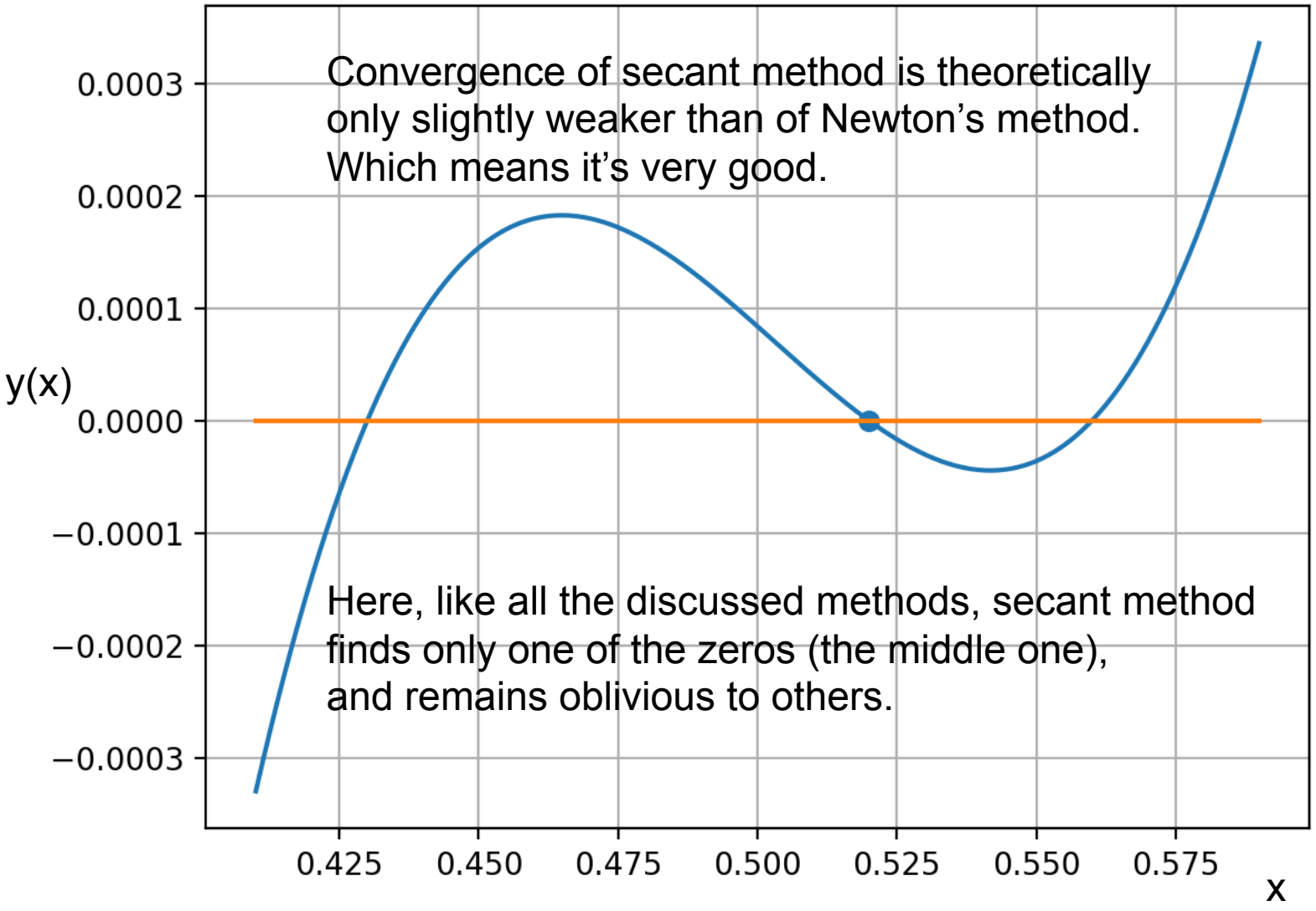
  S = slope of secant line

```
% python3 simple_secant.py
1 x 0.5098  dx -0.4902   y 4.07e-05
2 x 0.5097  dx -0.0002   y 4.14e-05
3 x 0.5193  dx 0.0096    y 2.61e-06
4 x 0.5192  dx 0.00064   y 2.58e-07
5 x 0.5199  dx 7.09e-05   y 2.51e-09
6 x 0.5199  dx 6.95e-07   y 2.49e-12
7 x 0.52    dx 6.90e-10   y 2.40e-17
 x0 =  0.52  count= -8  y= -0.0
t= 0.080s
```

Zero Finding

see http://planets.utsc.utoronto.ca/~pawel/pyth/

simple_secant.py

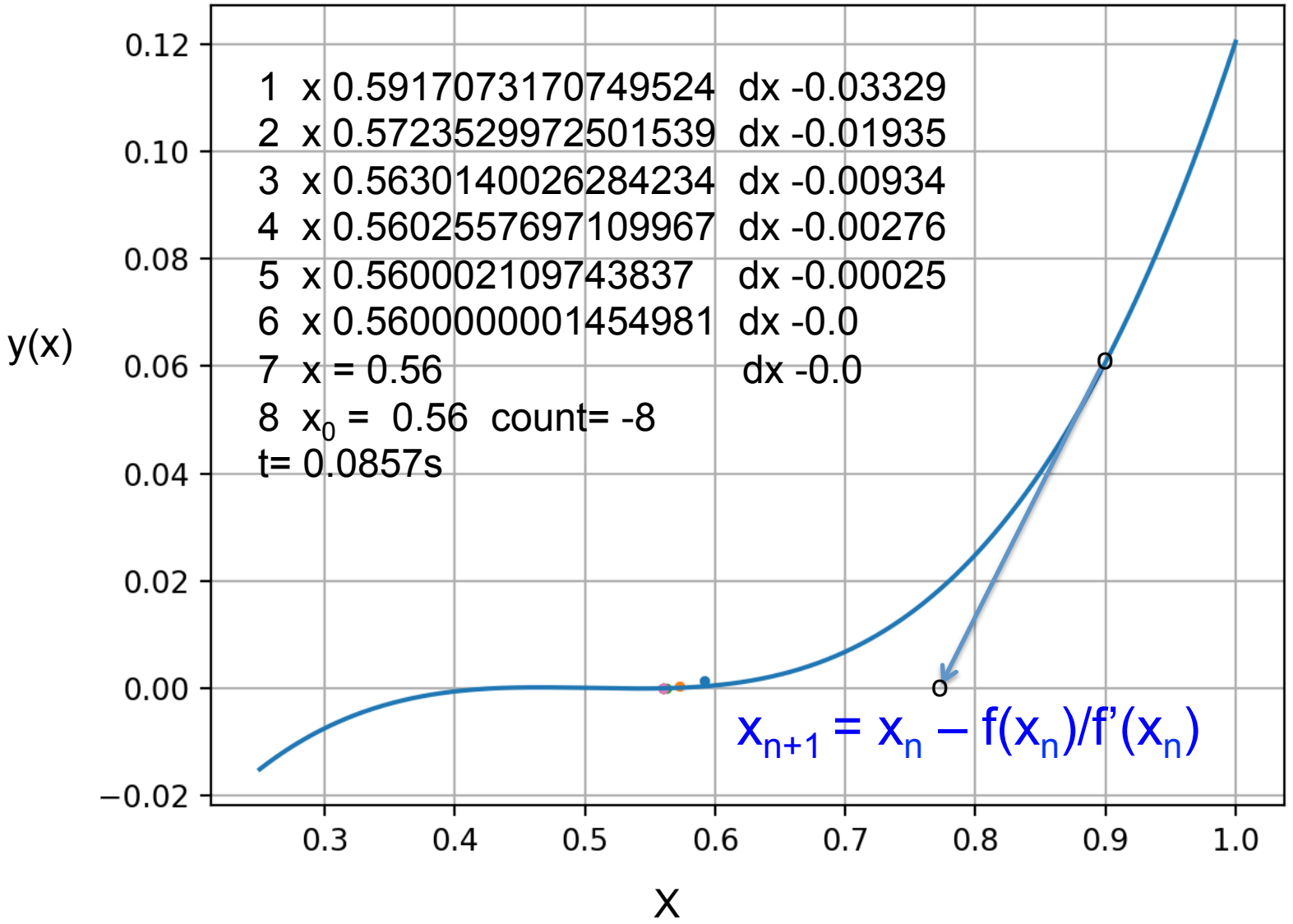


Zero Finding

see http://planets.utsc.utoronto.ca/~pawel/pyth

- Newton's method

simple_newton.py

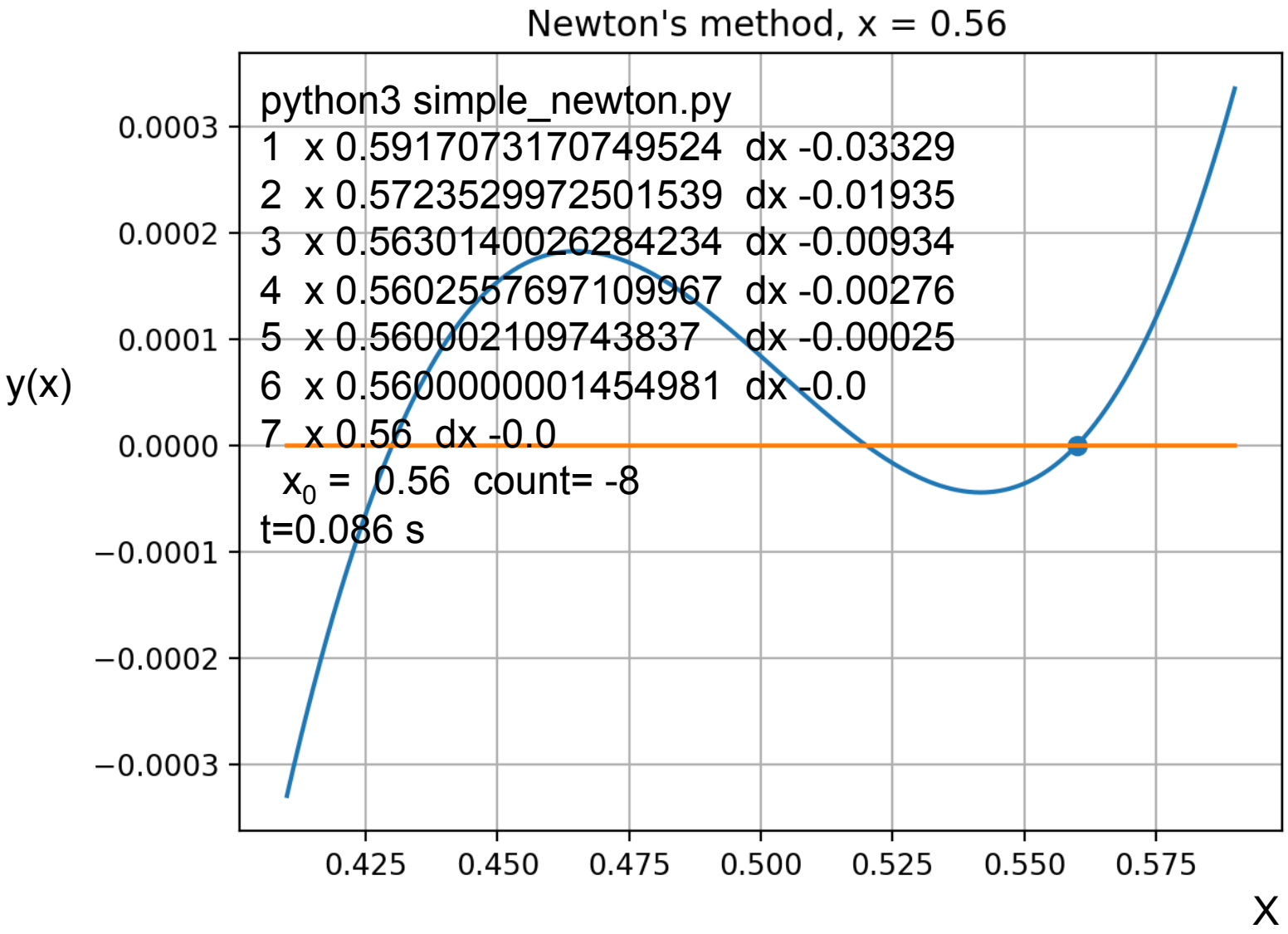


Root Finding

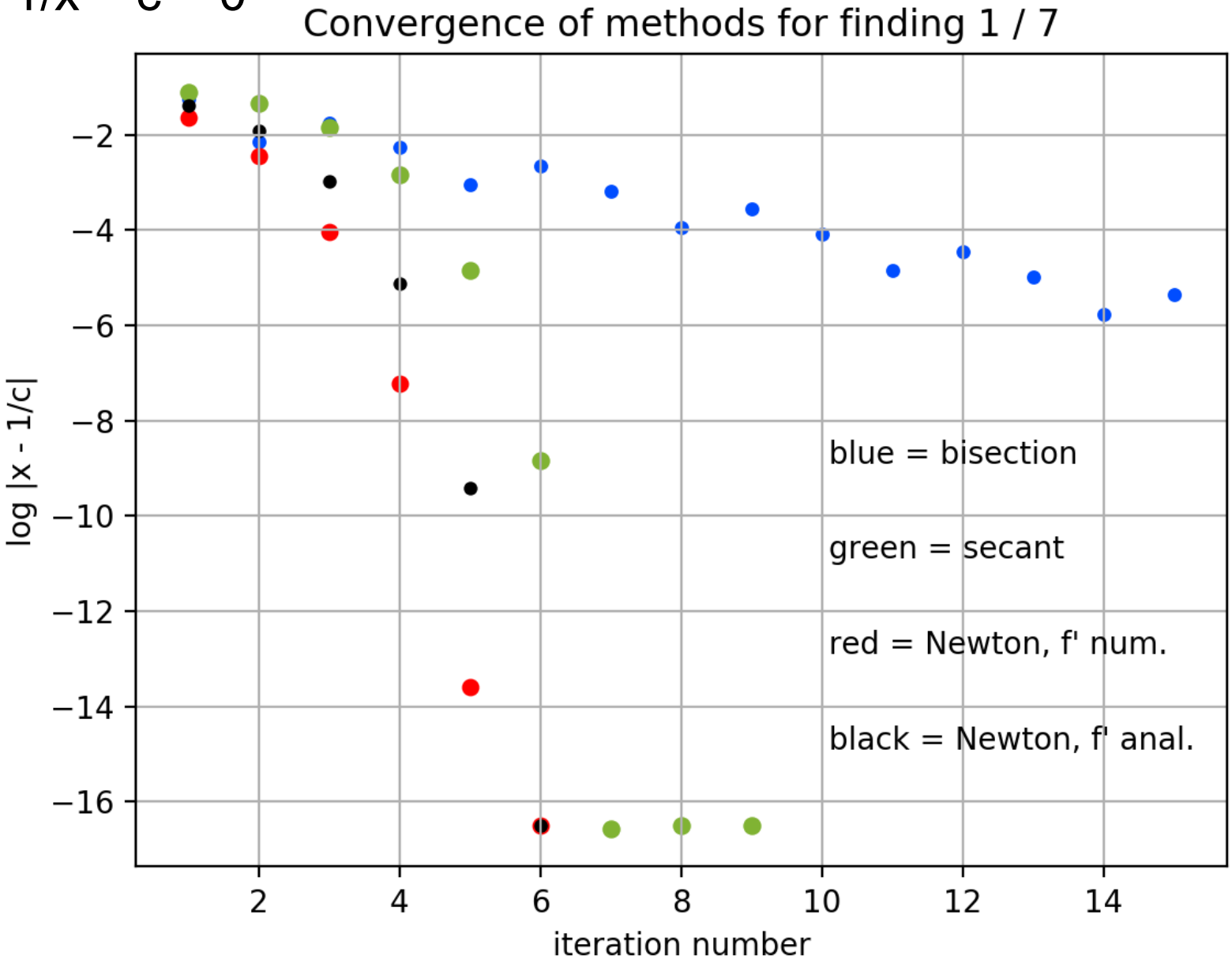
see http://planets.utsc.utoronto.ca/~pawel/pyth

- Newton's method

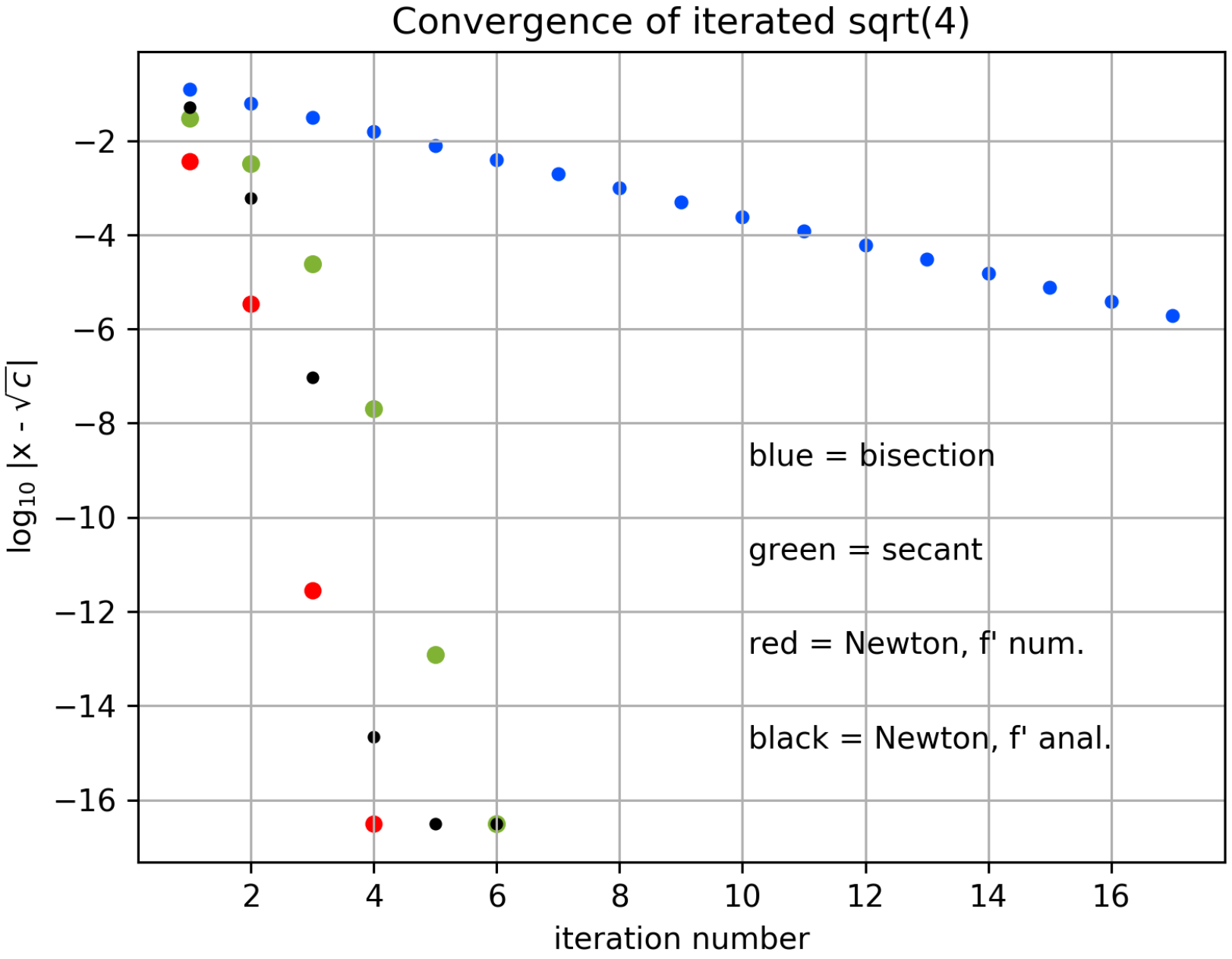
simple_newton.py



$f = 1/x - c = 0$


Convergence of methods for finding 1 / 7
log |x - 1/c|
−2
−4
−6
−8
−10
−12
−14
−16
2
4
6
8
10
12
14
iteration number
blue = bisection
green = secant
red = Newton, f' num.
black = Newton, f' anal.

$f = x^2 - c = 0$


Convergence of iterated sqrt(4)
log10 |x - √c|
−2
−4
−6
−8
−10
−12
−14
−16
2
4
6
8
10
12
14
16
iteration number
blue = bisection
green = secant
red = Newton, f' num.
black = Newton, f' anal.

Numerical calculus: integration

see http://planets.utsc.utoronto.ca/~pawel/pyth

err_int.py

- Theory discussed in texbook#2
- Turner et al., Springer 2018
- Section 3.2, p. 48

Peter R. Turner · Thomas Arildsen
Kathleen Kavanagh

Applied
Scientific Computing

With Python

3 Numerical Calculus 35
3.1 Introduction 35
3.2 Numerical Differentiation 38
3.3 Numerical Integration 48
3.4 Composite Formulas 58
3.5 Practical Numerical Integration 68
3.6 Conclusions and Connections: Numerical Calculus 78
3.7 Python Functions for Numerical Calculus 79

4 Linear Equations 81
4.1 Introduction 81
4.2 Gauss Elimination 84

Numerical integration

http://planets.utsc.utoronto.ca/~pawel/pyth

err_int.py

- Theory discussed in texbook#2
- Turner et al., Springer 2018
- Section 3.2, p. 48

- NEXT TIME..... READ AHEAD

- STUDY and RUN all the codes discussed in this lecture.
- HAVE FUN with Python3.