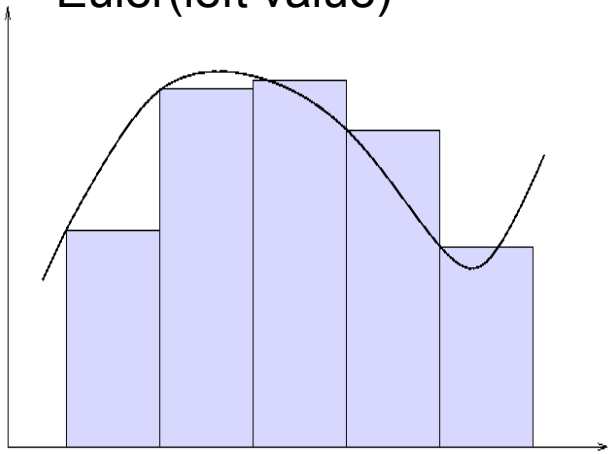
Lecture 9

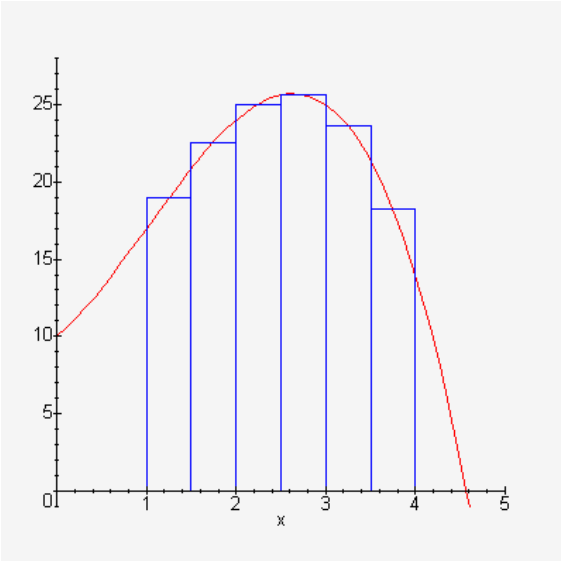
- Recap of integration:
  Formal proof of convergence of 2nd order integration methods
- **Linear algebra**
  - Sets of linear equations solved by Gauss elimination
  - Method of Least Squares
  - Fitting polynomials and other linear combinations of nonlinear functions to data
  - Fitting as denoising – one example in detail
  - Finding dark matter in galaxies
    - theory
    - observations
    - fitting

Last time we talked about these integration rules

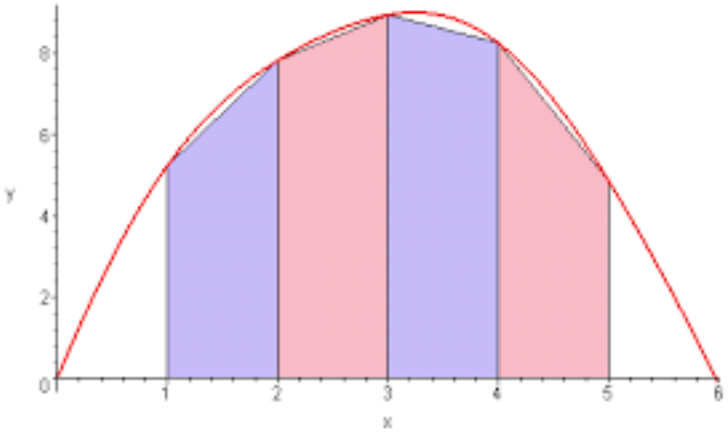
Euler(left value)

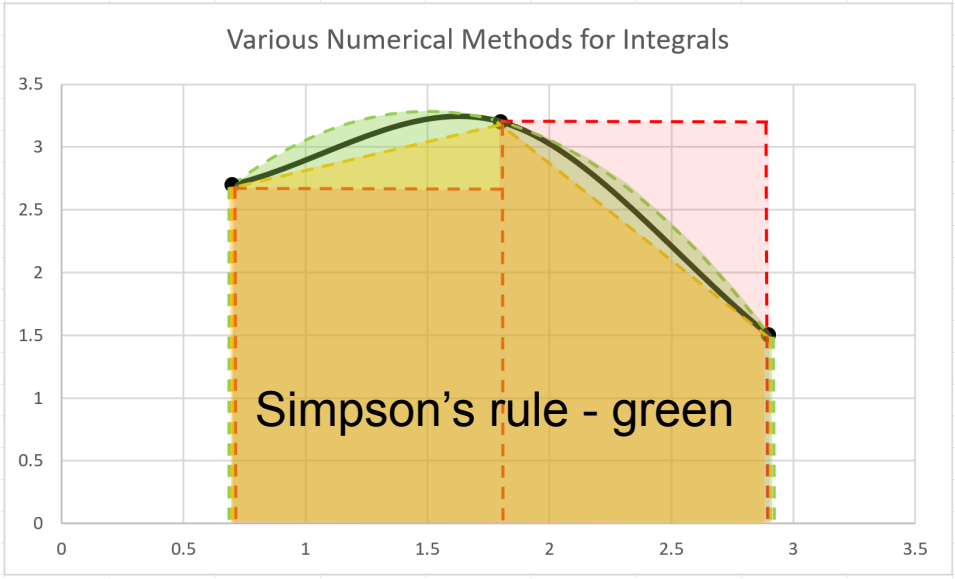


Midpoint method



Trapezoid method





- **Formal proof of 2nd order convergence of trapezoid integration rule:**
- Consider subinterval x_0 to x_1, of width $h = x_1 - x_0$. Denote $f_{\{0,1\}} := f(x_{\{0,1\}})$.
- Taylor expansion gives f(x) anywhere in this interval:
- $f(x) = f_0 + (x-x_0) f'_0 + \frac{1}{2} (x-x_0)^2 f''_0 + \ldots$
- In particular, we have for $x=x_1$:
- $f_1 = f_0 + h f'_0 + \frac{1}{2} h^2 f''_0 + \ldots$
- True integral of function f(x) in the subinterval equals, i.e., of it's Taylor expansion, is (as you should check by integration!)
- $I_{true} = h f_0 + \frac{1}{2} h^2 f'_0 + (h^3/6) f''_0 + \ldots$
- The trapezoid rule reads: $I = h (f_0 + f_1)/2$. Substituting f_1 from the blue formula above, we get
- $I = h (2f_0 + h f'_0 + \frac{1}{2} h^2 f''_0)/2 = h f_0 + \frac{1}{2} h^2 f'_0 + (h^3/4) f''_0 + \ldots$
- Thus $I = I_{true} - (h^3/12) f''_0 + \ldots = I_{true} - (h^3/12) f''(\xi)$, where ξ is somewhere in the subinterval.
- *The composite formula, constructed for N=(b-a)/h subintervals, has thus accuracy of order $O(N h^3) = O(h^2)$. Error of trapezoid integration on interval from a to b drops with shrinking h (or growing N) as ~h^2 (second order algorithm)* ■

- **Formal proof of 2nd order convergence of midpoint integration rule:**
- Consider subinterval x_0 to x_1, of width $h = x_1 - x_0$.
- Denote midpoint $\frac{1}{2}(x_0+x_1)$ as x_m, and $f_m := f(\frac{1}{2}(x_0+x_1)) = f(x_m)$.
- Taylor expansion gives f(x) anywhere in this interval:
- $f(x) = f_m + (x-x_m) f'_m + \frac{1}{2} (x-x_m)^2 f''_m + \ldots$
- True integral of function f(x) over the subinterval from $(x-x_m) =$ $-h/2$ to $(x-x_m) = +h/2$, equals (as you should check by integrating the blueTaylor expansion)
- $I_{true} = h f_m + (h^3/24) f''_m + \ldots$
- The midpoint rule reads: $I = h f_m$. The inaccuracy is thus
- $I - I_{true} = +(h^3/24) f''_m + \ldots = +(h^3/24) f''_m(\xi)$, where ξ is somewhere in the subinterval. This is $-\frac{1}{2}$ of the trapezoid error.
- *The composite midpoint formula for N=(b-a)/h subintervals, has thus accuracy of order $O(N h^3) = O(h^2)$. Error of midpoint integration on interval from a to b drops as $\sim h^2$ (2^{nd} order algorithm)* ∎

- **Formal proof of the 4th order of Simpson's rule:**
- Let x_m be the midpoint of the subinterval of width h.
- The trapezoid rule reads: $I_{tra} = h\,(f_0 + f_1)/2$ and has error on subinterval of width h which can be written as
- $I_{tra} - I_{true} = -(h^3/12)f''_m + O(h^5 f'''_m)$.
- The midpoint rule reads: $I_{mid} = h\,f_m$, and has error
- $I_{mid} - I_{true} = +(h^3/24)f''_m + O(h^5 f'''_m)$.
- Notice that all even powers of h & odd derivatives are absent from error terms. They cancel upon integration, because both rules are symmetric w.r.t. midpoint. All integrals from –h/2 to +h/2 of: $x\,f'_m$, $x^3 f'''_m$, $x^5 f^{(v)}_m$, ... , are zero, because of anti-symmetry of those functions. They contribute equal amounts of positive and negative contributions to error. That explains the $O(h^5)$ terms.
- How can we join the two rules to eliminate the $O(h^3)$ error term? By combining twice as much midpoint than trapezoid rule, i.e., by adding 1/3 of the first to 2/3 of the second.

- **Formal proof of the 4^{th} order of Simpson's rule:**

- That's Simpson's rule:

$$I_{Simp} = h(f_0 + 4f_m + f_1)/6, \qquad \text{with error}$$

$$I_{Simp} - I_{true} = O(h^5 f'''_m(\xi)), \text{ where } f'''_m = f'''_m(\xi) \sim \text{const.}$$

- *In [a,b] interval, the composite Simpson's rule*

$$I = h\,(f_0 + 4f_1 + 2f_2 + 4f_3 + \ldots + f_N)\,/\,6$$

for N=(b-a)/h subintervals, has accuracy of order $O(N h^5) = O(h^4)$. It's a 4^{th} order algorithm. ∎

Computer Linear Algebra

- Read Turner et al., chapter 4, p. 81
- If you do not remember basic linear algebra (vectors, matrix notation, what is a linear system of equations, matrix inversion by Gauss elimination, linear systems and their solutions, what is an eigenvalue and eigenvector), please ask TA during tutorial, after first looking at the link below.

Carl Friedrich Gauss (1777-1855)

- https://saylordotorg.github.io/text_intermediate-algebra/s06-05-matrices-and-gaussian-eliminat.html

Computer Linear Algebra

Examples of some scientific problems leading to linear systems of equations:

- Fitting theory to data
- De-noising measurements if underlying trends understood and can be modeled mathematically
- Solving discretized differential equations of science and engineering (many examples, as most of science since about 200 years ago uses the language of differential equations)
- Finding eigenvalues such as energy E in Schroedinger eq. of quantum mechanics, or rates of growth of spiral density waves in galaxies
- Finding modes in mechanical systems, e.g. shapes of the fastest growing modes in disk galaxies, distribution of stress and dilatation in vibrating objects.

- In addition, matrix operations are ubiquitous in signal and image processing, as well as machine learning (so-called AI = artificial intelligence)

Numerical fitting: Least Squares

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

Theory and methods:

- Linear systems of equations, Gauss elimination
- Least Squares Method (of C.F. Gauss, who else)
- Application: function fitting to data, i.e.
- Finding linear combinations of nonlinear functions to minimize chi-squared

(Computer) Linear Algebra

- To begin working with Python, start with this:
- A x = b [matrix times vector equals vector]

A linear system of N eqs. for N unknowns, A is a square NxN matrix, b the vector of constants of length N, x is the vector of N unknowns.

- x = np.linalg.solve(A,b) # Numpy solves the system

- How is a system of linear equations solved?

Gauss elimination

- Turner et al Chapter 4 Linear Equations, read pp. 81-93
- (skip 4.3 LU Factorization)

- The simplest & practical method is by C.F. Gauss, and it's called Gauss elimination.
- It works by eliminating unknowns one by one
- It uses matrix (augmented matrix) notation

Numerical fitting: Least Squares

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

Theory and methods:

- Linear systems of equations, Gauss elimination
- Least Squares Method (of C.F. Gauss, who else)
- Application: function fitting to data,
- Finding linear combinations of nonlinear functions to minimize chi-squared

- Fit linear polynomial to noisy data fit-2par.py
- Fit parabola (quadratic polynomial fit-3par.py
- Fit a more complicated function fit-4par.py

Least squares method

- A x = b [matrix times vector equals vector]

- x = np.linalg.solve(A,b) # Numpy solves the system

- Consider now the application called least-squares fit to data.
- What is it?
- It is a method C.F. Gauss first invented for finding parameters (called elements) of orbits of asteroids and comets in our solar system, from z minimum number of observations.
- It works by finding such coefficients of a linear combination of complicated functions, which provide the best fit to a data set.
- fit-3par.py - see our Python code repository.

Least Squares Method

- Turner et al Chapter 4.5, p. 115
- General function fit (blackboard calculation)
- $y(x) = c[0]\, f_0(x) + c[1]\, f_1(x) + \ldots + c[N-1]\, f_{N-1}(x)$

- c is a vector of N unknown coefficients we would like to compute.
- All f's are known, arbitrarily complicated and non-linear, even discontinuous, functions.
- Require that sum of squares of deviations of $y(x_i)$ from data $Y(x_i)$
- is minimal (i.e. the least)

- Notice that this method tries to avoid large deviations, which cause biggest contributions to the sum.

Least squares fit (parabola fit to noisy data): program fit-3par.py

- "Hidden truth" or a theoretical dependence is in the form
- y_th = a[0] f[0,x] + a[1] f[1,x] + a[2] f[2,x]
- = a[0] x*x + a[1] x + a[2]
- a = a_th = [-1, 2, 0] # assumed values. Q: can they be # recovered by the program?
- x-axis is 100 uniformly spread values from 0 to 3.
- Noise added to simulate observational errors. Y holds 100 y-values:
- Y = x*(2-x) + (np.random.rand(M)+np.random.rand(M)-1)
- Find set of parameters a = a_best_fit giving the best fit:
- E = np.sum((y_th-Y)**2) → minimum
- Taking N derivatives after all a[n] yields N equations with N unknown a's:
- A a = b [N x N system of linear equations]
- A[m,n] = np.sum(f[n,x]*f[m,x]) # see L9 for more details
- b = np.sum(f[n,x]*Y)) # see L9 for more details
- a = A^{-1} b # solution written in terms of inverse matrix A^{-1}

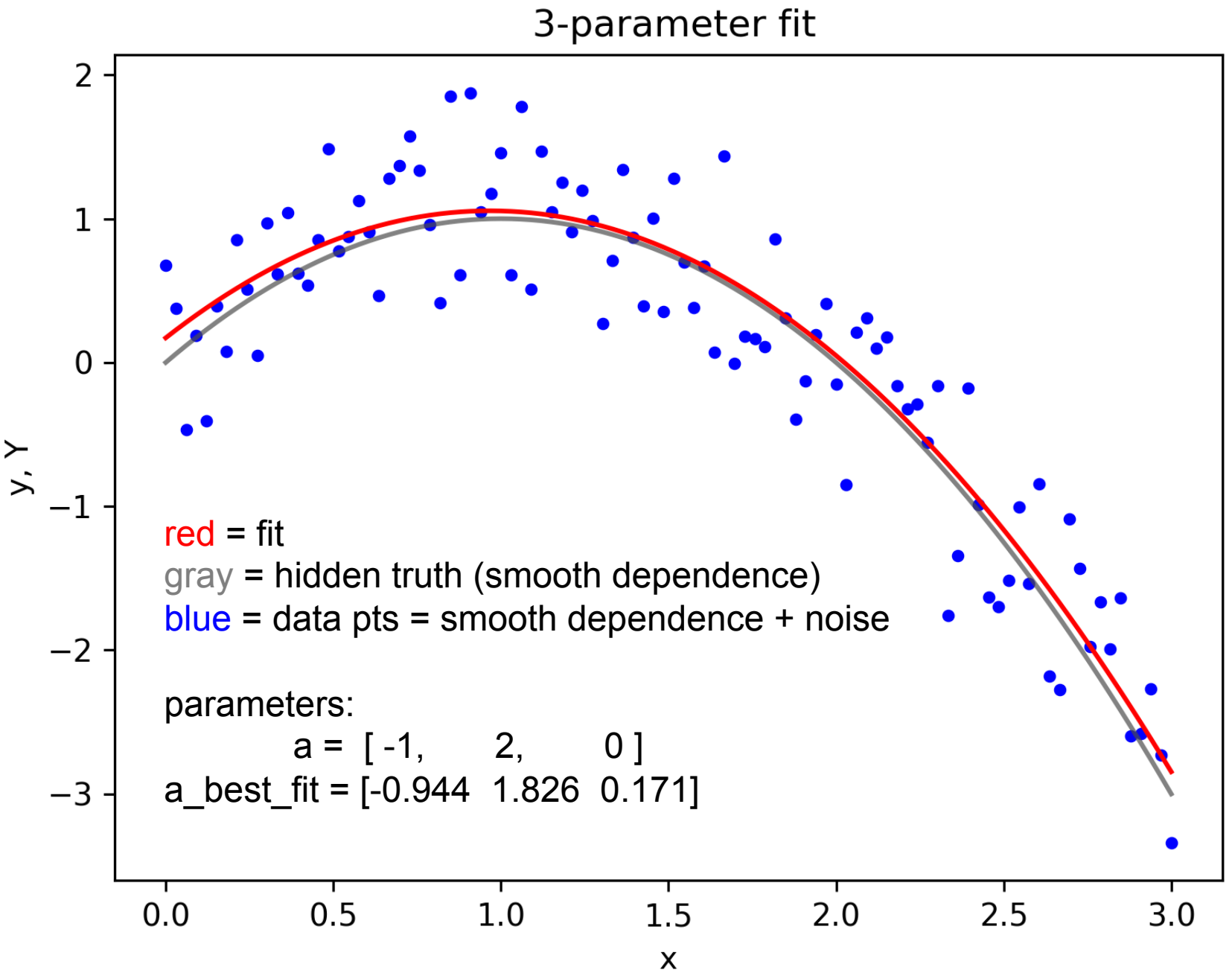
Numerical fitting: Least Squares

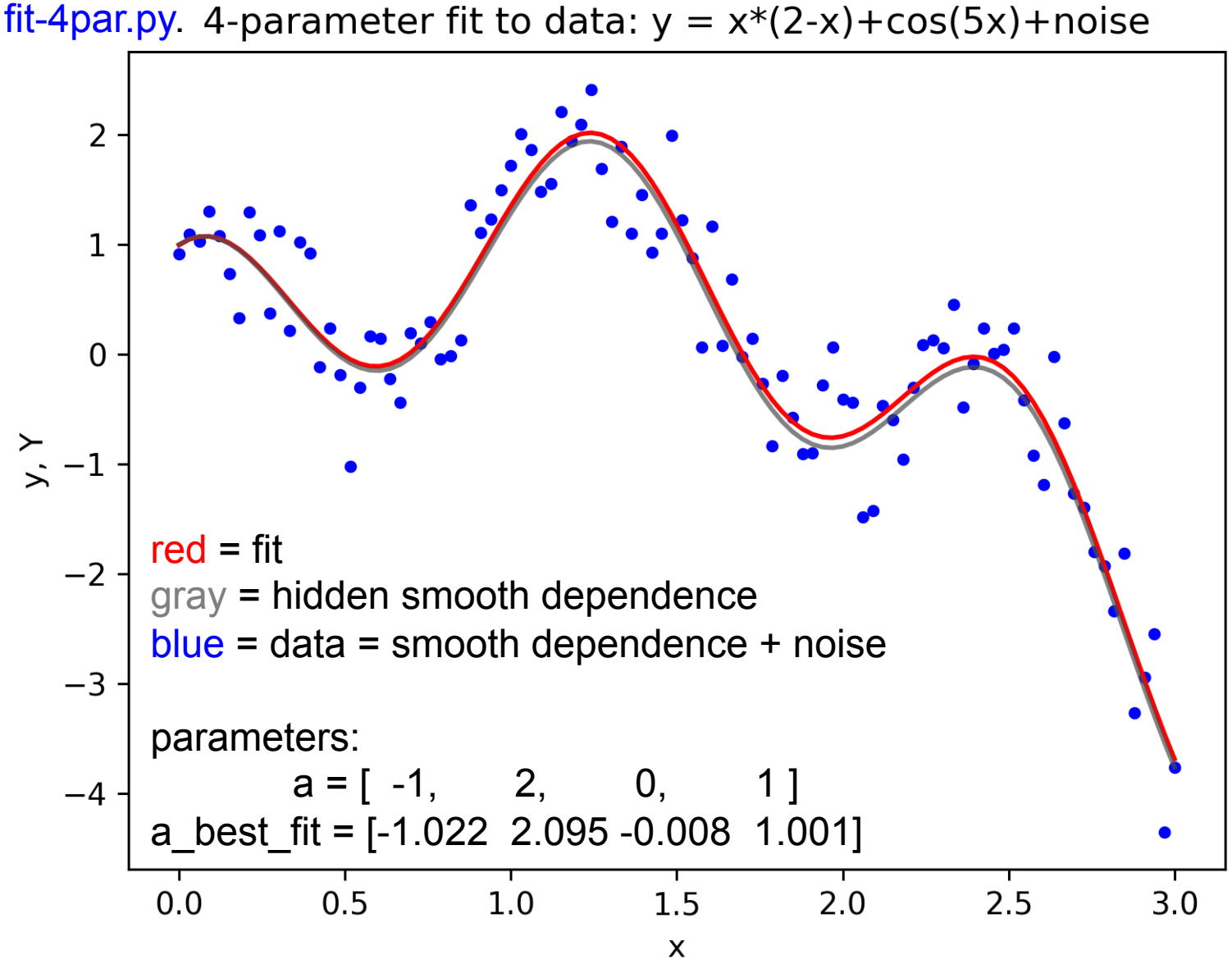
see http://planets.utsc.utoronto.ca/~pawel/pyth

Theory and methods:

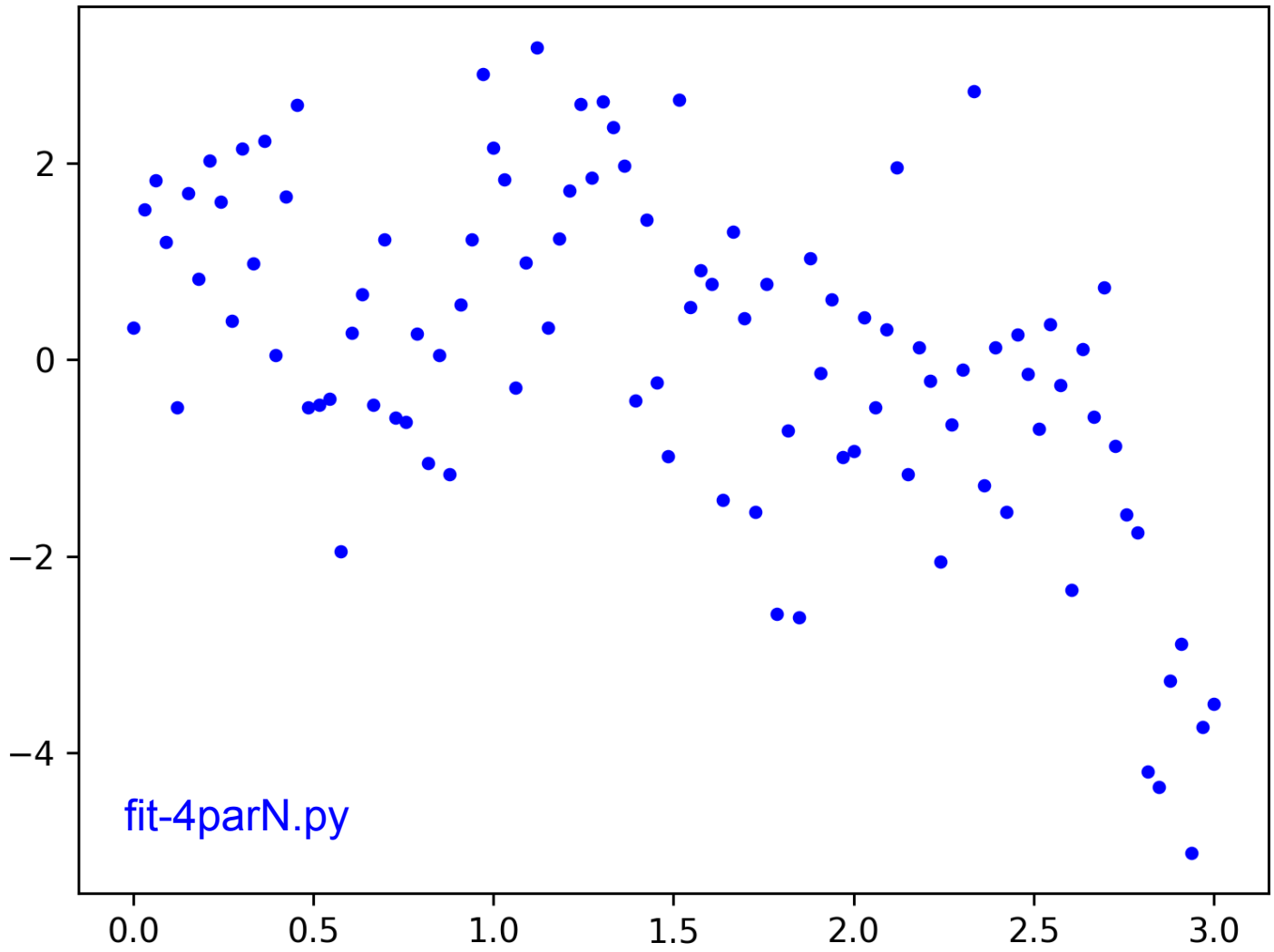
- Linear systems of equations, Gauss elimination
- Least Squares Method (of C.F. Gauss, who else)
- Application: function fitting to data, denoising data

- Fit linear polynomial to noisy data fit-2par.py
- Fit parabola (quadratic polynomial fit-3par.py
- Fit a more complicated function fit-4par.py

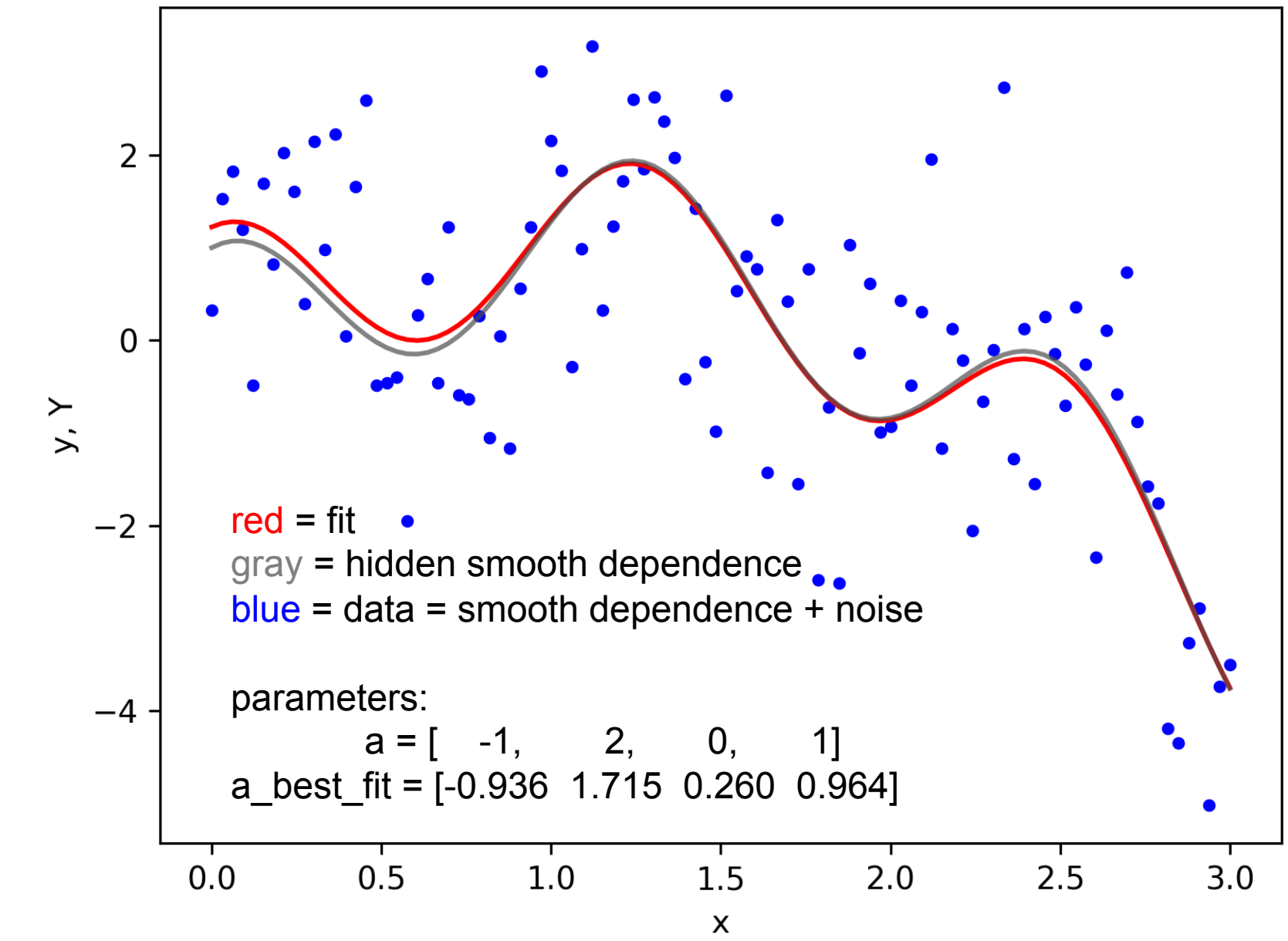

3-parameter fit
y, Y
x
red = fit
gray = hidden truth (smooth dependence)
blue = data pts = smooth dependence + noise
parameters:
a = [-1, 2, 0]
a_best_fit = [-0.944 1.826 0.171]



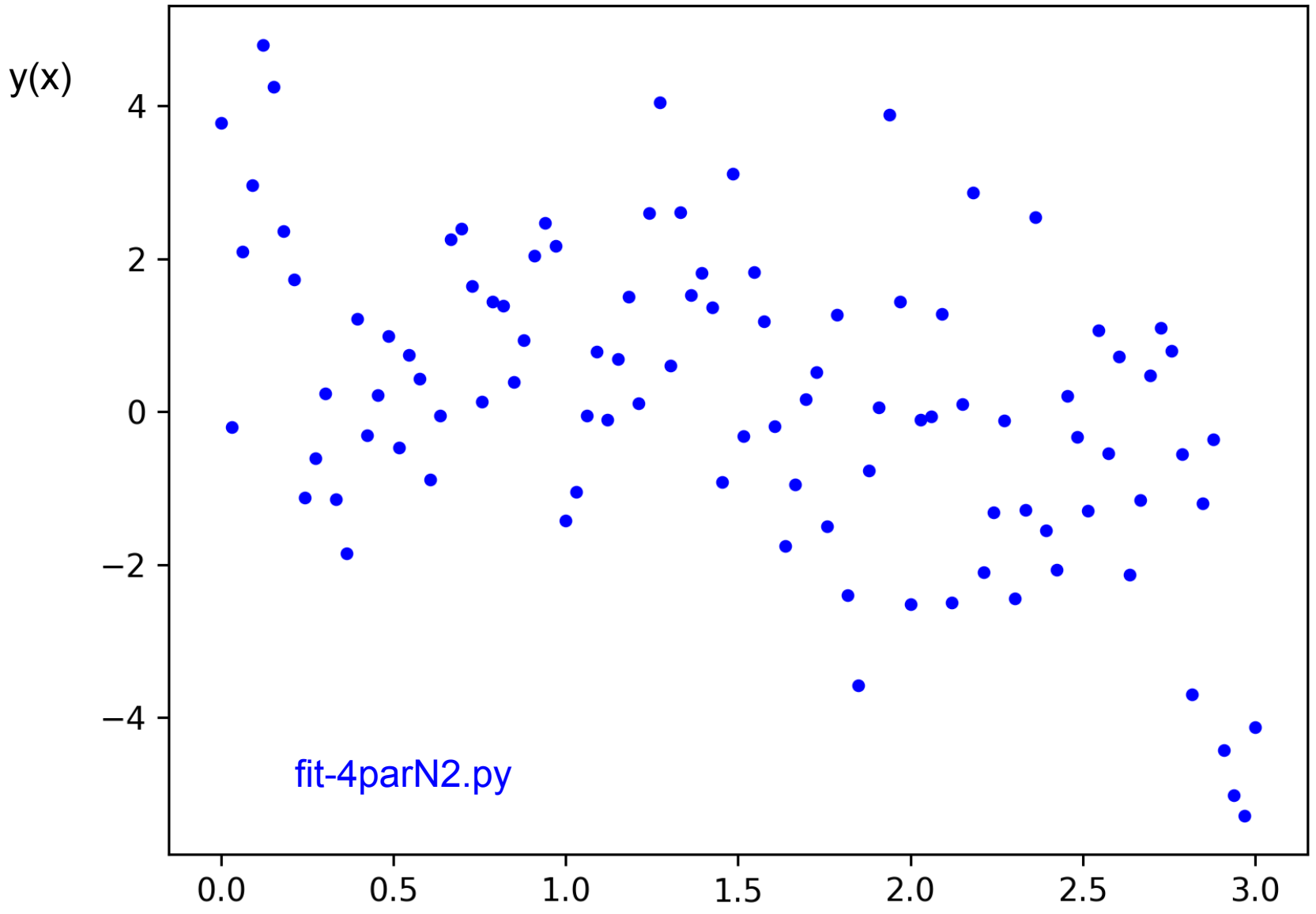
Finds the best combination of a parabola and sin(5x) that fits the data. Uncovers correctly all the parameters gray curve that was used to generate the blue points. This effectively de-noises the data set. Notice that we used the knowledge of what kind of functions to use, but that is often clear from the scientific context of a problem.



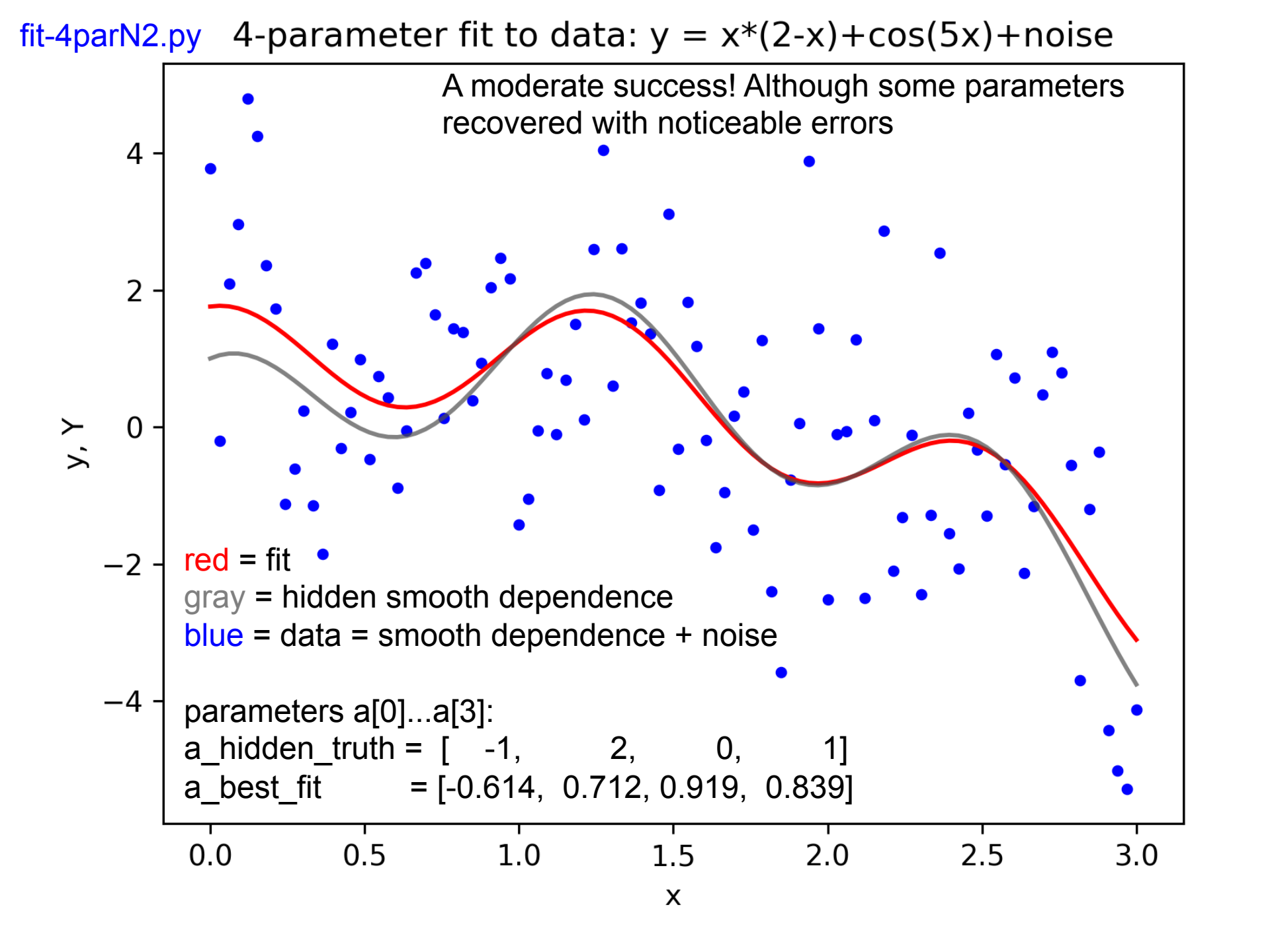
So much noise has been added here (varying in every execution of the code) that it becomes very hard to see what kind of periodic signal is hidden. Only something resembling a parabolic trend can be noticed. Let's now see how the linear algebra of the Least Squares method is going to recover for us the underlying regular variations.



Very nice recovery (denoising) of the “true” variations obscured by strong noise



Let’s add even more noise and test the strength of the method!
Can this data possibly be de-noised??


fit-4parN2.py
4-parameter fit to data: y = x*(2-x)+cos(5x)+noise
A moderate success! Although some parameters
recovered with noticeable errors
red = fit
gray = hidden smooth dependence
blue = data = smooth dependence + noise
parameters a[0]...a[3]:
a_hidden_truth = [-1, 2, 0, 1]
a_best_fit = [-0.614, 0.712, 0.919, 0.839]
y, Y
x
4
2
0
−2
−4
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Numerical fitting: Least Squares

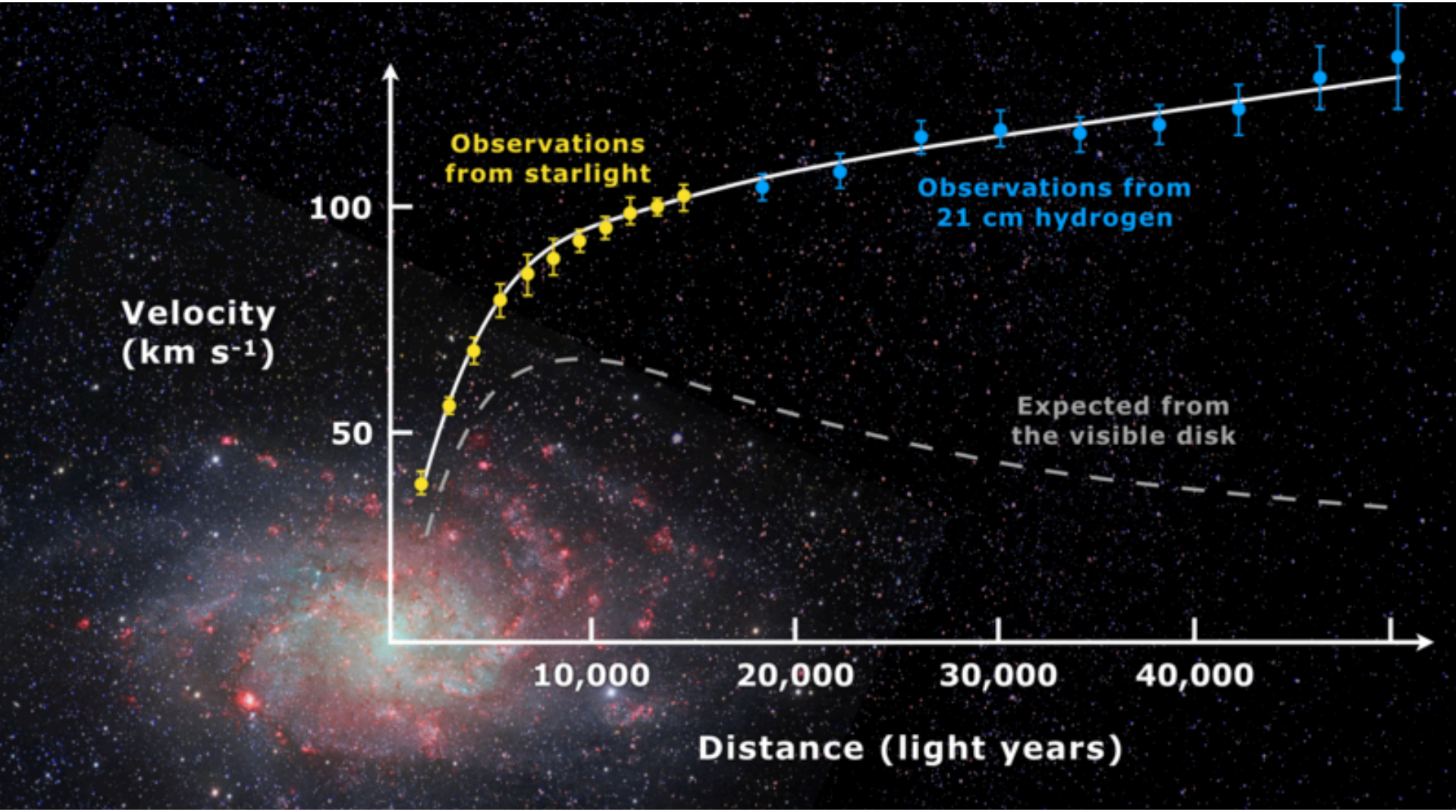
see http://planets.utsc.utoronto.ca/~pawel/pyth

Theory and methods:

- Least Squares Method (of C.F. Gauss, who else)
- Application: function fitting to data, i.e.
- Finding linear combinations of nonlinear functions to minimize chi-squared

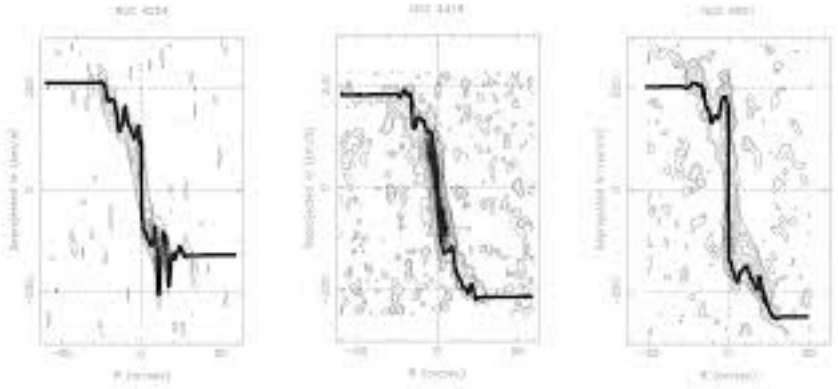
- Characterize Dark Matter in galaxies based on rotation curves fit-glx-4+2b.py

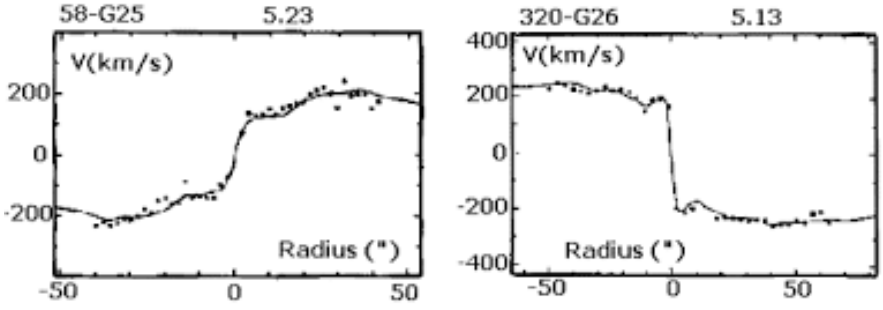
cf. https://en.wikipedia.org/wiki/Galaxy_rotation_curve

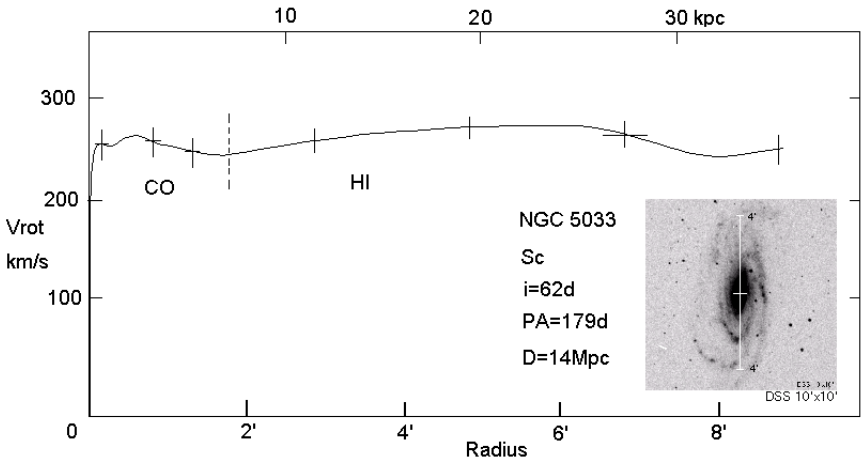


Rotation of galaxies and the measurements of Dark Matter

- Rotation curve is obtained from Doppler effect measurement in the light coming from galaxies approaching and receding parts



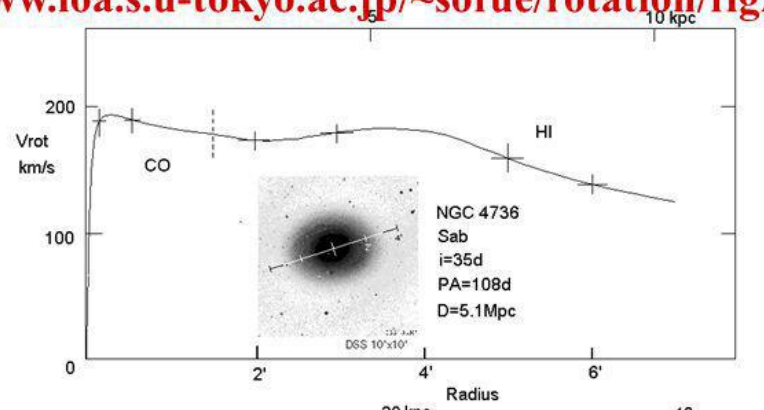


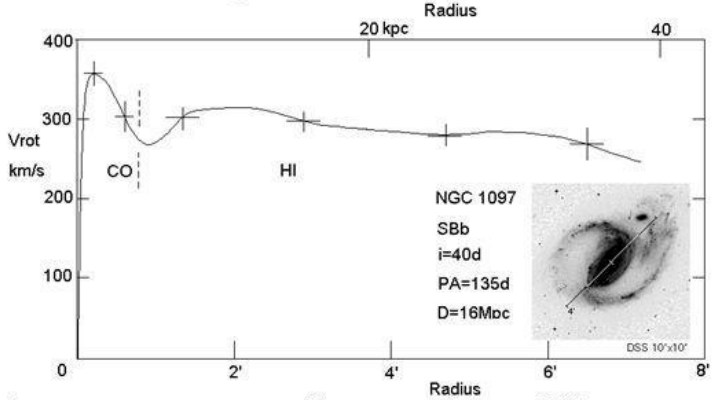


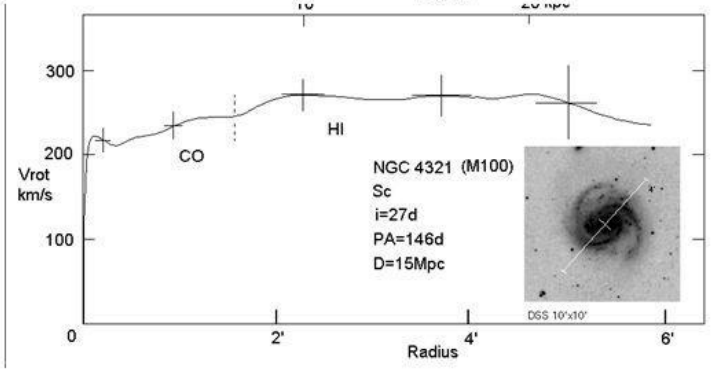
Examples of galaxy rotation curves, see

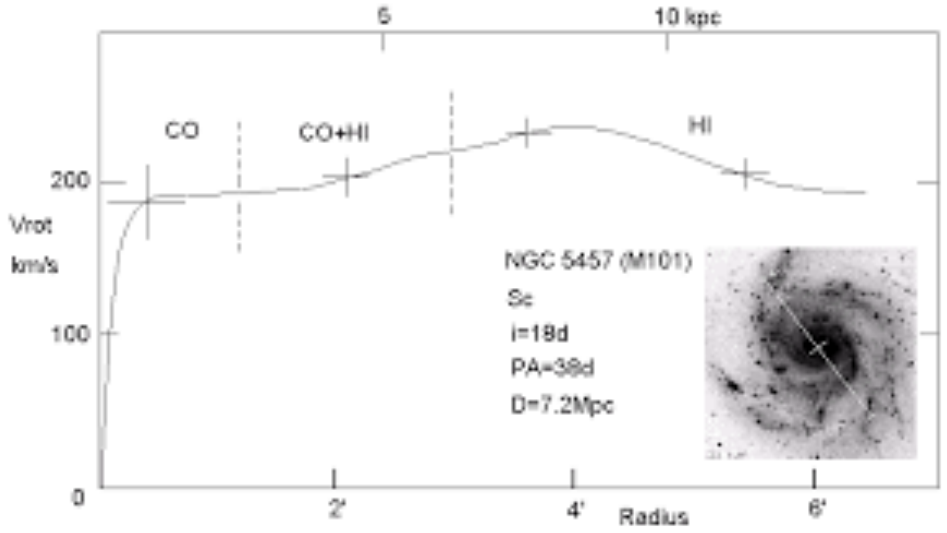
Y. Sofue, PASJ 49 (1997) p17

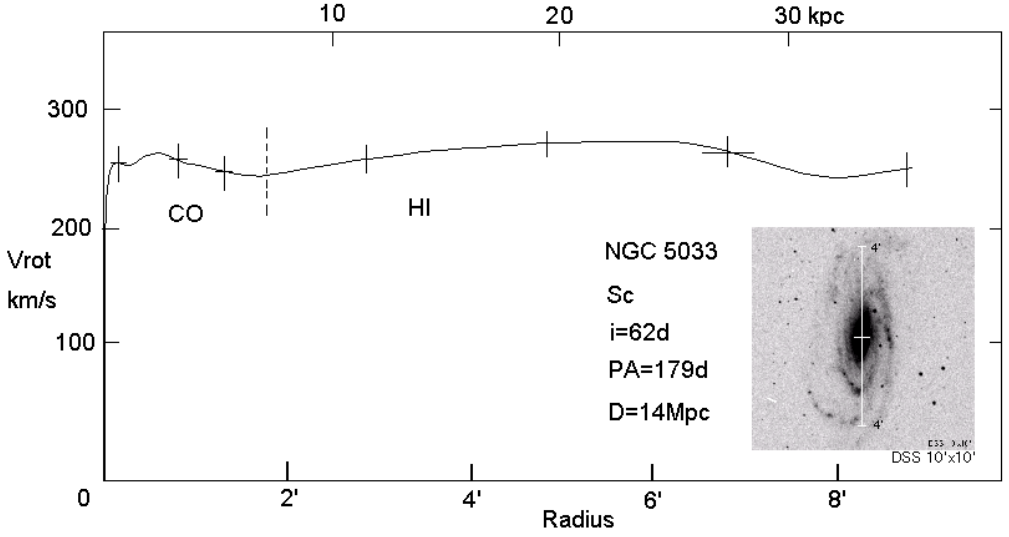
www.ioa.s.u-tokyo.ac.jp/~sofue/rotation/fig2.htm





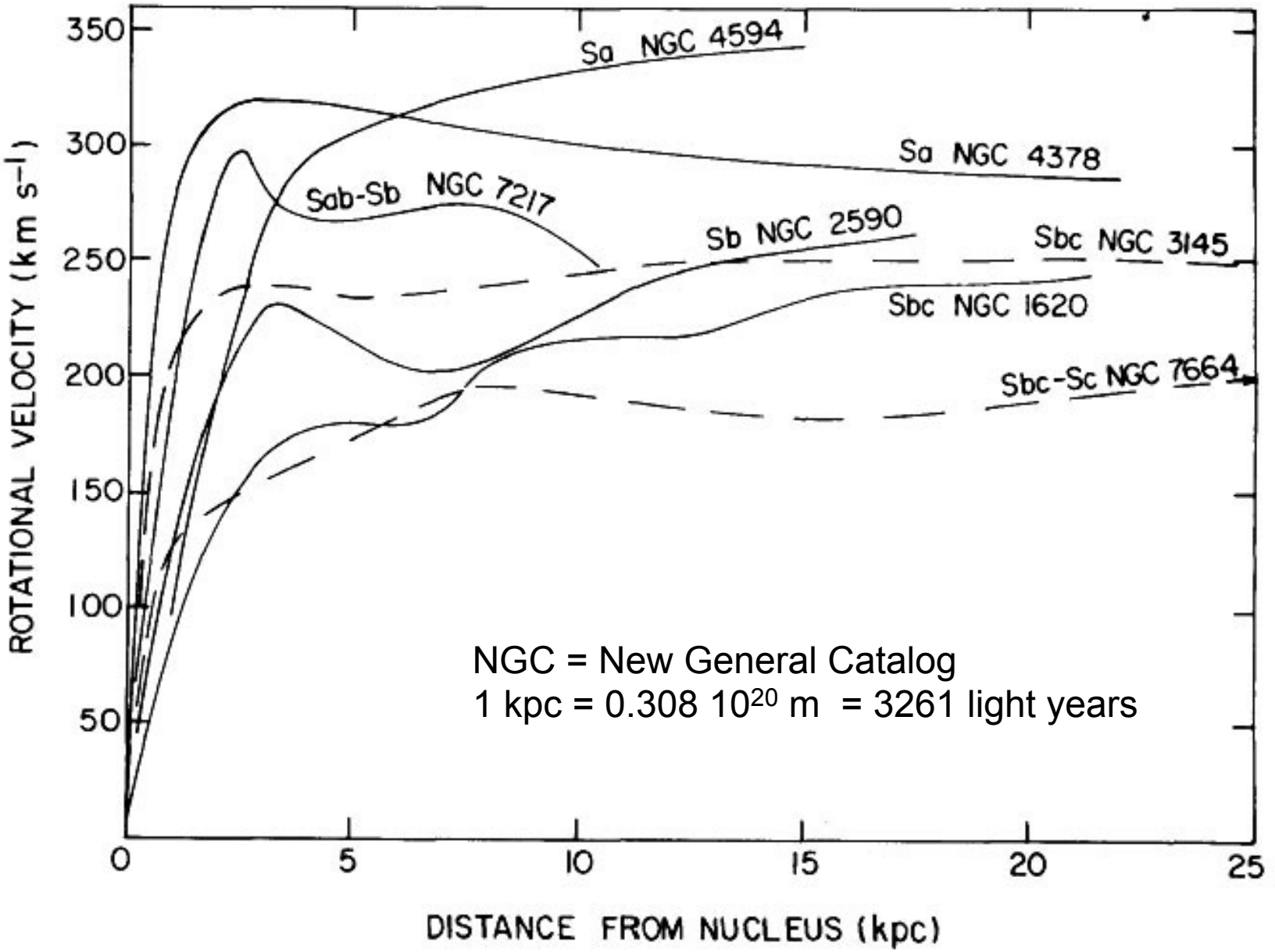






2GLSS P.H. Regan

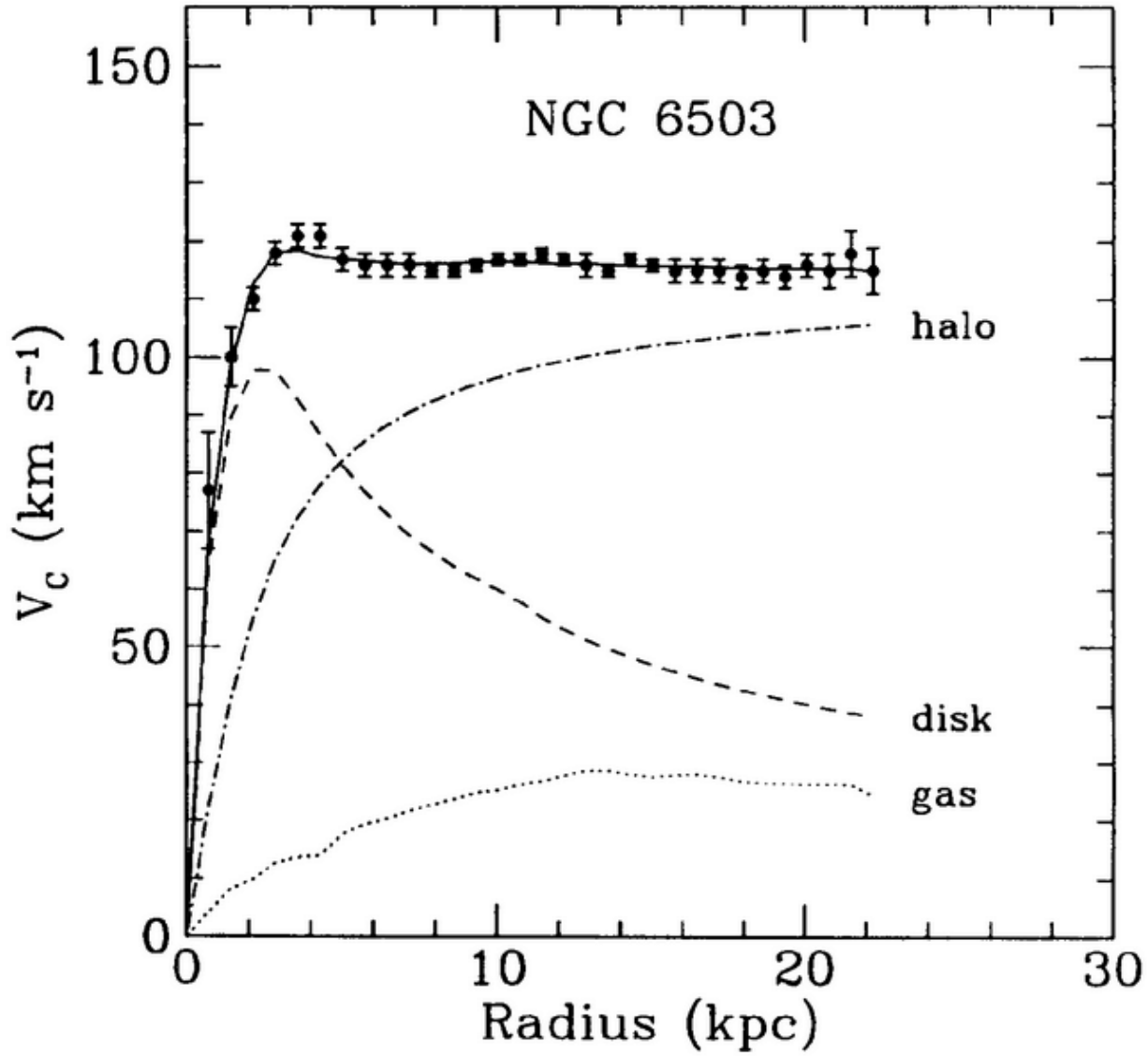
Observations of flat galaxy rotation curves



NGC = New General Catalog

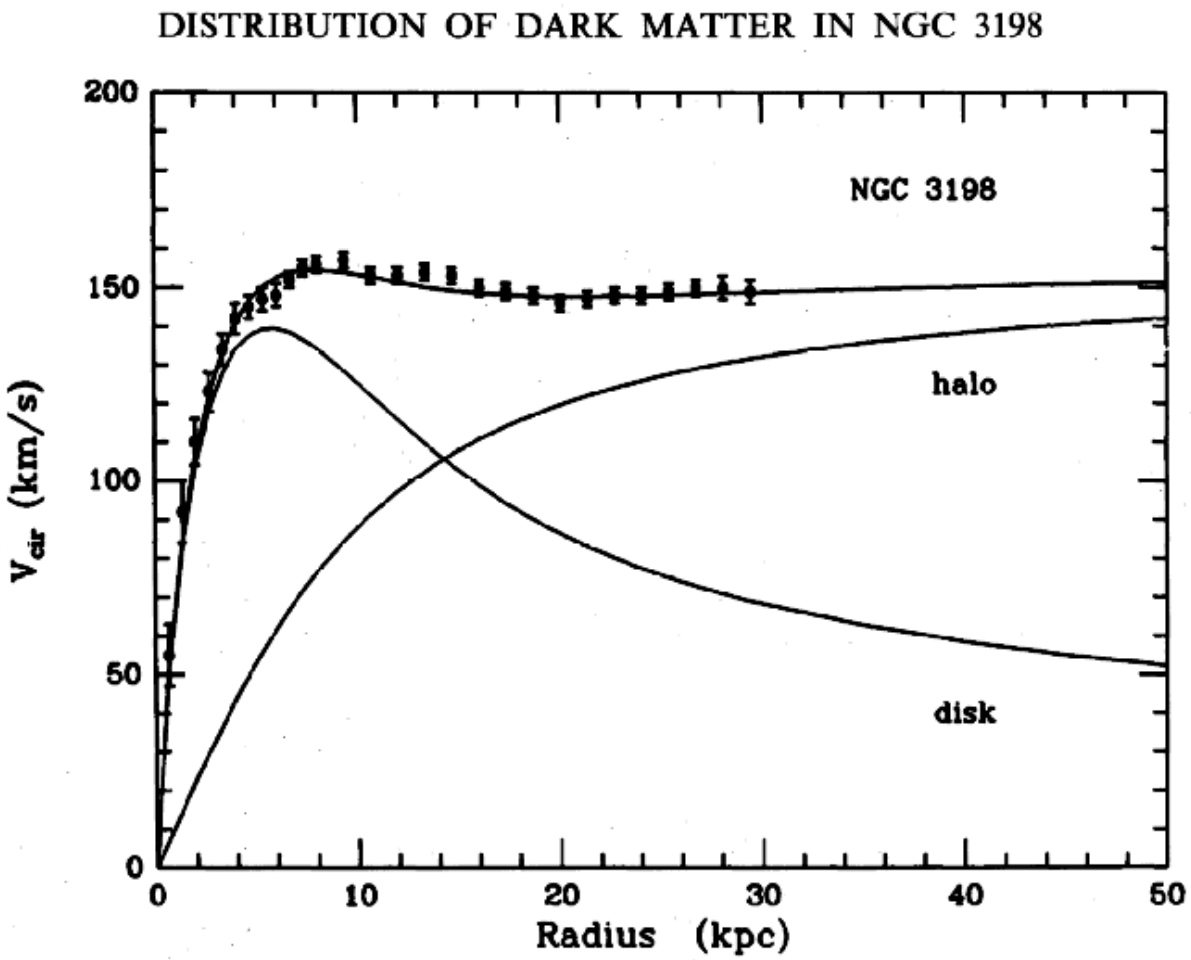
1 kpc = 0.308 10^{20} m = 3261 light years

Decomposition observed rotation curve of a spiral galaxy NGC 6503 into:
(i) gas disk, (ii) stellar disk, (iii) dark halo consisting of undiscovered physical particles



Sofue (2014)

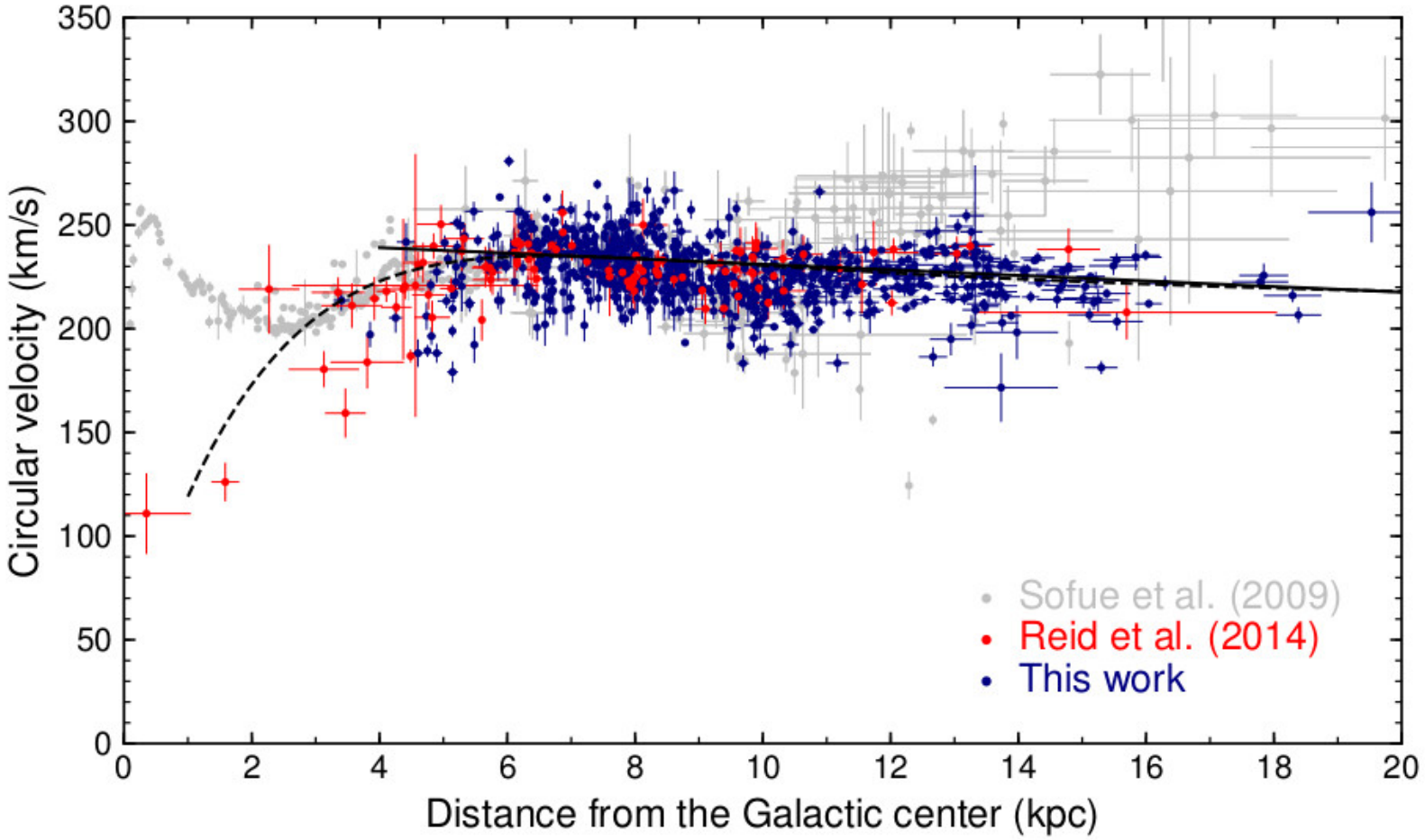
https://ned.ipac.caltech.edu/level5/Sept16/Sofue/frames.html



Two-component model of rotation of NGC 3198

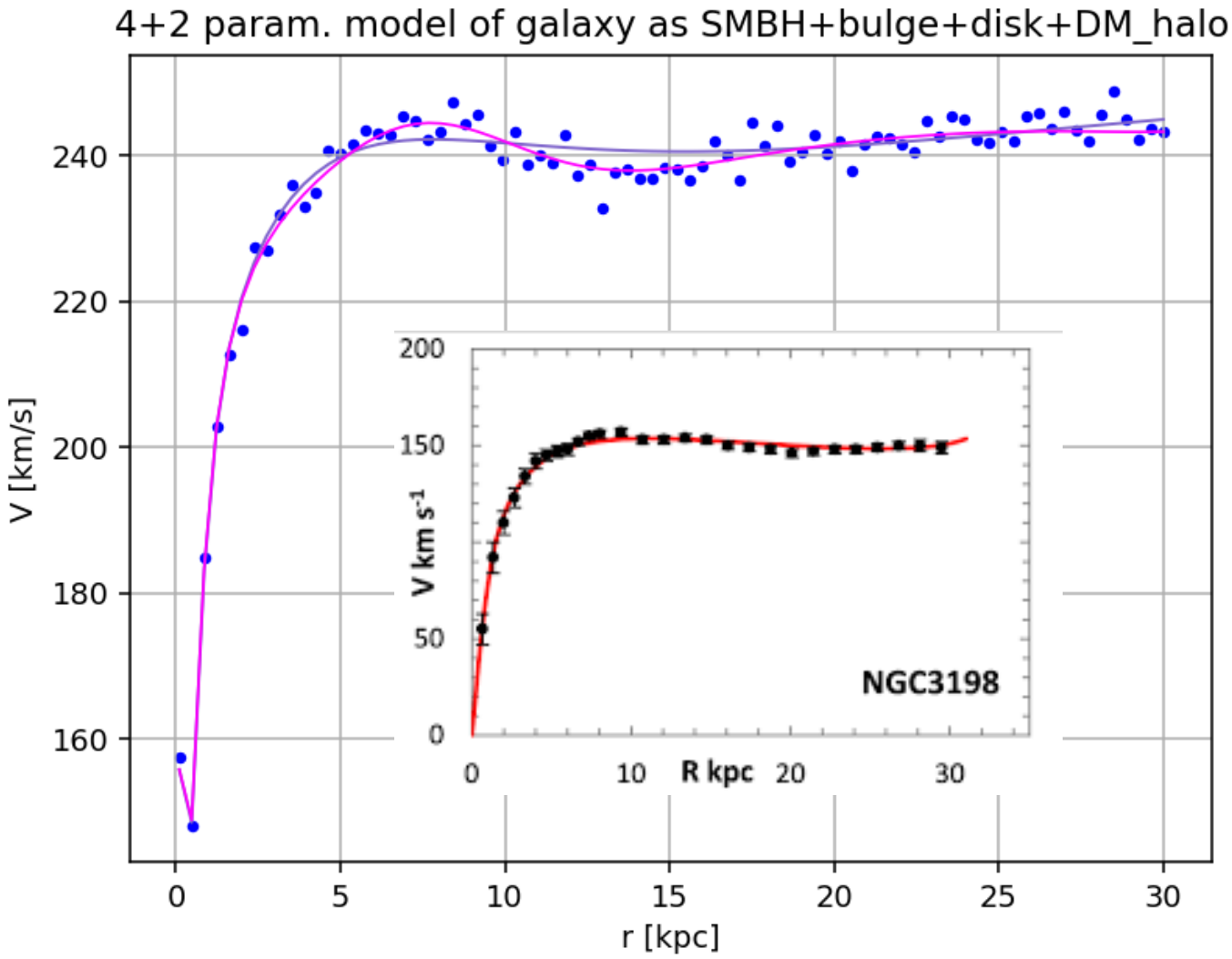
Rotation of galaxies and the measurements of Dark Matter

- Rotation curve of our Galaxy (Milky Way) shows similar flatness as other galaxies.



Four-component to fit synthetic rotation curve:
(i) central SMBH (supermassive black hole)

(ii) stellar bulge, (iii) stellar and gas disk, (iv) dark matter halo



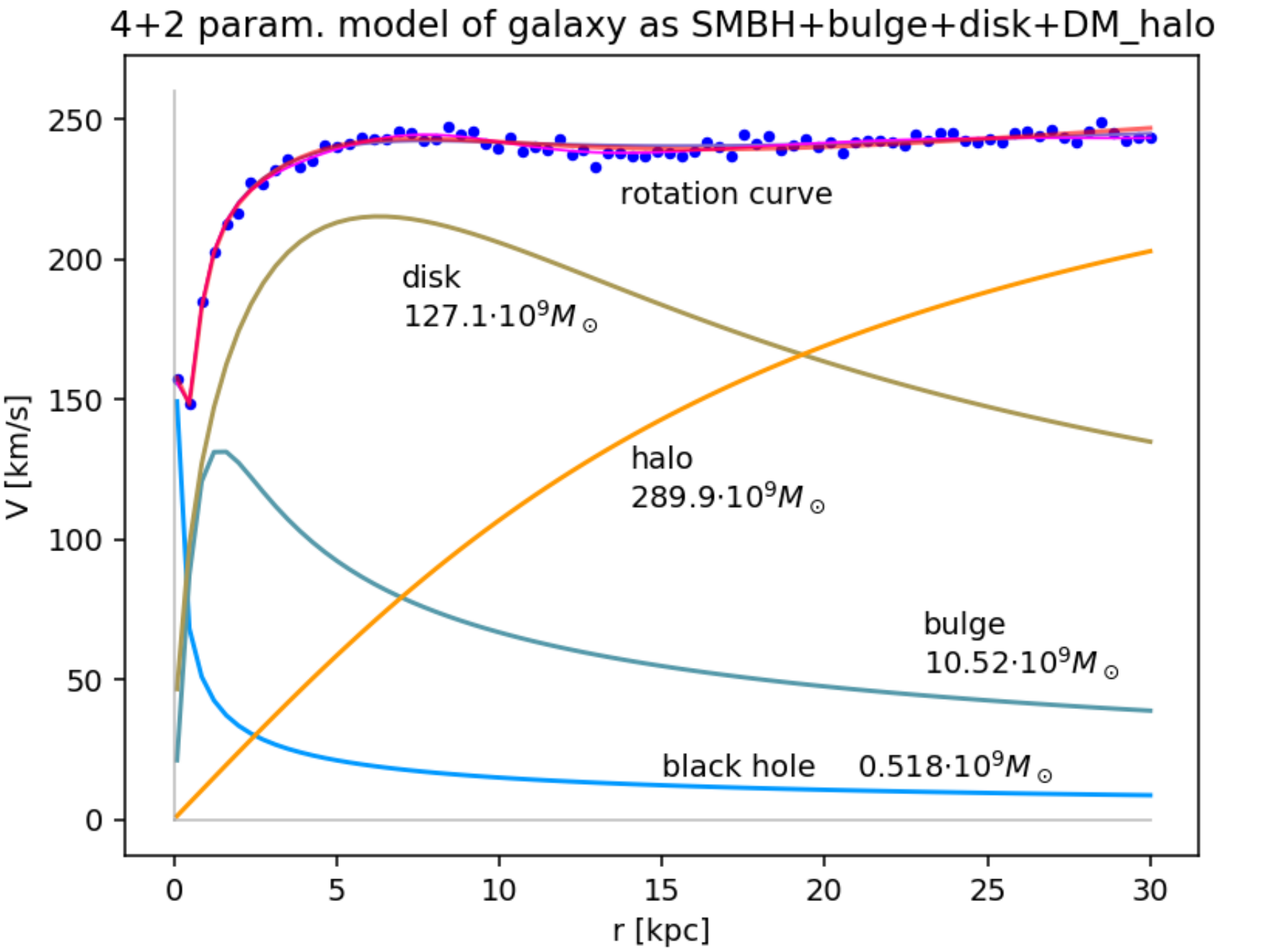
- Our least-squares function fitting so far computed very quickly
- They also worked well, in part because of the fact that we were finding only amplitudes of N additive functions
- Notice the red parameters below are not of this kind and could not directly be solved for by linear algebra system.
- $y(x) = (a + b\,x)^{10} + c\,x\cos(x) - d\exp(-x/(1+g)) + f\sin(kx)/(kx) + 2\log[x/(1+h^3)]$
- p=[a,b,c,d,f,h] – can all be found from one set of eqs. Ap = b.
- Q: Why is parameter h not marked in red?
- Q: why fitting

$$y(x) = a\exp[-(x-b)^2/c^2]$$ to M data points

can be done as easily as fitting a parabola to M points, but

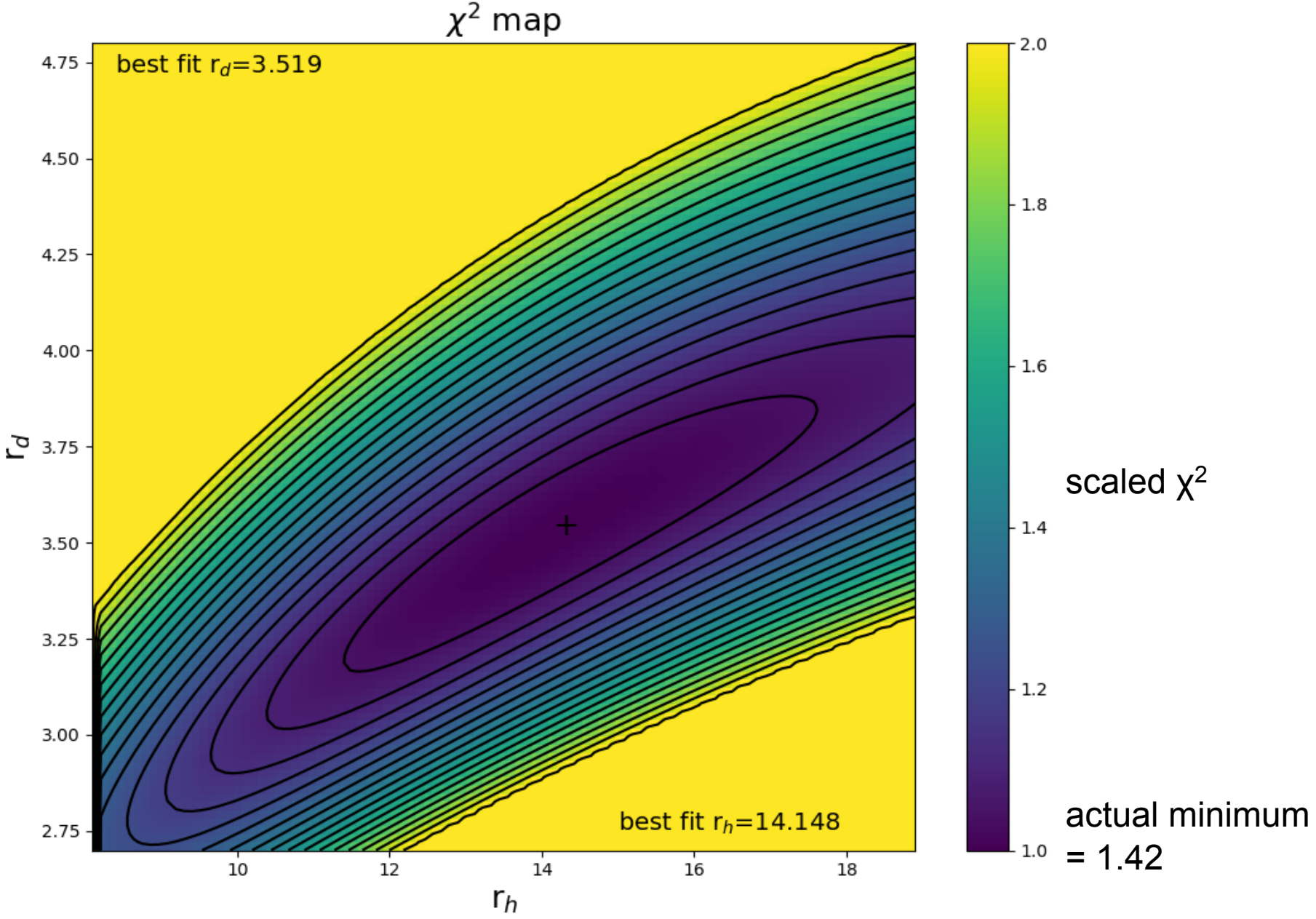
$$y(x) = a\exp[-(x-b)^2/c^2] + b\exp[-x^2/g^2]$$ cannot?

- In the galaxy modeling we first encounter those more difficult types of paramters.
- Least squares fitting with *arbitrary functions* and in *more dimensions*
- can be done, but requires solution of simultaneous nonlinear equations, for instance by coupled Newton's methods.
- this is the area of optimization theory, which is much wider than just the least squares method. Here we just use iterations to find r_d and r_h.


4+2 param. model of galaxy as SMBH+bulge+disk+DM_halo
rotation curve
disk
$127.1 \cdot 10^9 M_\odot$
halo
$289.9 \cdot 10^9 M_\odot$
bulge
$10.52 \cdot 10^9 M_\odot$
black hole
$0.518 \cdot 10^9 M_\odot$
V [km/s]
r [kpc]
0
50
100
150
200
250
0
5
10
15
20
25
30

fit-glx-4+2b.py

$\chi^2 = \Sigma_i\ (y-Y)^2_i/\sigma^2_i$; y=model, Y=data, σ=obs. std



Newton's method in multiple dimensions

See Turner et al (2018) chapter 5.6, p. 173

Our textbook explains how in higher dimensions instead of derivative
dF/dx of a function F: R→ R

we have a function that can be N-dimensional (think of a gradient in N dimensions, which is an N-dim vector). F: $R^N \rightarrow R^N$
and derivatives are forming N x N matrix called Jacobian matrix **J**

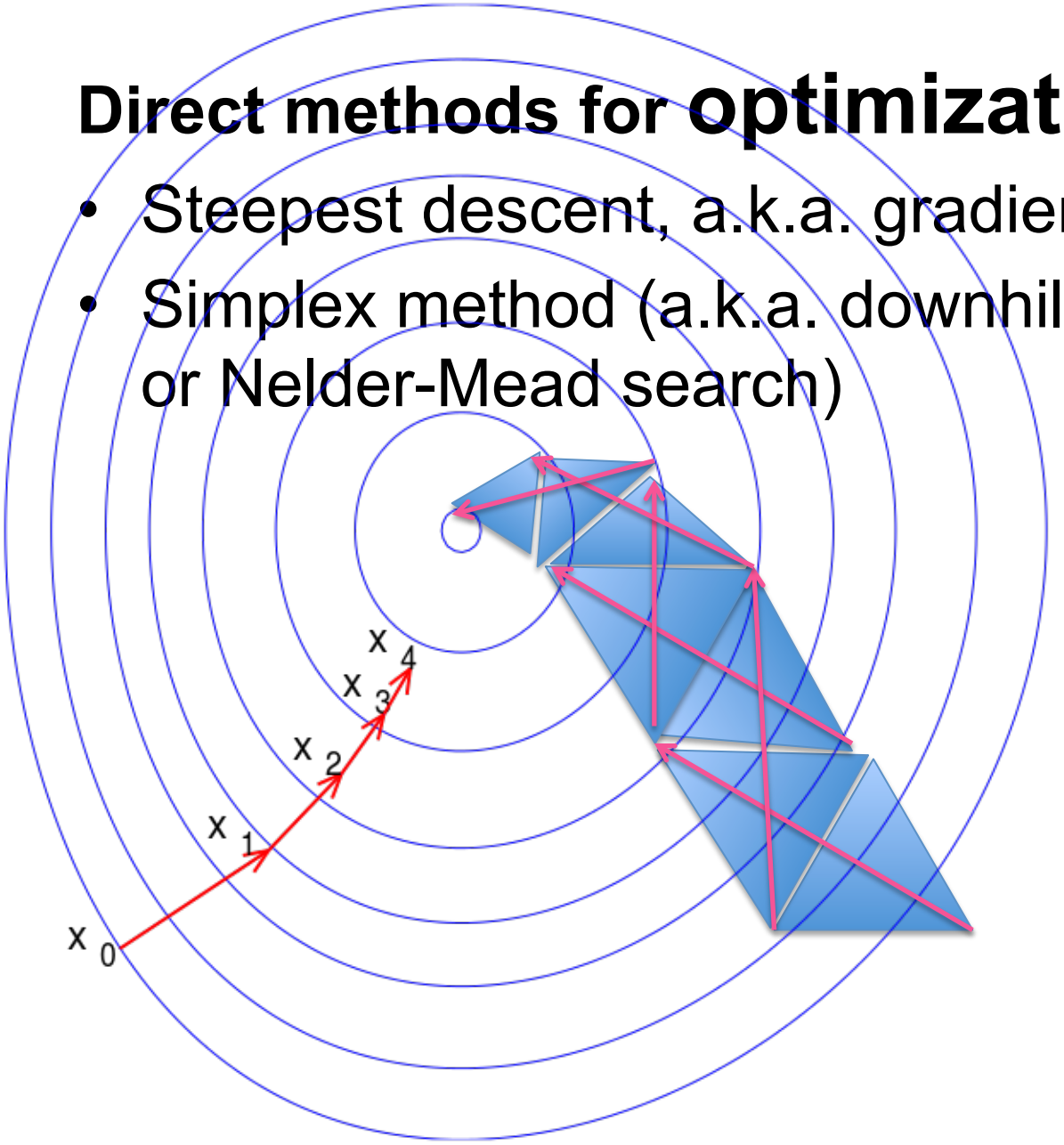
$J_{nm} = dF_n/dx_m$

Taylor expansion now reads $F(x+h) = F(x) + \mathbf{J}\, h + \ldots$

and the algorithm iterates (vectors) as: $x_{new} = x - \mathbf{J}^{-1} F$ where old (current) values are used on the right-hand side.
In minimization problems, we search for a zero of vector function F = grad f, where f is the scalar function we need to minimize/maximize. Newton's algorithms thus use inverse of Jacobian of F, which really is a Hessian (symmetric second derivative matrix of a scalar function f): : $x_{new} = x - \mathbf{H}^{-1} f$, where
$H_{nm} = d^2f/dx_n dx_m = d(df/dx_n)/dx_m$.
The search may be slowed down, if needed, by introduction of a constant in front of the inverse Hessian, which will prevent overshooting the minimum in one step.

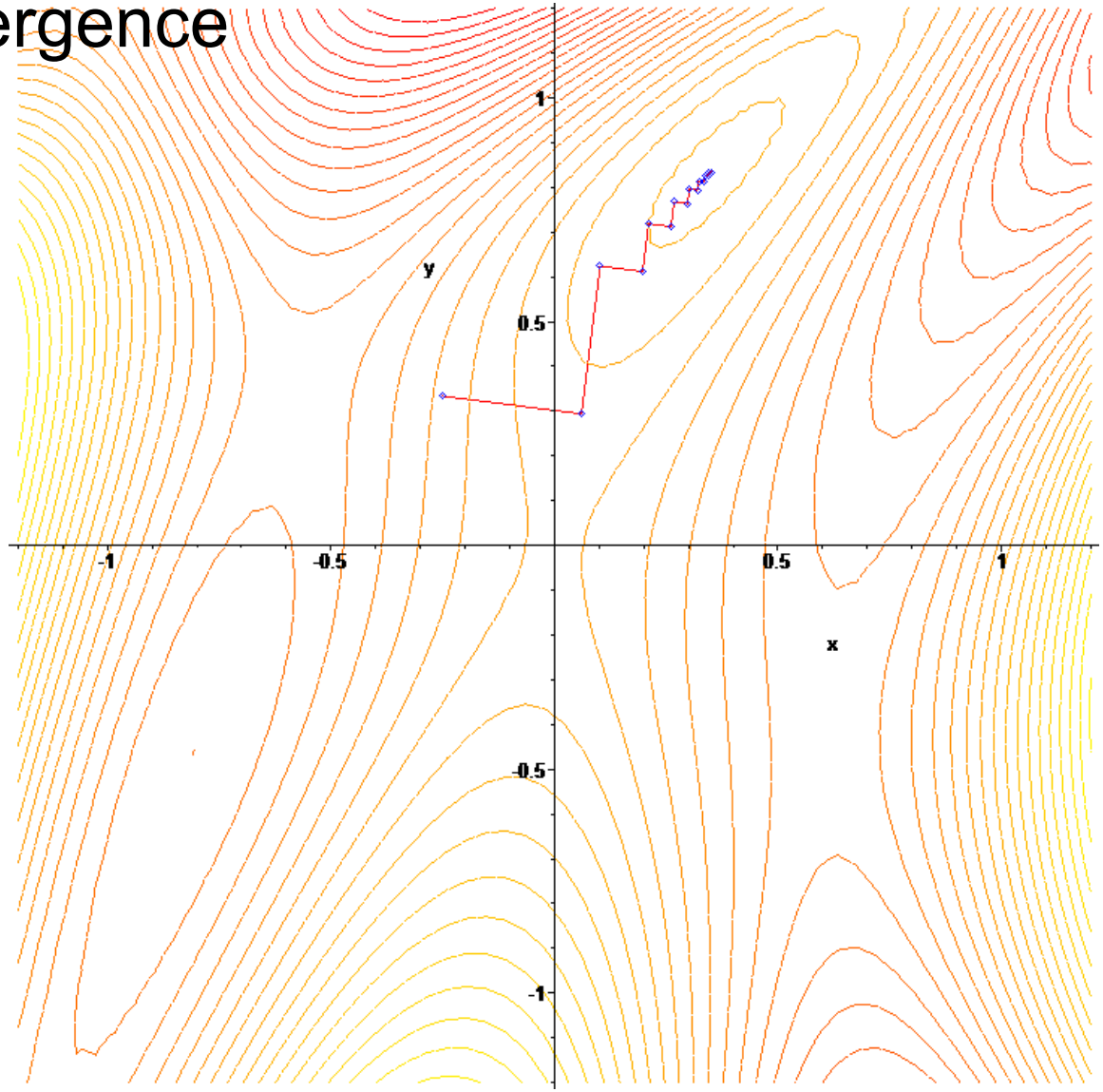
Direct methods for optimization in N dim.

- Steepest descent, a.k.a. gradient search method
- Simplex method (a.k.a. downhill simplex, simplicial or Nelder-Mead search)



Steepest descent method

- Problematic cases: zigzagging, slow asymptotic convergence



Nelder-Mead method or simplicial search a.k.a. polytope method

https://docs.scipy.org/doc/scipy/reference/optimize.minimize-neldermead.html#optimize-minimize-neldermead

describes the scipy implementation of Simplex (sometimes called downhill simplex) method of Nelder and Mead.

Nelder, J A, and R Mead. 1965. A Simplex Method for Function Minimization. The Computer Journal 7: 308-13.

OTHER METHODS available in SciPy:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize

describes about 10 different methods for minimization of scalar functions in N-dimensional space, including:

- CG methods (conjugate gradient methods) – PHYD57 will analyze this
- Newton-CG methods
- etc.

Interpolation

- Turner et al (2018) – read chapter 6, starting with p. 189

- Interpolation by polynomials: **Lagrange polynomials**
  - problem: wiggly polynomials
  - the trouble also explains why extrapolation is difficult
- Interpolation by piecewise polynomials:
  - **Splines**; much nicer than polynomials in general
  - cubic splines have continuous 0^{th}, 1^{st} and 2^{nd} deriv.
  - they require solving a tri-diagonal linear system for constants of piecewise cubic functions.
  - this is a cheap calculation, only O(N) arithm. operations
  - `f = scipy.interpolate.interp1d(X,Y,'cubic')`
  - offers several different interpolation schemes depending on string argument, input: X and Y (numpy arrays of length N). Result is a *function* that you can call with some other array of x's to interpolate & plot:
  - `plt.plot(x,f(x));  plt.show()`