Week 2

Numerical Integration

Finite differences

- Taylor expansion:
- $f(x_0+h) = f(x_0)+f'(x_0)\, h + f''(x_0)\, h^2/2! + \ldots$
- $f(x_0+h) - f(x_0) = f'(x_0)\, h + f''(x_0)\, h^2/2! + \ldots$
- $[f(x_0+h) - f(x_0)]/h = f'(x_0) + f''(x_0)\, h/2! + \ldots$

- Approximation good to O(h)

Finite differences

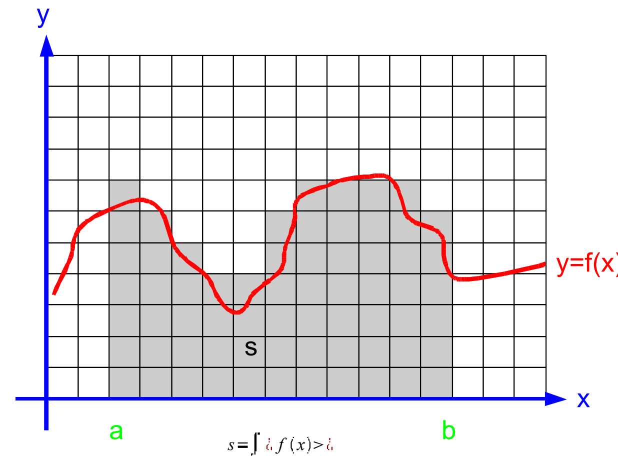
- Taylor expansion: MIDPOINT
- $f(x_0 + h/2) = f(x_0) + f'(x_0)\, h/2 + f''(x_0)\, (h/2)^2/2! + \ldots$
- $f(x_0 - h/2) = f(x_0) + f'(x_0)\, (-h/2) + f''(x_0)\, (-h/2)^2/2! + \ldots$
- $f(x_0 + h/2) - f(x_0 - h/2) = f'(x_0)h + f'''(x_0)\, (h/2)^3\, 2/3! + \ldots$
- $[f(x_0 + h/2) - f(x_0 - h/2)]/h = f'(x_0) + f'''(x_0)\, (h/2)^2\, /3!$
- Approximation good to $O(h^2)$

Numerical Integration

- Finding the area under a curve

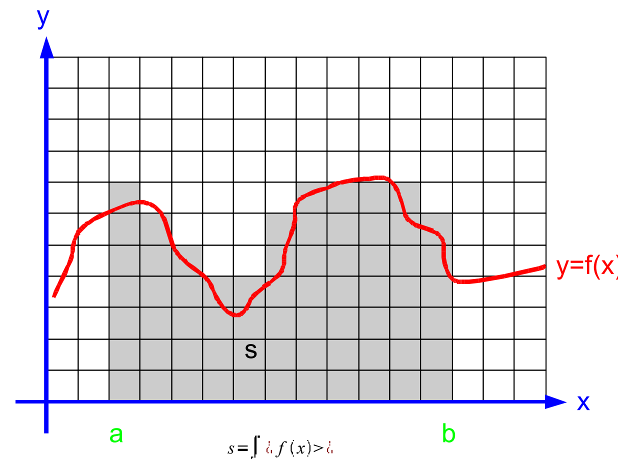
- An alternative to analytical solutions (i.e. doing the maths)

  – When a formula can' t be symbolically integrated
  – When it is computationally cheaper to evaluate numerically than analytically
  – When a formula isn't available – only numerical data



$$s = \int_{x0}^{x1} f(x).dx$$

Numerical Integration



- Finding the area under a curve
- An alternative to analytical solutions (i.e. doing the maths)
- Simplest: N panels each width h
- Approximate as rectangle

$$s = \int_a^b f(x).dx = \sum_{n=0}^{N-1} \int_{xn}^{xn+h} f(x).dx$$

$$\approx \sum_{n=0}^{N-1} [f(x_n) + f'(x_n)h + f''(x_n)h^2/2 + \ldots]h$$

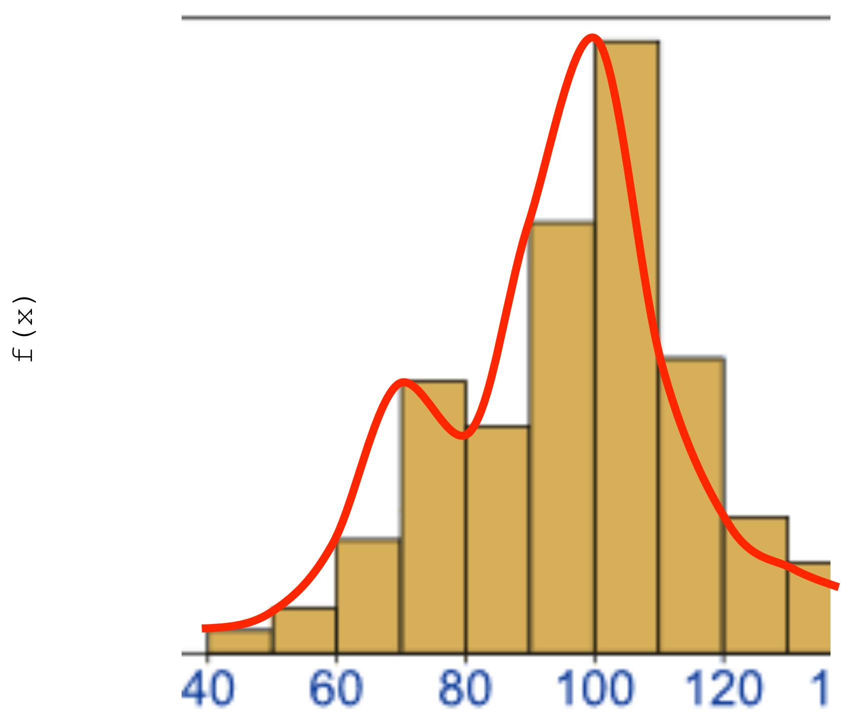
$$\approx \sum_{n=0}^{N-1} f(x_n)h + O(h^2)$$

Rectangles

$$\approx \sum_{n=0}^{N-1} f(x_n)h + O(h^2)$$

- Divide the function into a series of rectangular panels

- This is the simplest way. Height of the rectangle set by function value at start (left point)

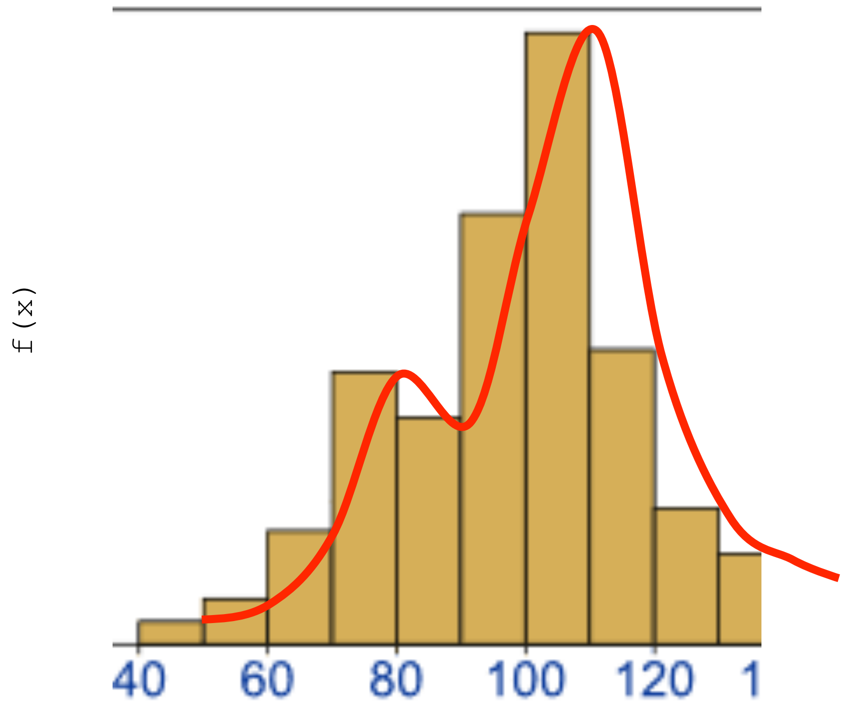


Rectangles

$$\approx \sum_{n=1}^{N} f(x_n)h + O(h^2)$$

- Divide the function into a series of rectangular panels

- This is the simplest way. Height of the rectangle set by function value at right hand point



Rectangles – MIDPOINT

$$s = \int_{x0}^{x1} f(x).dx$$

$$= \sum_{n=1}^{N-1} \int_{xn-h/2}^{xn+h/2} f(x).dx$$

$$= \sum_{n=1}^{N-1} \int_{xn-h/2}^{xn} f(x).dx + \sum_{n=1}^{N-1} \int_{xn}^{xn+h/2} f(x).dx$$

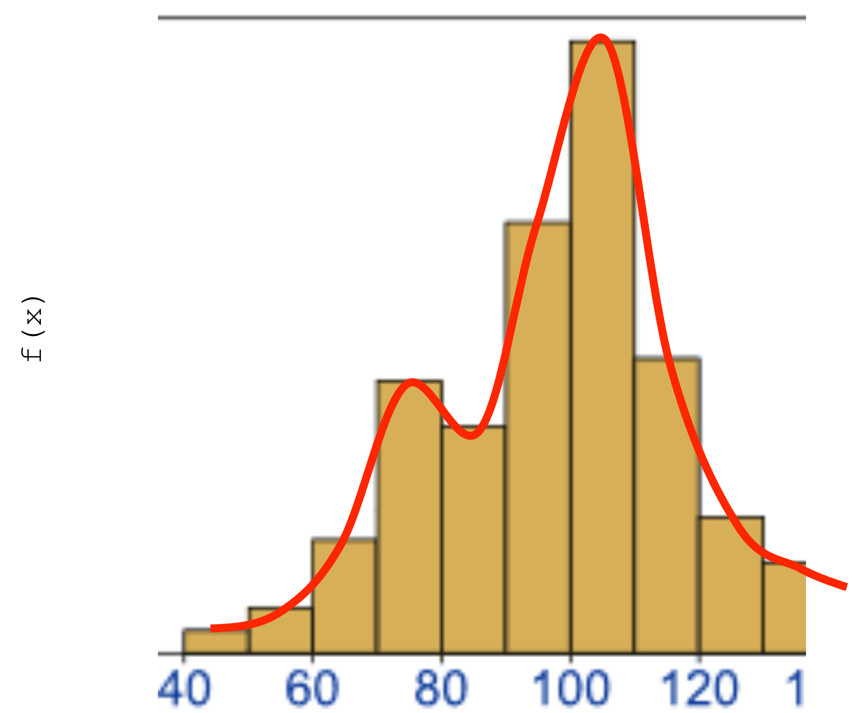
$$\approx \sum_{n=1}^{N-1} [f(x_n) + f'(x_n)(-h/2) + f''(x_n)(-h/2)^2/2 + \ldots]h/2 +$$

$$+ [f(x_n) + f'(x_n)(h/2) + f''(x_n)(h/2)^2/2 + \ldots]h/2$$

$$= \sum_{n=1}^{N-1} [f(x_n) + f''(x_n)(h/2)^2/2 + \ldots]h \approx \sum_{n=1}^{N-1} f(x_n)\,h + O(h^3)$$

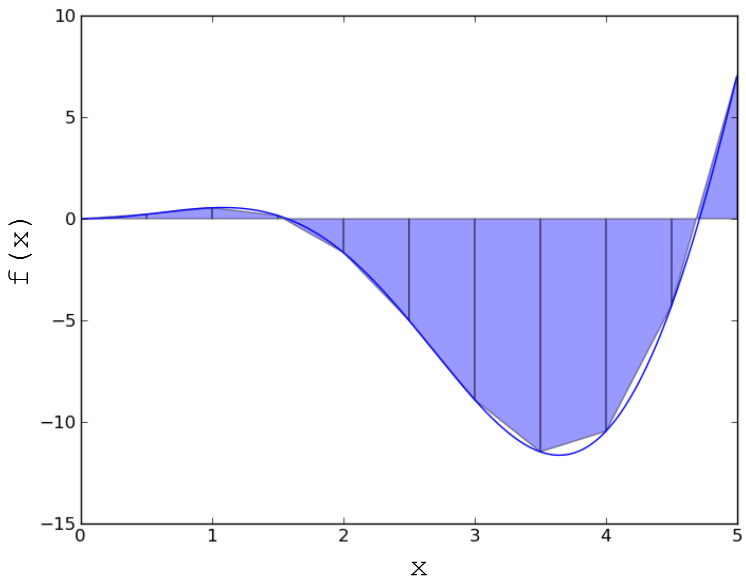
Rectangles

- Divide the function into a series of rectangular panels
- This is the simplest way
- And with midpoint its same number of calculations but better accuracy!



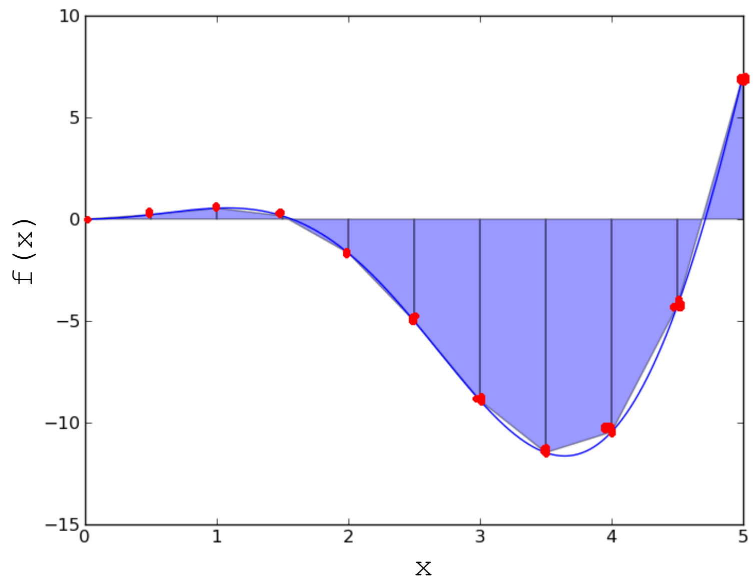
Trapezium Rule

- Instead of rectangles, use trapeziums
- Ie use first order derivative information



Computational Cost

- 2 function evaluations per panel
- But edges are shared
  - 1 per panel + 1
- More accurate than rectangles for no extra function evaluations



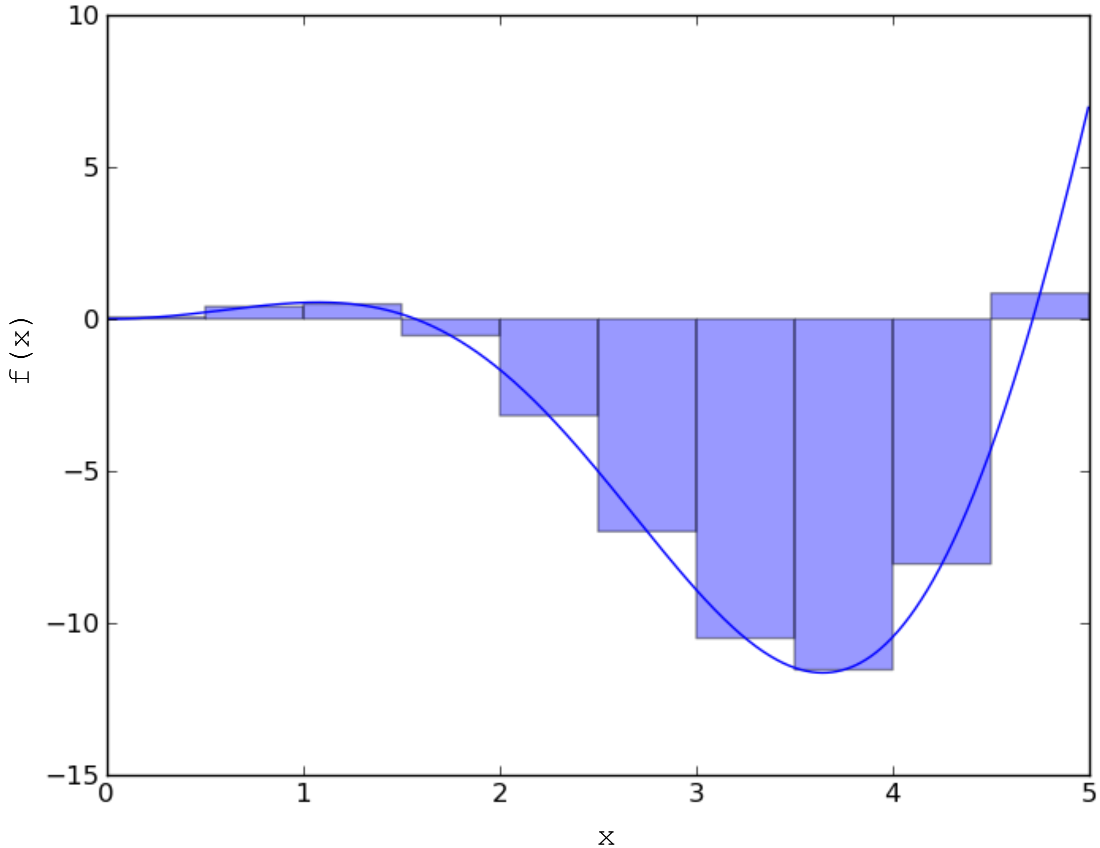
What’s going on?

- We are fitting analytical expressions to each panel of our function
- N^{th} order Lagrange polynomial expansions
- We then analytically integrate these small chunks

Rule	Expression
Rectangle	$y = k_0$
Trapezium	$y = k_0 + k_1x$

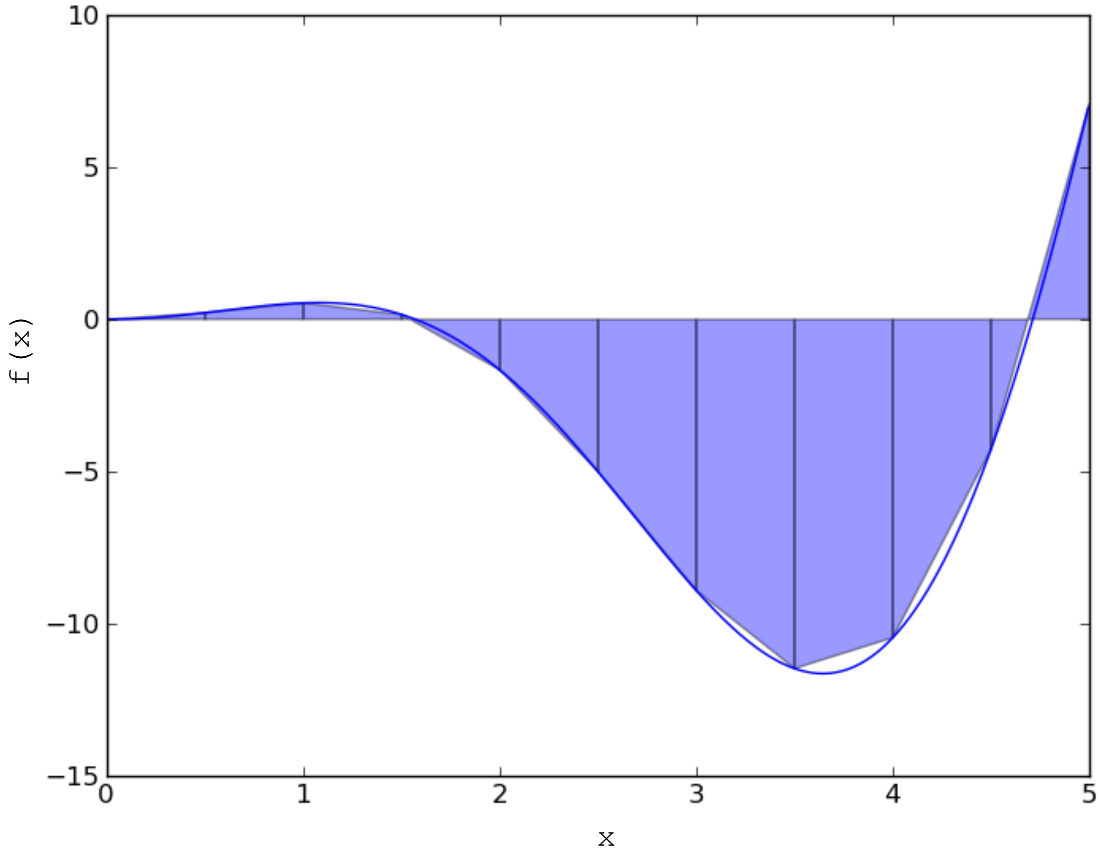
Midpoint rule (rectangle)

Panel area : y = $k_0 + k_1x + k_2x^2 + k_3x^3$



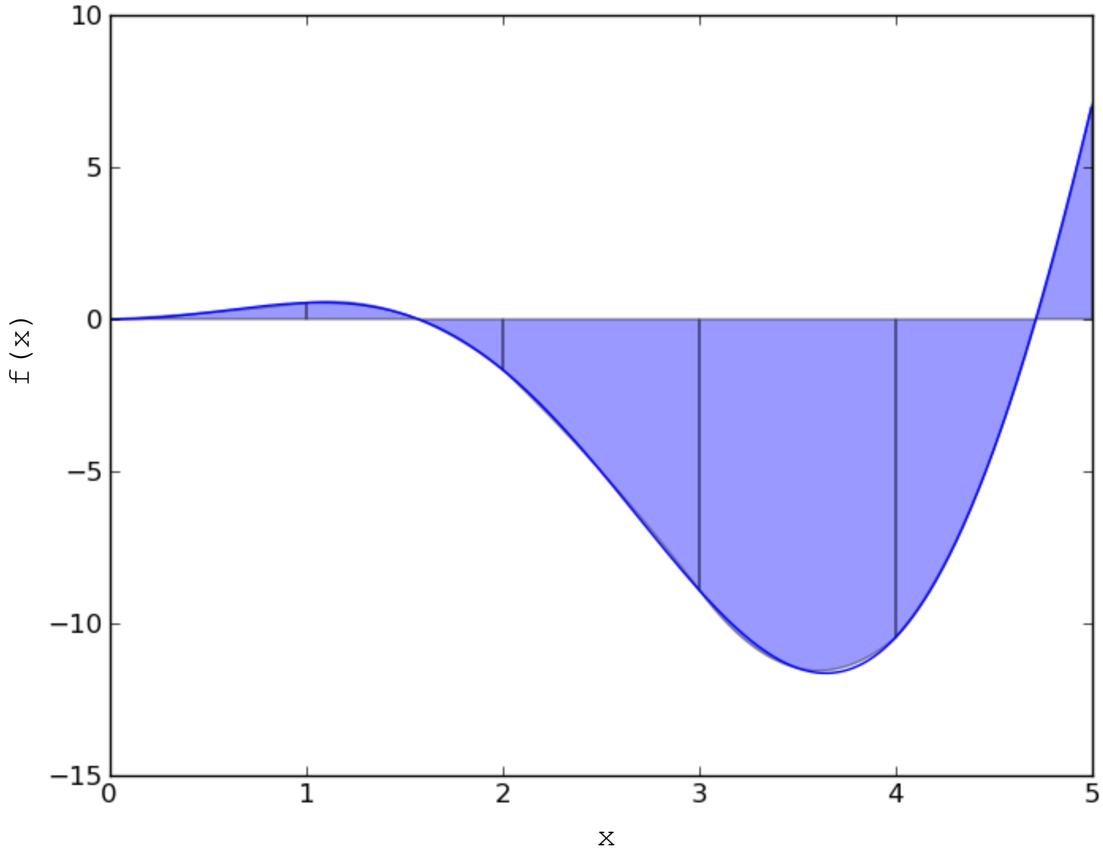
Trapezium Rule (GCSE!)

Panel area : y = $k_0 + k_1x + k_2x^2 + k_3x^3$



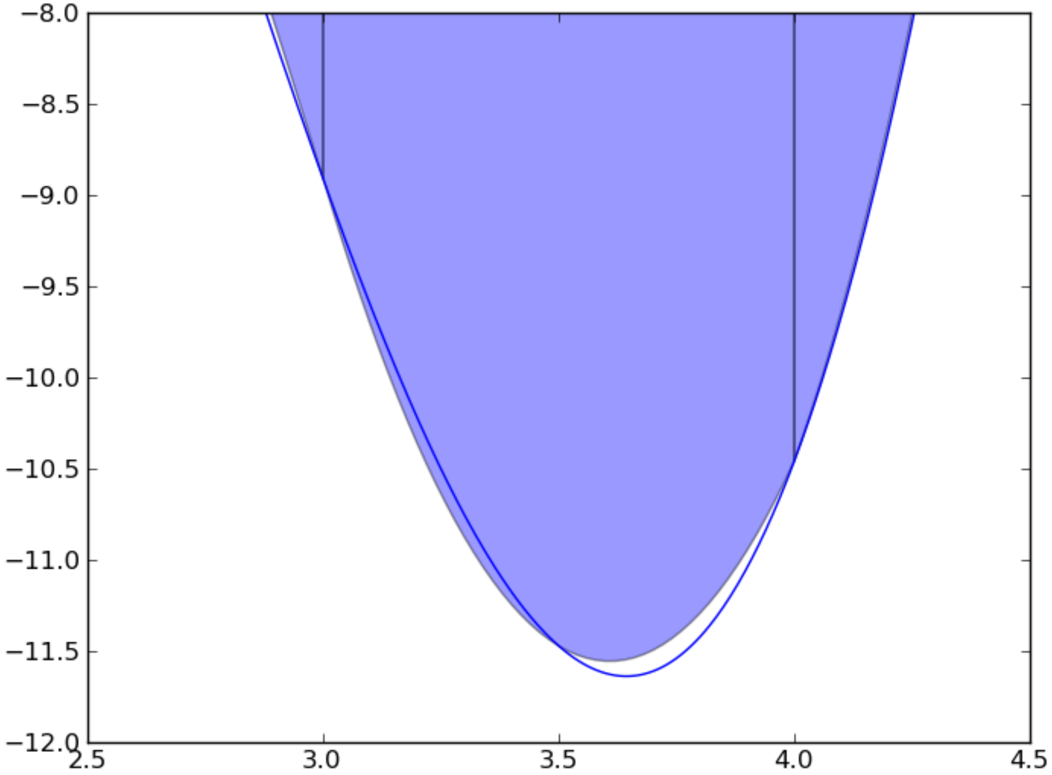
Simpson's Rule

Panel area : y = $k_0 + k_1x + k_2x^2 + k_3x^3$

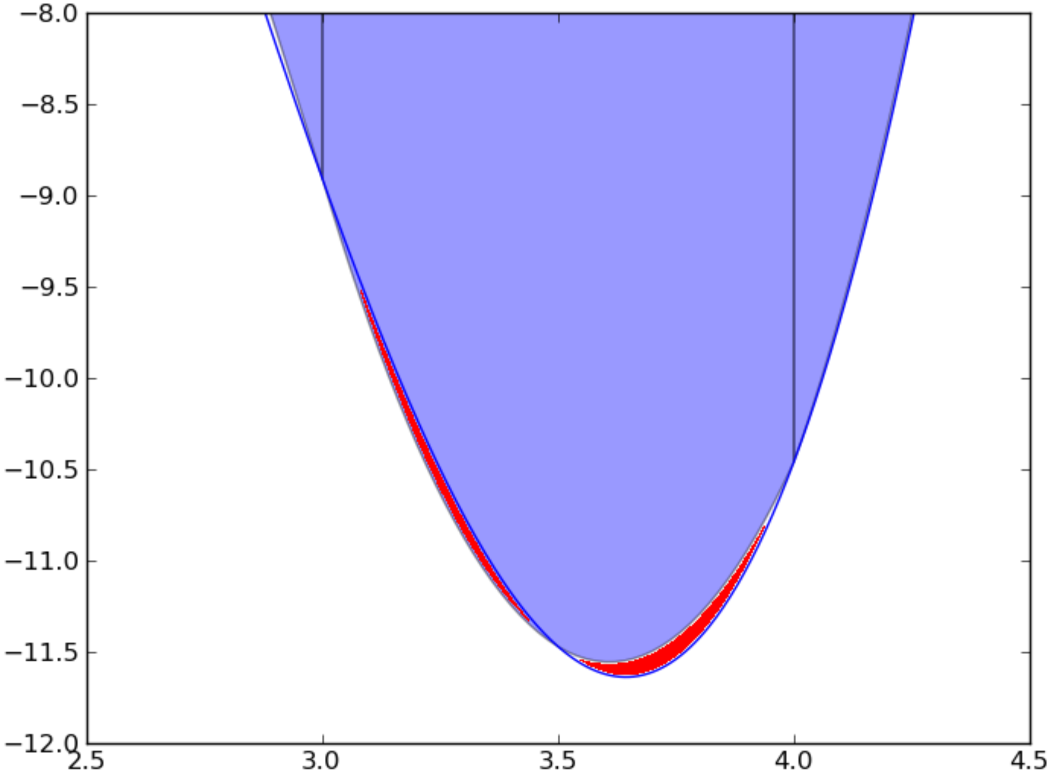


Simpson's Rule

Panel area : y = $k_0 + k_1x + k_2x^2 + k_3x^3$



Simpson's Rule



Simpson’s Rule - formula

- Use quadratic information – second derivative

- Panel $a <= x <= b$

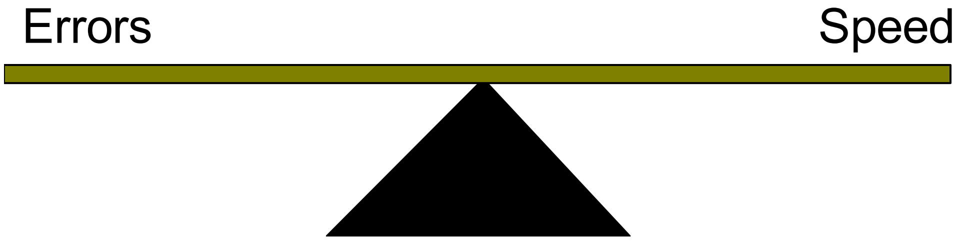
- $m = (a + b) / 2$

m for middle!

$$\int_b^a f(x) = \frac{b-a}{6}\left(f(a) + 4.f(m) + f(b)\right)$$

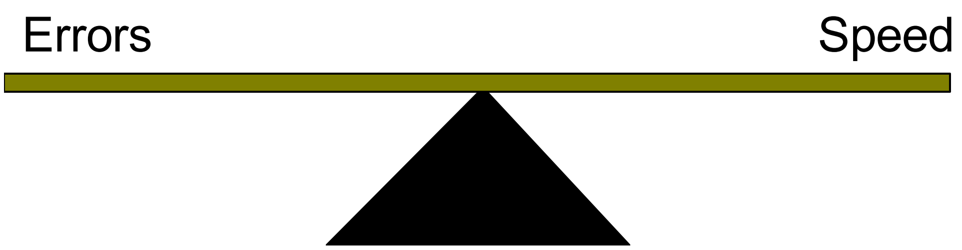
It’s all about the balance

- In real world use, computing the function calls costs time – complicated functions!
- You need some desired level of accuracy
- The choice of algorithm makes more difference than the panel size



It’ s all about the balance

- In real world use, computing the function calls costs time – complicated functions!
- You need some desired level off accuracy
- The choice of algorithm makes more difference than the panel size



- How accurate do you need your answer?

Error scaling

Method	Order	Panel area formula	Function evaluations	Error (order)
Rectangle (midpoint)	0	$(b-a)f(m)$	N	$2(b\text{-}a)^3$
Trapezium	1	$\frac{(b-a)}{2}\left[f(a)+f(b)\right]$	N+1	$(b\text{-}a)^3$
Simpson	2	$\frac{(b-a)}{6}\left[f(a)+4f(m)+f(b)\right]$	2N+1	$(b\text{-}a)^5$

$$b-a \propto \frac{1}{N}$$

So doubling the number of panels decreases the error:

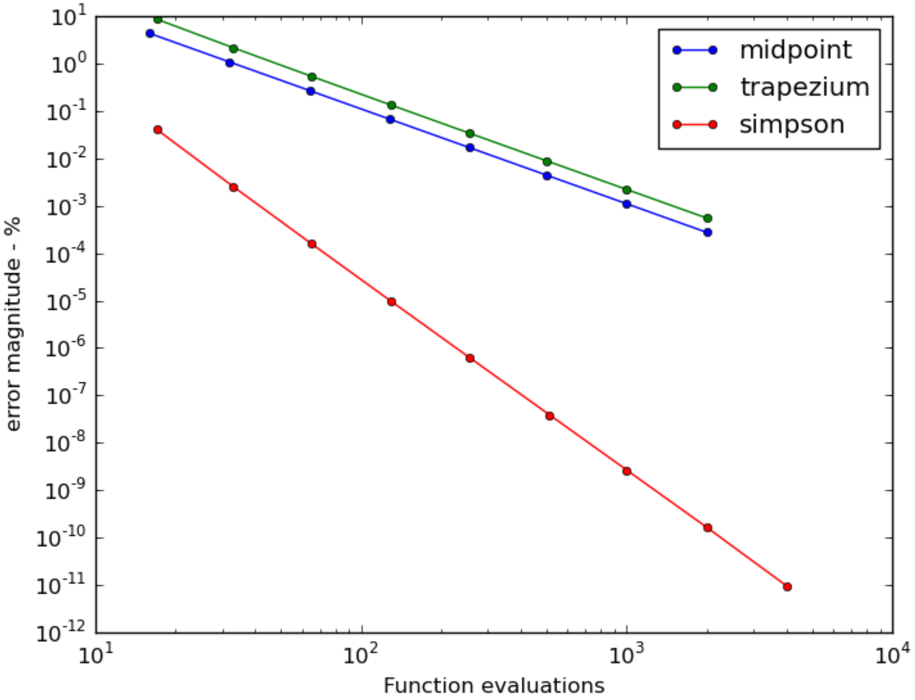
Rectangle – 8x

Trapezium – 8x

Simpson – 32x

Accuracy vs. computational cost

$$\int_0^4 x^2 \sin(x)$$



Simpson's rule is the clear winner – higher order methods are even better, but are rarely needed

Higher order methods

- Simpson's 3/8's rule
- Boole's rule
- Any higher order you want

- Generally, Simpson's rule is enough

Making some code…

A Practical Algorithm

- Let's code an integrator with the **midpoint** rule
- Weekly assessment is to code an integrator with Simpson's rule
- Find the definite integral of *f(x)* between x_0 and x_1
- Let's use 5 panels

$$area = \int_{x0}^{x1} f(x).dx$$

Start with the equation

- $S = h*f_1 + h*f_2 + h*f_3 + h*f_4 + h*f_5$

- 5 function evaluations
- 5 multiplies

Factorise

- $S = h*(f_1 + f_2 + f_3 + f_4 + f_5)$
- 5 function evaluations
- 1 multiply
- Potentially less rounding errors

Specify panel width?

- Your integration function needs to decide on a panel width.
- We could tell the code to use a specific width, *panel_width*, but depending on the integration range we may not get an integer number of panels
- E.g. integrate 0 <= x <= 1.2 with a *panel_width* of 0.5
  - *Blackboard example*
- We would have to add some more code to handle this 'special case' (e.g. use a different width final panel)
  - **It's 'special case' code that makes most of the bugs!**

Specify number of panels?

- Instead we could specify the number of panels to use, *N_panels*

- The code then computes
  - panel_width = *(x1-x0)/N_panels*

- Now we know that the panels always fit the integration range – no special case code needed.

Specify number of panels?

```
from __future__ import division
import numpy


def f(x):
        return x**4
```

Specify number of panels?

```
def integrate_rect(a,b,n_panels):
        h=(b-a)/n_panels
        func_sum=0.0
        for ix in range(n_panels):
                x=a+ix*h+h/2 # not x=x+h as cumulative
                func_sum=func_sum+f(x)

        return func_sum*h     #at end so only do it once
```

Specify number of panels?

```
a=0

b=2

num= integrate_rect(a,b,100)

#test the code using the analytic solution

ana=(b**5)/5-(a**5)/5

print num, ana, (num-ana)/ana
```

demo0.py - G:\teaching\2010-2011\CompPhys\02 numerical integratio...

File Edit Format Run Options Windows Help

```python
from __future__ import division

import numpy

def f(x):
    return x**4

# Variables

# a - left  (x-axis)  of a panel
# b - right (x-axis)  of a panel
# m - middle (x-axis) of a panel

def integrate_rect(x0, x1, n_panels):
    ''' Integrate the function f between x0 and x1 '''
    # Split the intervale x0 <= x <= x1 into panels
    panel_width = (x1-x0) / n_panels

    # Some of f(0) + f(1) + f(2) + ...
    func_sum = 0

    for ix in range(n_panels):
        # Find the left edge of this panel
        a = x0 + ix * panel_width
        # Find the midpoint
        m = a + panel_width / 2
        func_sum += f(m)

    return panel_width * func_sum

x0, x1 = 0, 2
print integrate_rect(x0, x1, 100)
# Analytical solution is x**5/5
print (x1**5/5) - (x0**5/5)
```

Ln: 9 Col: 0

Test code

Outputs are

6.399466676

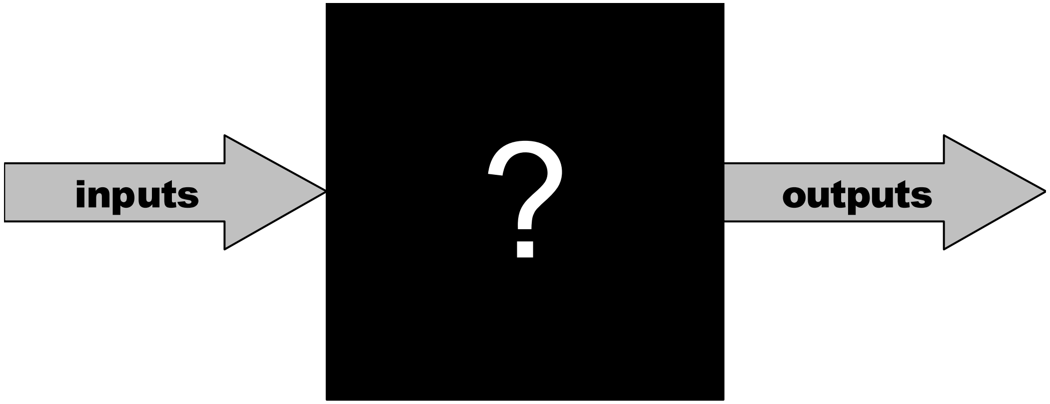
and

6.4

Black Box integrators

Black Box

- ‘black box’ code is some third party module
- You know how to use it (*API Documentation*)
- Perhaps you don’t know or care about the details of how it works
  - *Caveat Emptor*
  - *Brain rot!*



scipy.integrate

Python Shell

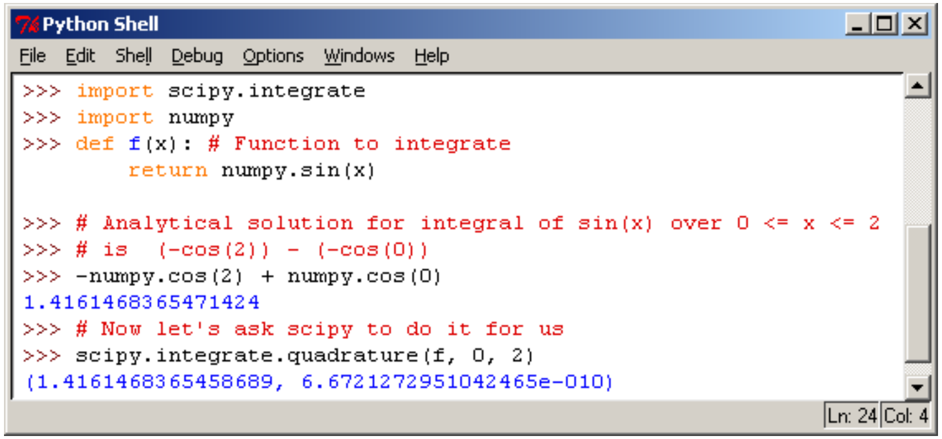
File Edit Shell Debug Options Windows Help

```
>>> import scipy
>>> import scipy.integrate
>>> help (scipy.integrate)
Help on package scipy.integrate in scipy:

NAME
    scipy.integrate

FILE
    d:\lang\python25\lib\site-packages\scipy\integrate\__init__.py

DESCRIPTION
    Integration routines
    ====================

     Methods for Integrating Functions given function object.

       quad          -- General purpose integration.
       dblquad       -- General purpose double integration.
       tplquad       -- General purpose triple integration.
       fixed_quad    -- Integrate func(x) using Gaussian quadrature of order n.
       quadrature    -- Integrate with given tolerance using Gaussian quadrature.
       romberg       -- Integrate func using Romberg integration.
```

Ln: 689 Col: 4

Example

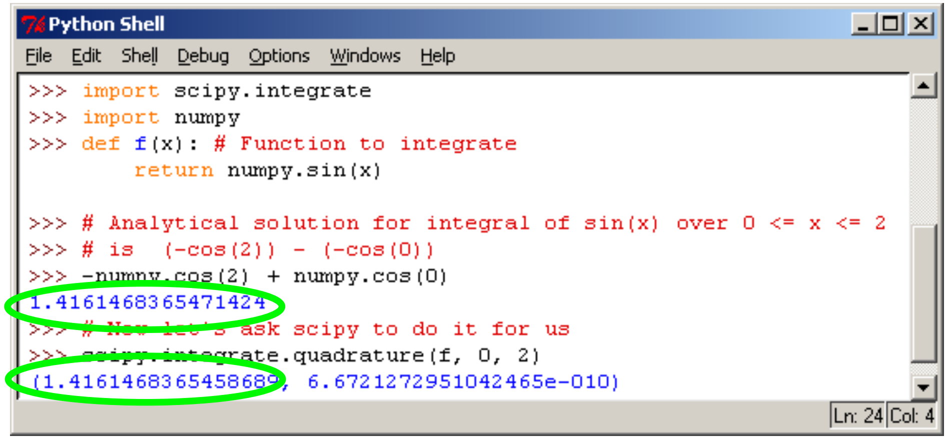
- Let' s integrate sin(x)
- Simple function so we can also do this analytically



Example

- Let' s integrate sin(x)

- Simple function so we can also do this analytically



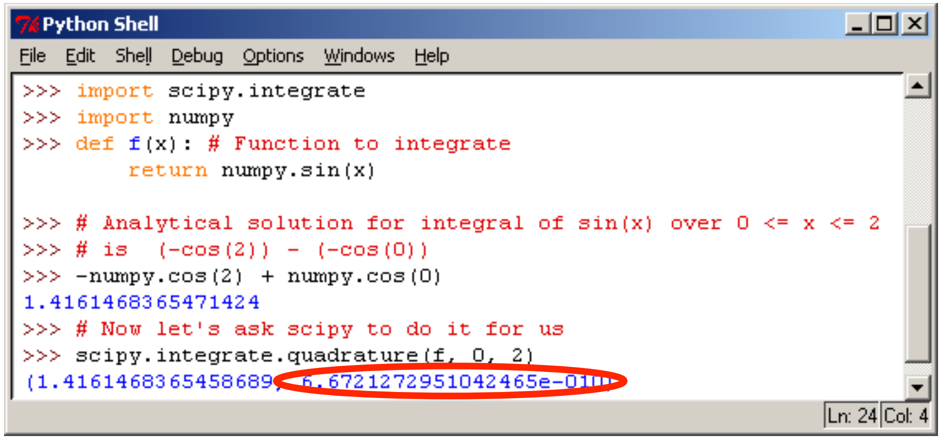
SciPy is correct! – to ten decimal places anyway

It' s always a good idea to compare third party *black box* code to a simple anayltical case:

1. **This checks that their module isn't completely broken!**
2. **It checks that you are using their module correctly**

Example

- Let' s integrate sin(x)
- Simple function so we can also do this analytically



SciPy gives us an error / accuracy value

But how does SciPy know this for any function in the absence of an analytical solution?

Tolerance analysis

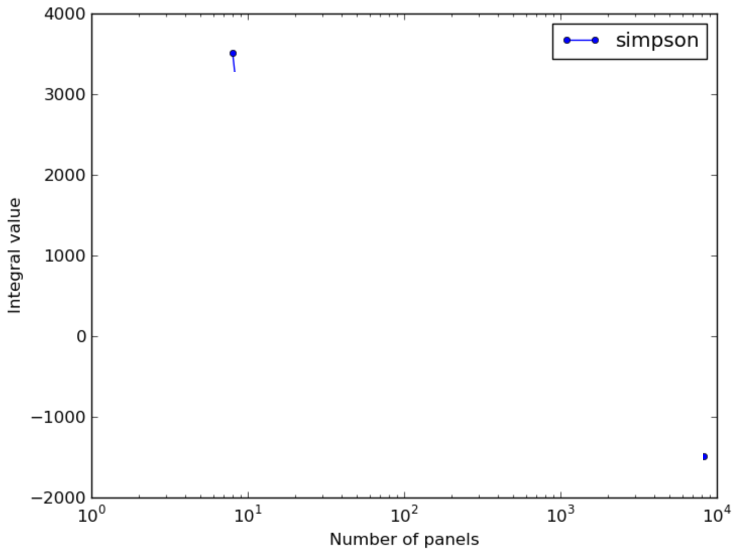
Tolerance driven approach

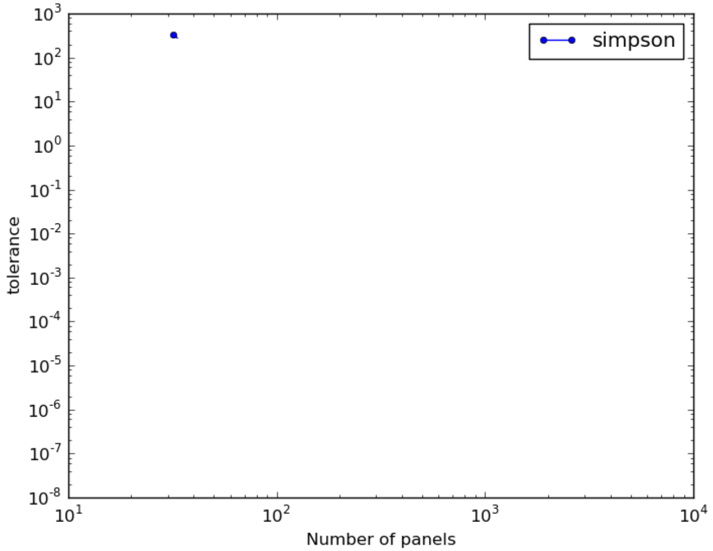
- Many third party integrators work to deliver a certain ‘tolerance’
- Tolerance – what is the change in the computed value if the number of panels is doubled?
- The tolerance asymptotically approaches zero as the error reduces with increasing step size
- This allows a known accuracy to be reached in the absence of an analytical solution (i.e. real problems!)

Tolerance example

Evaluate for N panels

$$\int_0^{64} x^2 \sin(x)\,dx$$

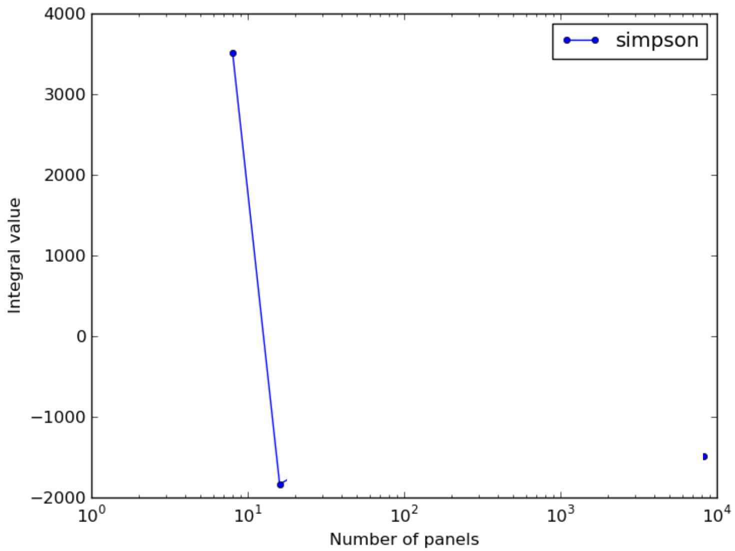


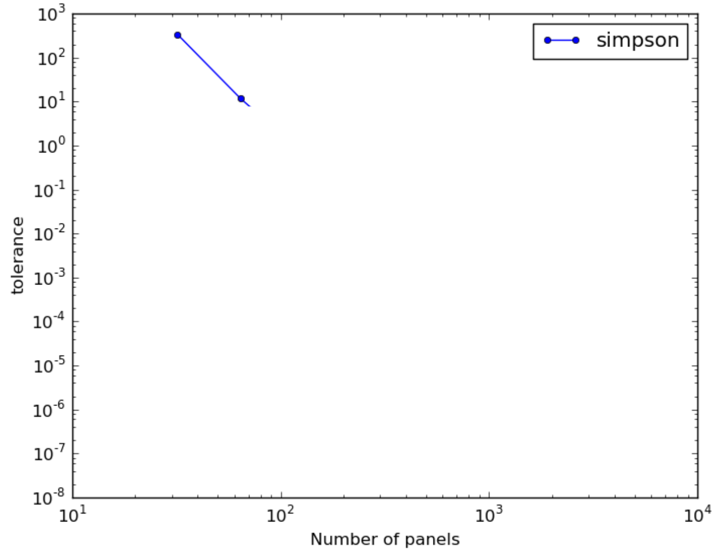


Tolerance example

Evaluate for N panels

$$\int_0^{64} x^2 \sin(x)dx$$

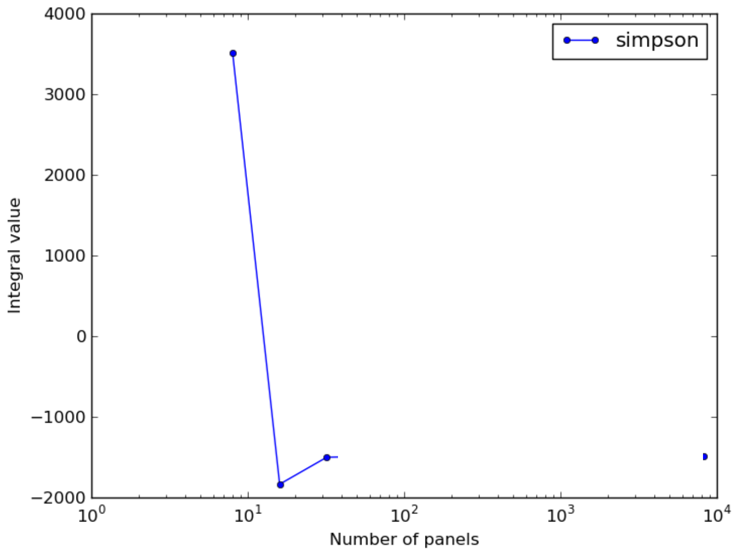


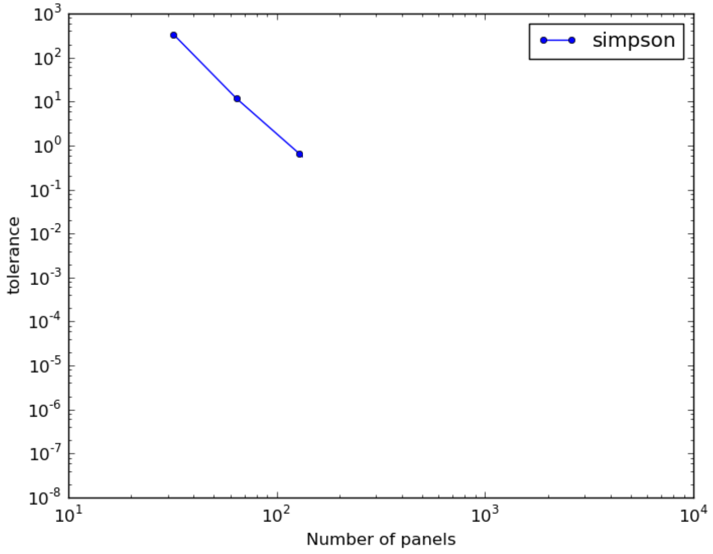


Tolerance example

Evaluate for N panels

$$\int_0^{64} x^2 \sin(x)dx$$

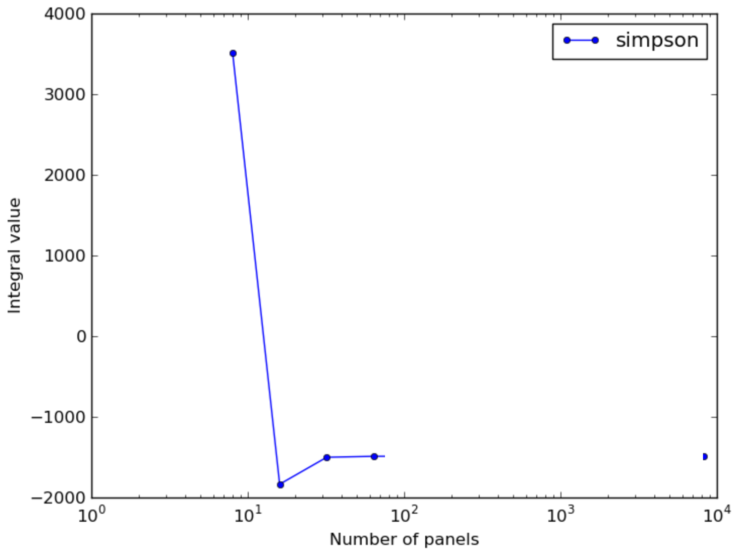


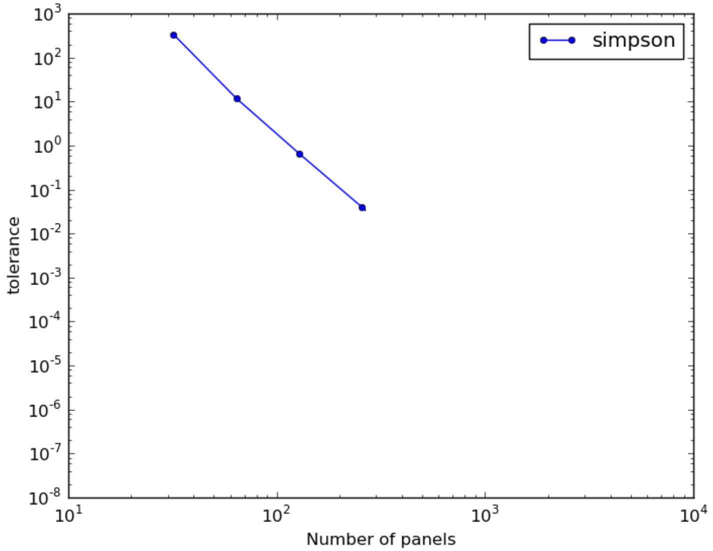


Tolerance example

Evaluate for N panels

$$\int_0^{64} x^2 \sin(x)dx$$

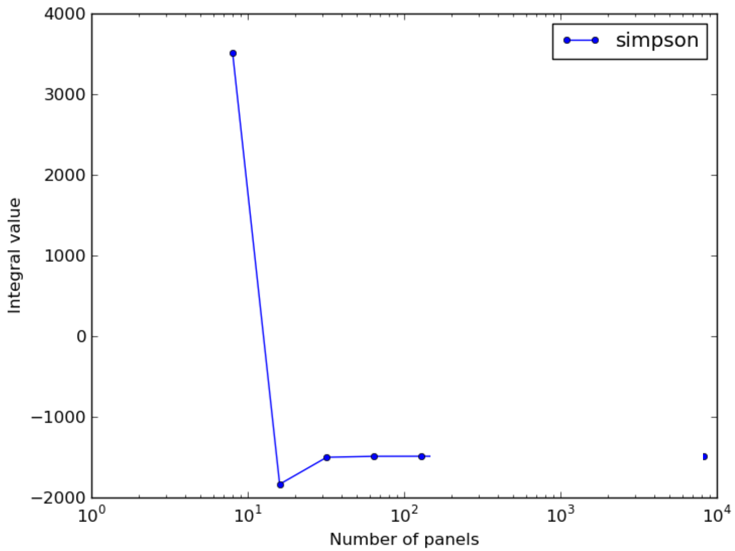


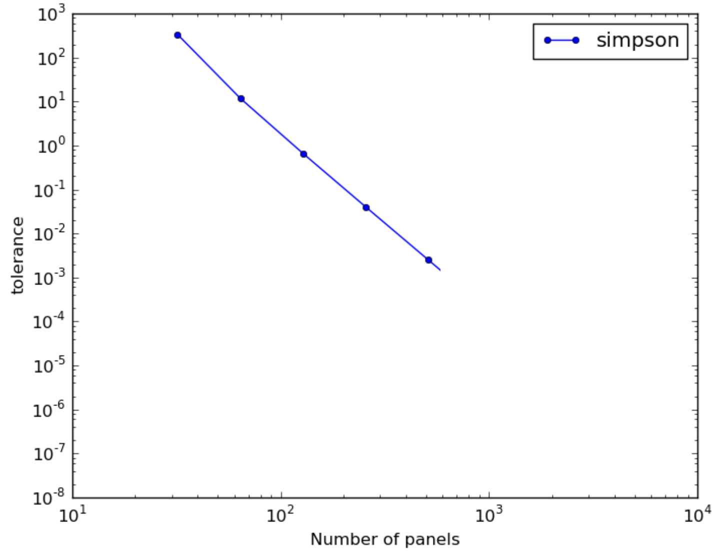


Tolerance example

Evaluate for N panels

$$\int_0^{64} x^2 \sin(x)\,dx$$

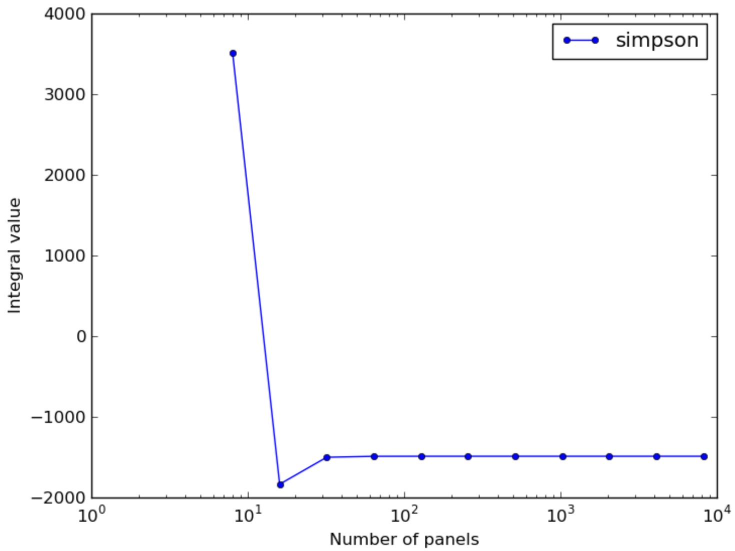


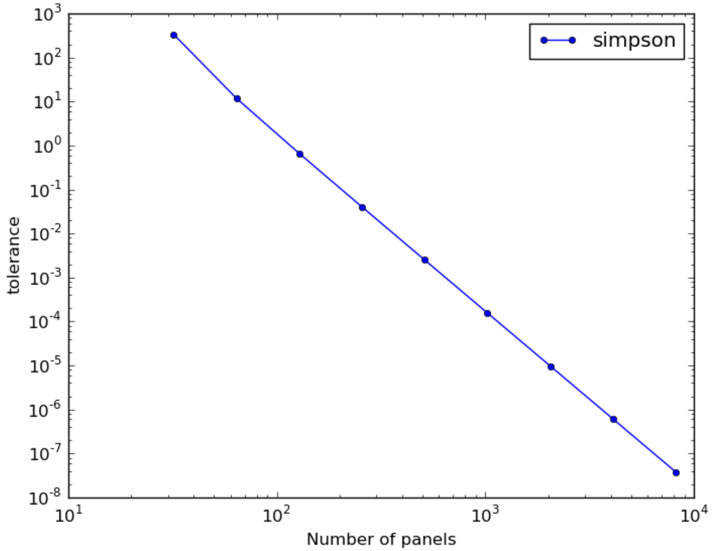


Tolerance example

Evaluate for N panels

$$\int_0^{64} x^2 \sin(x)dx$$





Stretch excercises

- Derive Simpson's rule
  - On paper, fit a 2nd order polynomial to the left-, mid- and right- points of a panel; f(a), f(m); f(b)
  - Integrate this polynomial fit
- Build a tolerance driven integrator:
- What happens if you make panel width too small?