Lecture 9 Review

Poisson and Gaussian probability distributions.

Pick random numbers from arbitrary probability distribution.

First peek at fitting data.

Numerical Derivatives

We need to know to to take a derivative df(x)/dx of a function f(x) at x.

$$f'(x) \equiv \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$ (from calculus land)

Taylor series expand f(x + h). Recall that h << 1:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \ldots$$

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\zeta)$$ (exact, also from calculus)

ζ unknown !! ($x \leq \zeta \leq x + h$)

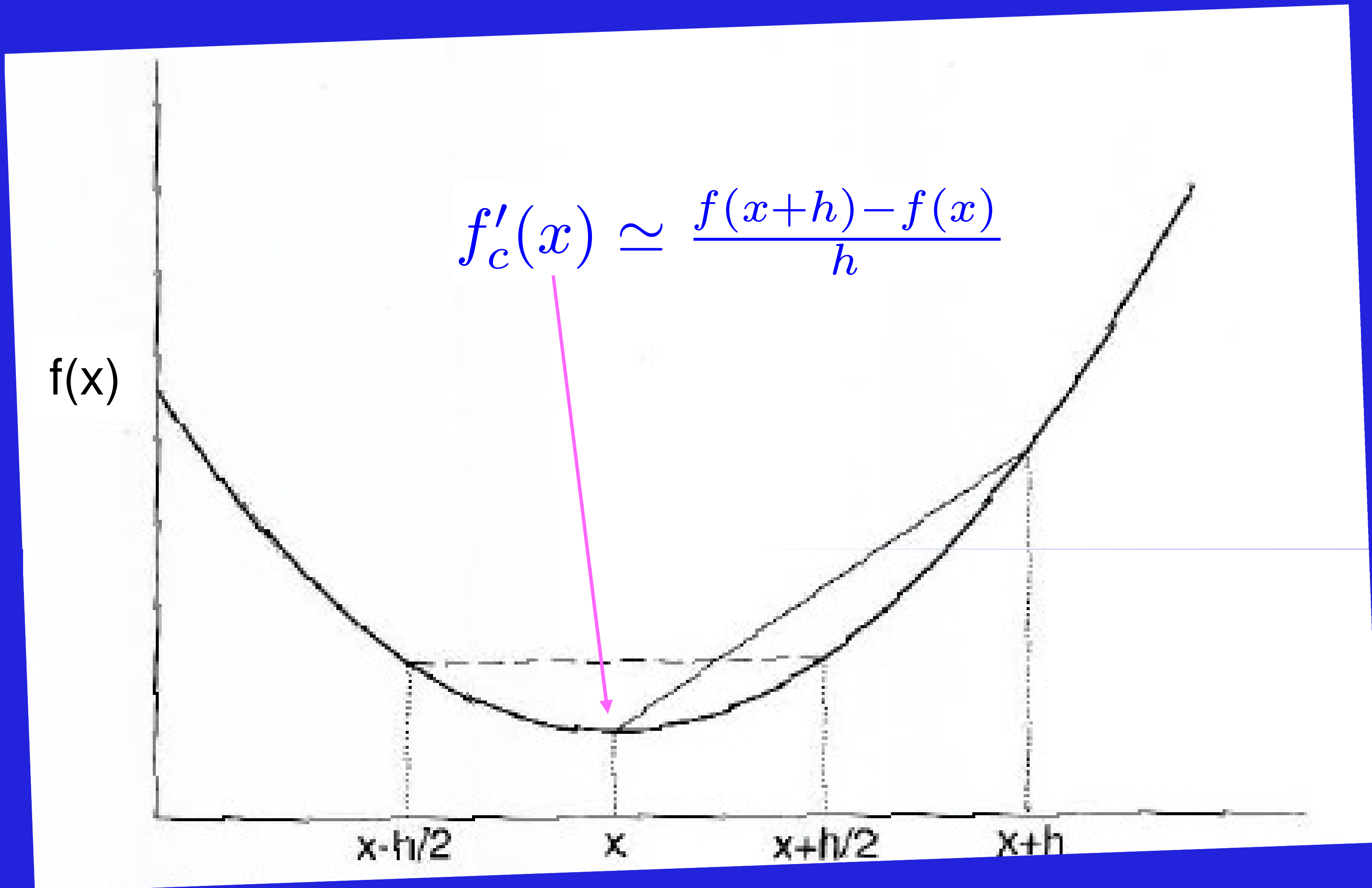
$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(\zeta)$$

"truncation error"

$$f'_c(x) \simeq \frac{f(x+h) - f(x)}{h}$$

"forward derivative"

Forward Derivative



$$f'_c(x) \simeq \frac{f(x+h)-f(x)}{h}$$

Not too bad (truncation error $\propto$ h), but we can do better.

Central Difference Derivative

Use alternative, but entirely equivalent, definition of df(x)/dx at x.

$$f'(x) \equiv \lim_{h\to 0} \frac{f(x+h/2)-f(x-h/2)}{h}$$

Taylor series expand f(x + h/2):

$$f(x+h/2) = f(x) + \frac{h}{2} f'(x) + \frac{h^2}{8} f''(x) + \frac{h^3}{48} f'''(x) + \ldots$$

and f(x - h/2):

$$f(x-h/2) = f(x) - \frac{h}{2} f'(x) + \frac{h^2}{8} f''(x) - \frac{h^3}{48} f'''(x) + \ldots$$

$$f'(x) = \frac{f(x+h/2)-f(x-h/2)}{h} - \frac{h^2}{24} f'''(\zeta)$$

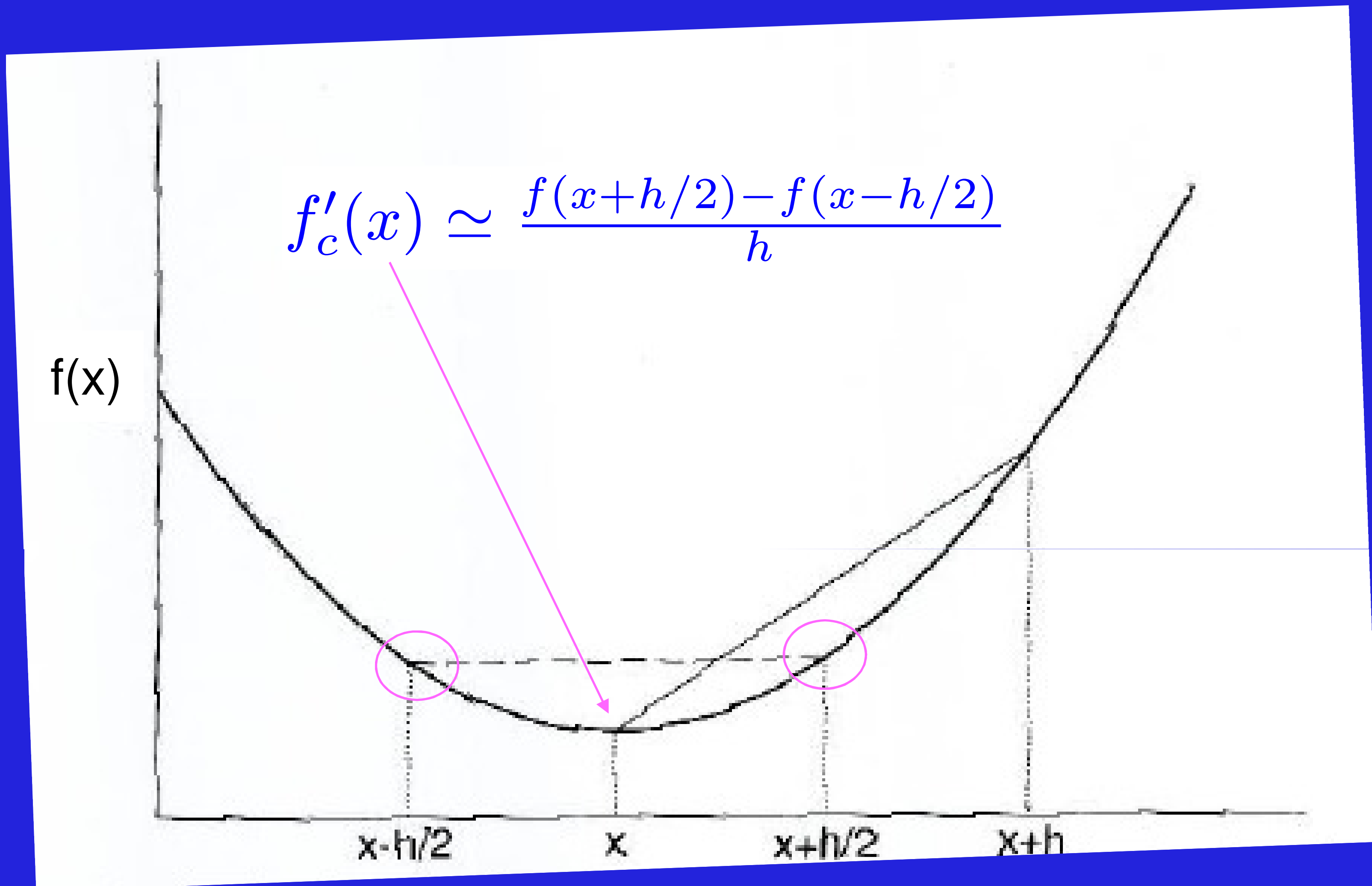
$$f'_c(x) \simeq \frac{f(x+h/2)-f(x-h/2)}{h}$$

Our workhorse derivative

Truncation error $\propto h^2$

"central difference derivative"

Central Difference Derivative (2)



Again, our workhorse (truncation error $\propto h^2$)

GSL Routine (see derivative.cc)

```
#include <iostream>
#include <iomanip>
#include <gsl/gsl_math.h>
#include <gsl/gsl_deriv.h>

using namespace std;

double f (double x, void * params)
{
  return pow (x, 1.5);   // <----- CHANGE ME
}

int main ()
{
  gsl_function F;
  double result, abserr;

  F.function = &f; // no touch
  F.params = 0;    // no touch

cout << "f(x) = x^(3/2)" << endl;

 gsl_deriv_central (&F, 2.0, 1e-8,
&result, &abserr);
  cout << "x = 2.0" << endl;
  cout << "f'(x) = " <<
setprecision(11)<< result << " +/- "
<< abserr << endl;
  cout << "exact = " <<
setprecision(11) << 1.5 * sqrt(2.0)
<< endl << endl;

  gsl_deriv_forward (&F, 0.0, 1e-8,
&result, &abserr);
  printf ("x = 0.0\n");
  printf ("f'(x) = %.10f +/- %.10f\n",
result, abserr);
  printf ("exact = %.10f\n", 0.0);

  return 0;
}
```

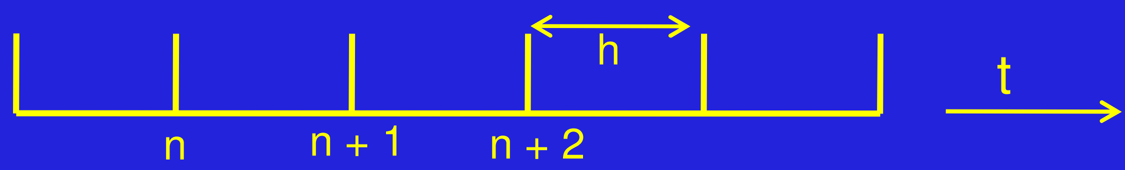
Runge-Kutta (2nd order)

We need a technique to solve ordinary differential equations (ODEs)

An ODE has one independent variable.

Formally, $\frac{dy}{dt} = f(t,y) \Rightarrow y(t) = \int f(t,y)dt$

Discretizing, $y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t,y)dt$



Approximately, $f(t,y) \simeq f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}}) + (t - t_{n+\frac{1}{2}})\frac{df}{dt}(t_{n+\frac{1}{2}}) + \mathcal{O}(h^2)$

Substitute last equation into 2nd and take note,

$$\int_{t_n}^{t_{n+1}} (t - t_{n+\frac{1}{2}})dt = \left.\frac{(t-t_{n+\frac{1}{2}})^2}{2}\right|_{t_n}^{t_{n+1}} = 0$$

$$\int_{t_n}^{t_{n+1}} f(t,y) \simeq f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})h + \mathcal{O}(h^3)$$

$$\Rightarrow y_{n+1} \simeq +y_n + f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})h + \mathcal{O}(h^3)$$

So far, so good but …

2^{nd} order RK (2)

$$y_{n+1} \simeq +y_n + f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})h + \mathcal{O}(h^3)$$

… we don't know what $y_{n+1/2}$ is.

We only calculate quantities at t_n, t_{n+1}, … to get $f(t_n, y_n)$.

Dead end?

Noooo. Approximation to the rescue.

$$\begin{aligned} y_{n+\frac{1}{2}} &\simeq y_n + \frac{dy}{dt}\frac{h}{2} \\ &\simeq y_n + \frac{1}{2}hf(t_n, y_n) \end{aligned}$$

$$\begin{aligned} \vec{y}_{n+1} &\simeq y_n + \vec{k}_2 \\ \vec{k}_2 &= h\vec{f}(t_n + \tfrac{h}{2}, \vec{y}_n + \vec{k}_1/2) \\ \vec{k}_1 &= h\vec{f}(t_n, \vec{y}_n) \end{aligned}$$

"2^{nd} order Runge - Kutta" algorithm for ODE solution.

Looks complicated. Looks can be deceiving.

Solving ODEs

Write ODE in standard format.

$$\frac{d\vec{y}(t)}{dt} = \vec{f}(t, \vec{y})$$

$\vec{y}$ and $\vec{f}$ are N-dimensional vectors.

The idea is to express any order ODE as N simultaneous 1st order ODEs

$$\vec{y} = \begin{pmatrix} y^{(0)}(t) \\ y^{(1)}(t) \\ \vdots \\ y^{(N-1)}(t) \end{pmatrix} \quad \vec{f} = \begin{pmatrix} f^{(0)}(t, \vec{y}) \\ f^{(1)}(t, \vec{y}) \\ \vdots \\ f^{(N-1)}(t, \vec{y}) \end{pmatrix}$$

$$\frac{dy^{(0)}(t)}{dt} = f^{(0)}(t, \vec{y})$$

$$\frac{dy^{(1)}(t)}{dt} = f^{(1)}(t, \vec{y})$$

No y-derivatives

Solving ODEs (2)

Consider Newton's 2nd law in 1 dimension.

$$d^2x/dt^2 = F(t, x, dx/dt)$$

Write in standard form.

1st step ⟶

$$y^{(0)}(t) \equiv x(t)$$

$$dx/dt = dy^{(0)}/dt \equiv y^{(1)}$$

⟵ 2nd step

So,

$$dy^{(0)}/dt \equiv y^{(1)}$$

$$dy^{(1)}/dt = F(t, y^{(0)}, y^{(1)})$$

Equations in vector form

$$f^{(0)} = y^{(1)}(t)$$

$$f^{(1)} = F(t, y^{(0)}, y^{(1)})$$

Example Solution of an ODE

Hooke's law: $m\ d^2x/dt^2 = -kx$

$$d^2x/dt^2 = -\frac{k}{m}x$$

1st step → $y^{(0)}(t) = x(t)$

2nd step → $\frac{dy^{(0)}}{dt} = f^{(0)} = y^{(1)}$

$$\frac{d^2x}{dt^2} = \frac{d}{dt}\left(\frac{dy^{(0)}}{dt}\right) = -\frac{k}{m}x$$

$$\frac{d}{dt}y^{(1)}(t) = -\frac{k}{m}x$$

$$f^{(1)} = -\frac{k}{m}x$$

$$\frac{d\vec{y}(t)}{dt} = \vec{f}(t,\vec{y})$$

$$\frac{dy^{(0)}}{dt} = y^{(1)}$$

$$\frac{dy^{(1)}}{dt} = -\frac{k}{m}y^{(0)}$$

Preferred Algorithm for ODE Solution

4th-order Runge-Kutta algorithm

$$
\begin{aligned}
\vec{y}_{n+1} &\simeq y_n + \tfrac{1}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) \\
\vec{k}_1 &= h\vec{f}(t_n, \vec{y}_n) \\
\vec{k}_2 &= h\vec{f}(t_n + \tfrac{h}{2}, \vec{y}_n + \tfrac{\vec{k}_1}{2}) \\
\vec{k}_3 &= h\vec{f}(t_n + \tfrac{h}{2}, \vec{y}_n + \tfrac{\vec{k}_2}{2}) \\
\vec{k}_4 &= h\vec{f}(t_n + h, \vec{y}_n + \vec{k}_3)
\end{aligned}
$$

term of $\mathcal{O}(h^4)$ neglected

t is independent variable

h is step size in t, $h < 1$

YIKES !!

Summary

Numerical derivatives: forward and central difference.

ODE solver: 4th order Runge-Kutta

Don’t suffer in silence. Scream for help!!!