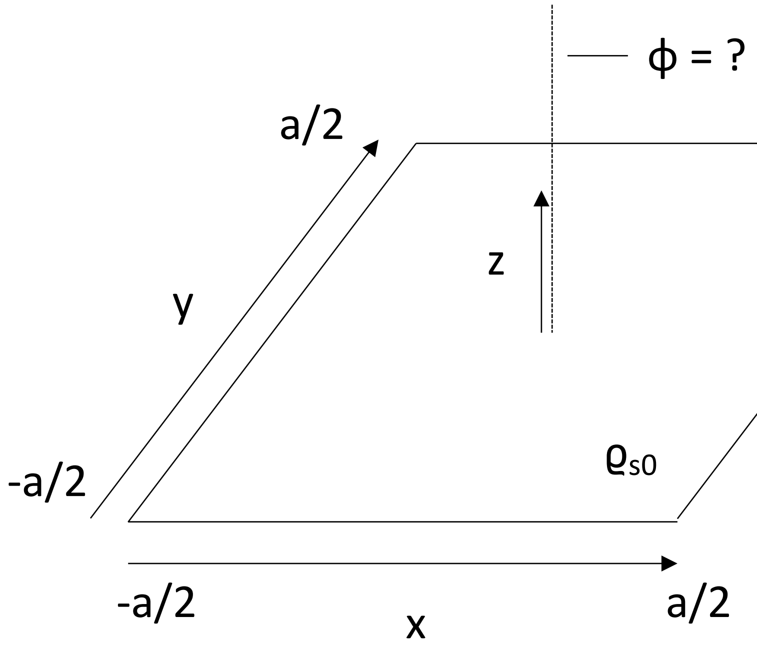
DD2370 Computational Methods for Electromagnetics *2D Integral: Convergence, Performance*

Stefano Markidis – KTH Royal Institute of Technology

Problem

- Calculate the electrostatic potential on the symmetry axis of a uniformly charged square
  - Calculate numerically a 2D Integral



$$\phi(0,0,z) = \frac{\rho_{s0}}{4\pi\epsilon_0}\int_{x'=-a}^{a} dx' \int_{y'=-a}^{a} \frac{dy'}{(x'^2 + y'^2 + z^2)^{1/2}} = \frac{\rho_{s0}}{\pi\epsilon_0} I(z,a)$$

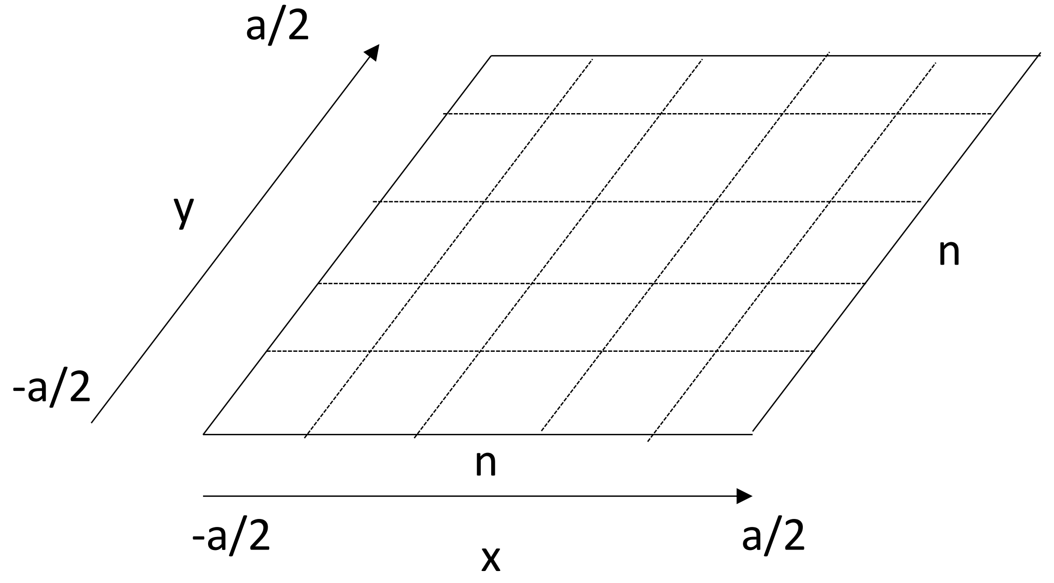
Calculate numerically $I(z,a) \equiv \int_{x'=0}^{a} dx' \int_{y'=0}^{a} \frac{dy'}{(x'^2 + y'^2 + z^2)^{1/2}}$

Strategy to Calculate Numerically the Integral

To calculate the integral *I (a,z)* over *dx* and *dy* numerically:

1. We split the square into n^2 smaller squares of side $h = a/n$
2. On each small square, we apply a simple integration rule (*midpoint*)
3. We sum all the contributions from small squares

$$I(z,a) \equiv \int_{x'=0}^{a} dx' \int_{y'=0}^{a} \frac{dy'}{(x'^2 + y'^2 + z^2)^{1/2}}$$



Numerical Integration over Small Square

- Lets at first consider **a single square** of size Δx·Δy.

Centered form

$$I = \int_0^{\Delta y}\int_0^{\Delta x} f(x,y)\,dxdy = \int_{-\Delta y/2}^{\Delta y/2}\int_{-\Delta x/2}^{\Delta x/2} f(\Delta x/2 + x, \Delta y/2 + y)\,dxdy$$

Using **finite-difference trick** (Next module topic): we apply 2D Taylor expansion to *f(Δx/2+x,Δy/2+y)* at expansion point *(Δx/2,Δy/2)* which gives

$$\begin{aligned} f(\Delta x/2 + x, \Delta y/2 + y) &= f(\Delta x/2, \Delta y/2) \\ &+ x f_x(\Delta x/2, \Delta y/2) \\ &+ y f_y(\Delta x/2, \Delta y/2) \\ &+ O(x^2 + y^2) \longleftarrow \text{Truncation Error} \end{aligned}$$

Numerical Integration over Small Square

$$I = \int_{-\Delta y/2}^{\Delta y/2} \int_{-\Delta x/2}^{\Delta x/2} f(\Delta x/2 + x, \Delta y/2 + y)\, dxdy$$

$\big\downarrow$ Substitute with Truncated Taylor Expansion

$$= \int_{-\Delta y/2}^{\Delta y/2} \int_{-\Delta x/2}^{\Delta x/2} f(\Delta x/2, \Delta y/2) + xf_x(\Delta x/2, \Delta y/2) + yf_y(\Delta x/2, \Delta y/2) + O(x^2 + y^2)$$

$$= \int_{-\Delta y/2}^{\Delta y/2} \int_{-\Delta x/2}^{\Delta x/2} f(\Delta x/2, \Delta y/2)\, dxdy$$

$$+ \int_{-\Delta y/2}^{\Delta y/2} \int_{-\Delta x/2}^{\Delta x/2} xf_x(\Delta x/2, \Delta y/2) + yf_y(\Delta x/2, \Delta y/2) dxdy$$

$$+ \int_{-\Delta y/2}^{\Delta y/2} \int_{-\Delta x/2}^{\Delta x/2} O(x^2 + y^2)\, dxdy$$

$$= f(\Delta x/2, \Delta y/2)\Delta x \Delta y + 0 + \Delta y \int_{-\Delta x/2}^{\Delta x/2} O(x^2)\, dx + \Delta x \int_{-\Delta y/2}^{\Delta y/2} O(y^2)\, dy$$

$$= f(\Delta x/2, \Delta y/2)\Delta x \Delta y + O(\Delta y \Delta x^3 + \Delta x \Delta y^3) \longrightarrow I = f(c)\Delta x^2 + O(\Delta x^4)$$

Square Dx = Dy

Truncation Error

Error over whole Square

Truncation Error

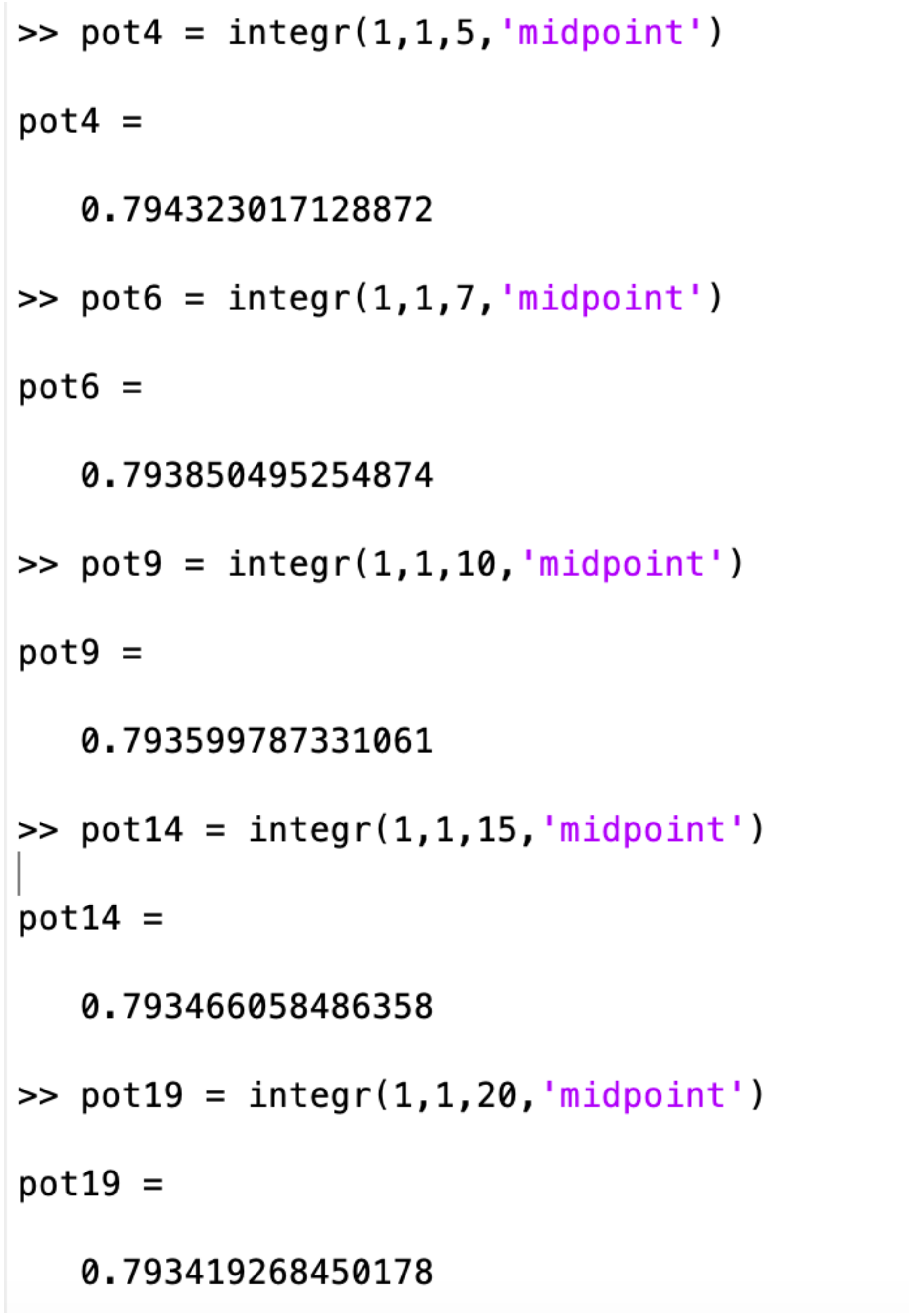
$$\epsilon = N_x \cdot N_y \cdot O(\Delta x^4) = N_x^2 \cdot O(\Delta x^4) = \frac{1}{\Delta x^2} O(\Delta x^4) = O(\Delta x^2)$$

The method is said **second order** accuracy

Matlab Code (Available in Canvas)

```matlab
function pot = integr(z, a, n, rule)

x  = linspace(0, a, n+1);
y  = linspace(0, a, n+1);
h  = a/n;
zs = z^2;

if (strcmp(rule, 'midpoint'))
  profile on
  % Midpoint integration
  xs(1:n) = (x(1:n) + h/2).^2;
  ys(1:n) = (y(1:n) + h/2).^2;
  % Set the accumulator to zero
  intSquare = 0;
  for i=1:n
      for j=1:n
          intSquare = intSquare + 1./sqrt(xs(i) + ys(j) + zs);
      end
  end
  profile off
```

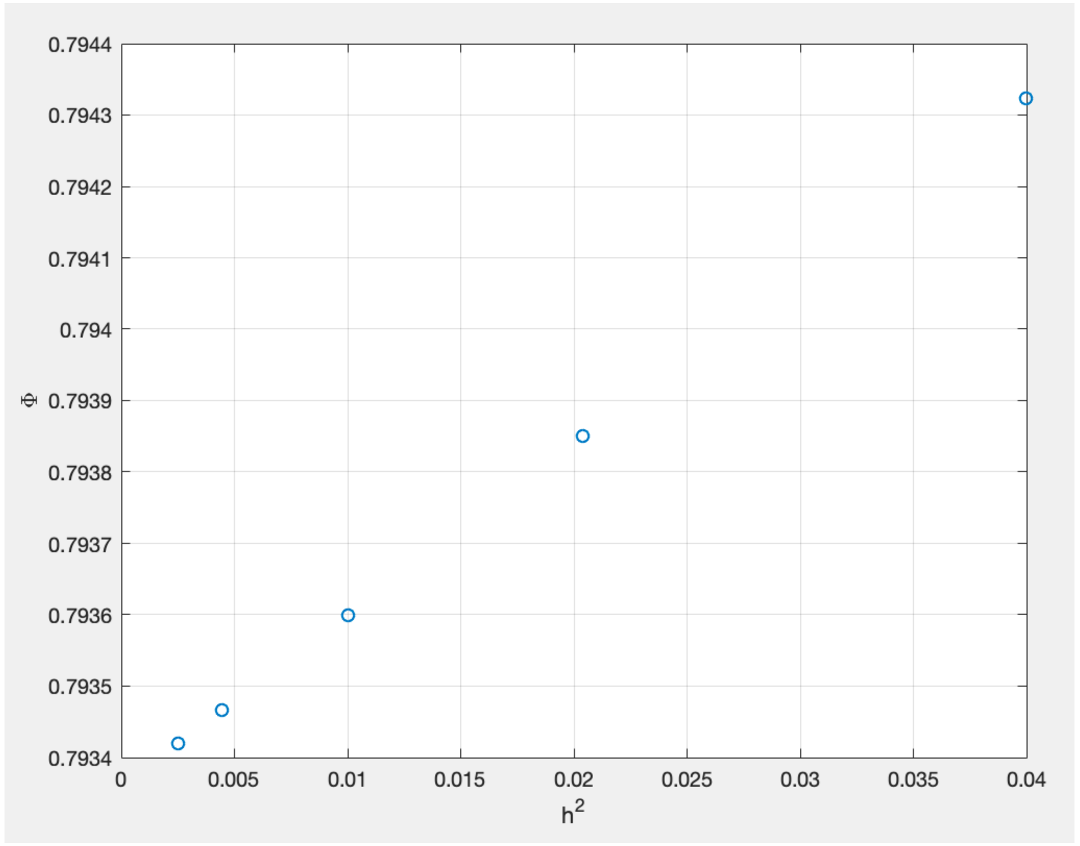
Run The Code



```
>> pot4 = integr(1,1,5,'midpoint')

pot4 =

   0.794323017128872

>> pot6 = integr(1,1,7,'midpoint')

pot6 =

   0.793850495254874

>> pot9 = integr(1,1,10,'midpoint')

pot9 =

   0.793599787331061

>> pot14 = integr(1,1,15,'midpoint')

pot14 =

   0.793466058486358

>> pot19 = integr(1,1,20,'midpoint')

pot19 =

   0.793419268450178
```

Convergence Study

Increase n



```
>> pot = [pot4 pot6 pot9 pot14 pot19];
>> h2 = [0.2^2 0.14286^2 0.1^2 0.06667^2 0.05^2];
>> plot(h2,pot,'o')
>> xlabel('h^2')
>> ylabel('\Phi')
>> grid on
```

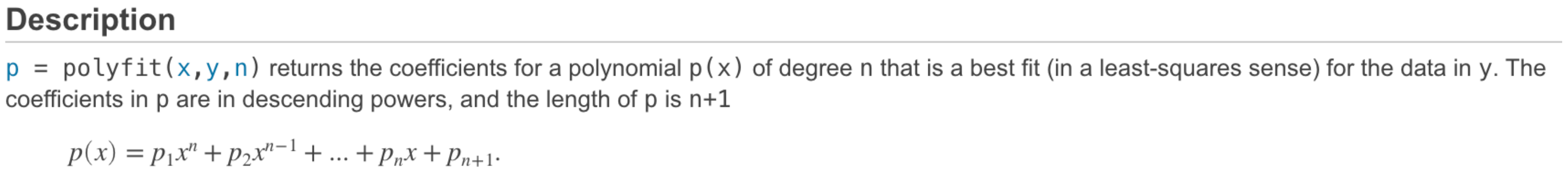
Plot pot against h^2

Extrapolation to Zero Cell Size (h=0)

$$I_{\mathrm{midp}}(h) = I_0 + I_2 h^2 + \cdots$$

I_0 is the extracted result.

We extrapolate the computed results as a polynomial fit in *h^2* using the MATLAB command.

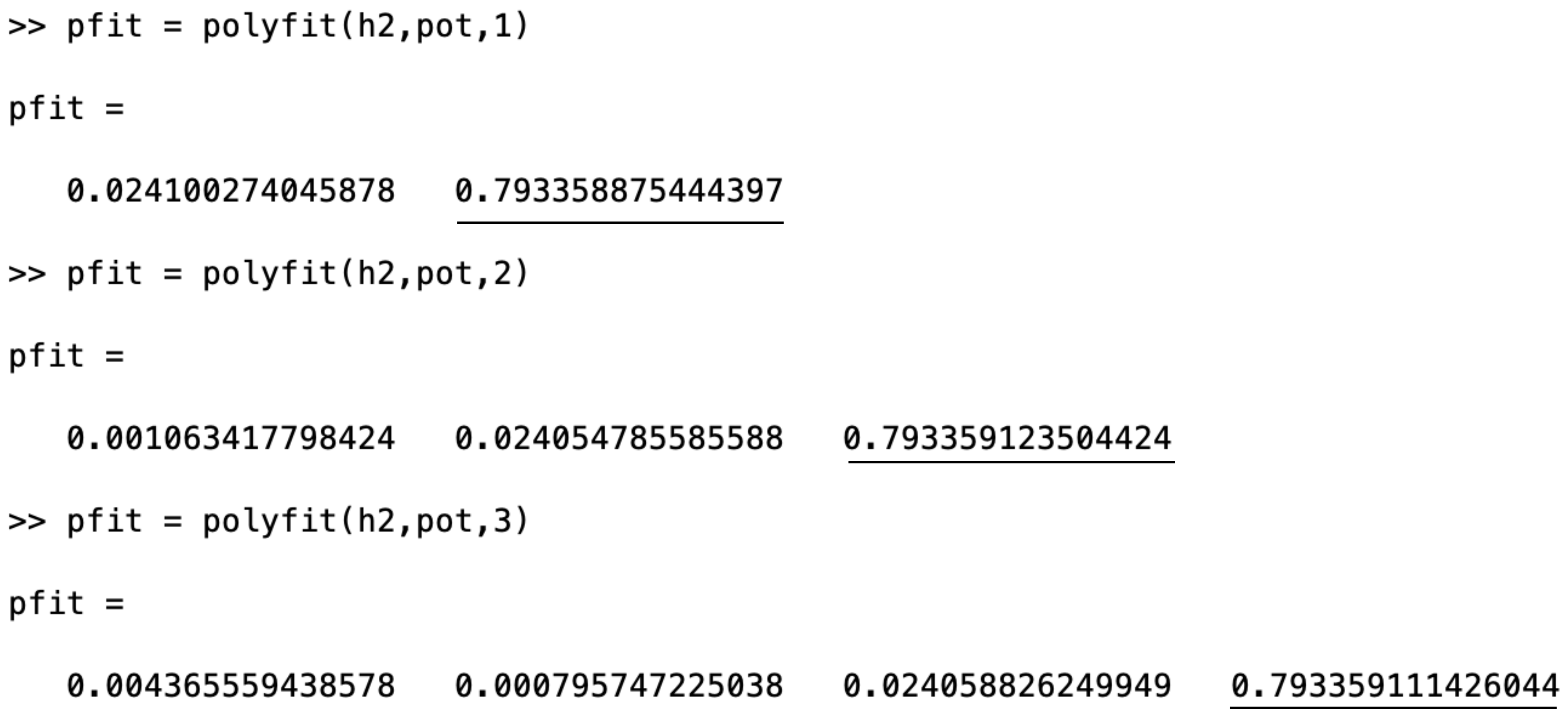


Description

`p = polyfit(x,y,n)` returns the coefficients for a polynomial `p(x)` of degree n that is a best fit (in a least-squares sense) for the data in y. The coefficients in p are in descending powers, and the length of p is n+1

$$p(x) = p_1x^n + p_2x^{n-1} + \ldots + p_nx + p_{n+1}.$$

Extrapolation to Zero Cell Size



```
>> pfit = polyfit(h2,pot,1)

pfit =

   0.024100274045878   0.793358875444397

>> pfit = polyfit(h2,pot,2)

pfit =

   0.001063417798424   0.024054785585588   0.793359123504424

>> pfit = polyfit(h2,pot,3)

pfit =

   0.004365559438578   0.000795747225038   0.024058826249949   0.793359111426044
```

How to Estimate Convergence Rate in Practice

- In many real-word case (e.g. complex geometries, complex boundary conditions, presence of interfaces, no exact solution, different parameters for resolution studies) it is challenging either to estimate the accuracy order or estimate errors.
- Strategy
  - Take High Resolution Case as ”Exact” solution
  - Calculate the error with cases with smaller h
    - Assume Round Error is small
  - Plot h, h^2, h^3, … on the same plot

In Real-world Scenarios

```
>> pot1000 = integr(1,1,1000,'midpoint')

pot1000 =

   0.793359145382788
```

Exact Solution

"Approximate" solutions

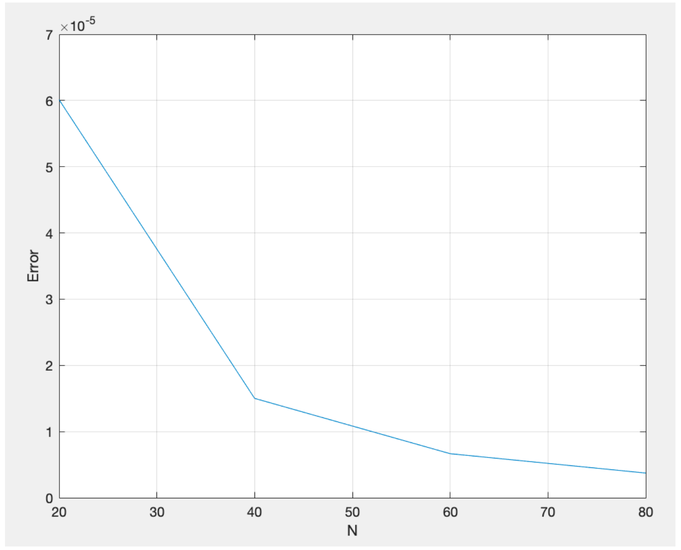
```
>> pot20 = integr(1,1,20,'midpoint')

pot20 =

   0.793419268450178

>> pot40 = integr(1,1,40,'midpoint')

pot40 =

   0.793374156894327

>> pot60 = integr(1,1,60,'midpoint')

pot60 =

   0.793365803701214

>> pot80 = integr(1,1,80,'midpoint')

pot80 =

   0.793362880142623
```

Calculate Error

```
>> errorInt = [abs(pot20 - pot1000), (pot40 - pot1000), abs(pot60 - pot1000), abs(pot80 - pot1000)]

errorInt =

   1.0e-04 *

   0.601230673901165   0.150115115394867   0.066583184258340   0.037347598350612
```

```
>> plot([20 40 60 80],errorInt)
>> xlabel('N'); ylabel('Error')
>> grid on
```

Linear Scale ☹

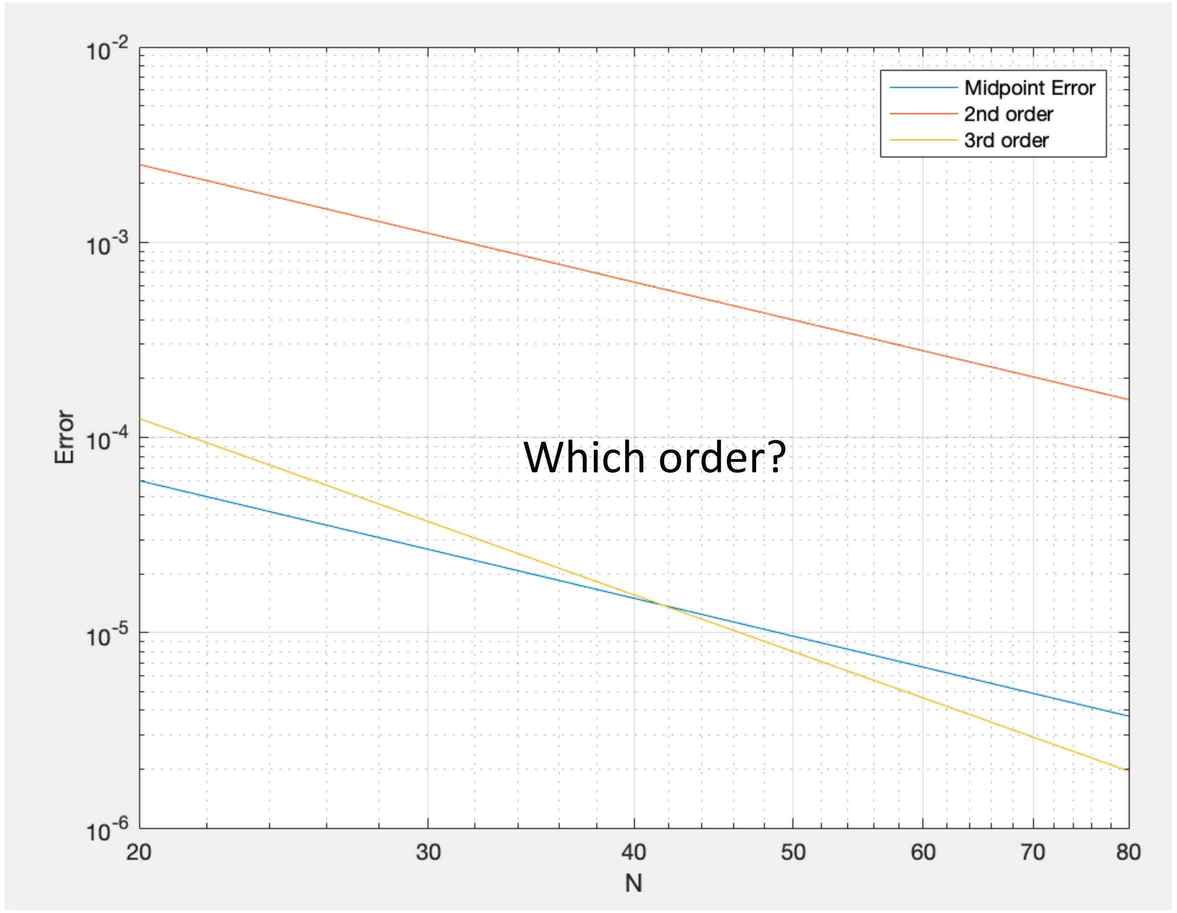


$\longrightarrow$

Loglog scale

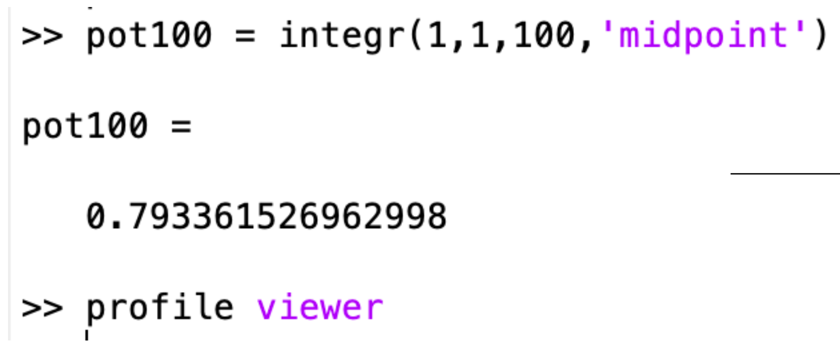
```
>> loglog([20 40 60 80],errorInt)
>> xlabel('N'); ylabel('Error')
grid on
```

```
>> hold on; loglog([20 40 60 80],1./[20 40 60 80].^2)
>> loglog([20 40 60 80],1./[20 40 60 80].^3)
>> legend('Midpoint Error','2nd order','3rd order')
```



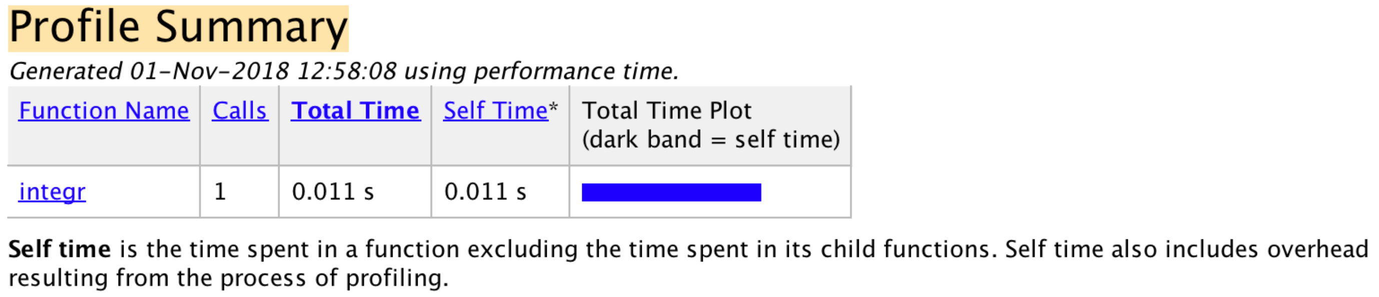
Which order?

Timing using Matlab Profiler



```
>> pot100 = integr(1,1,100,'midpoint')

pot100 =

   0.793361526962998

>> profile viewer
```

…

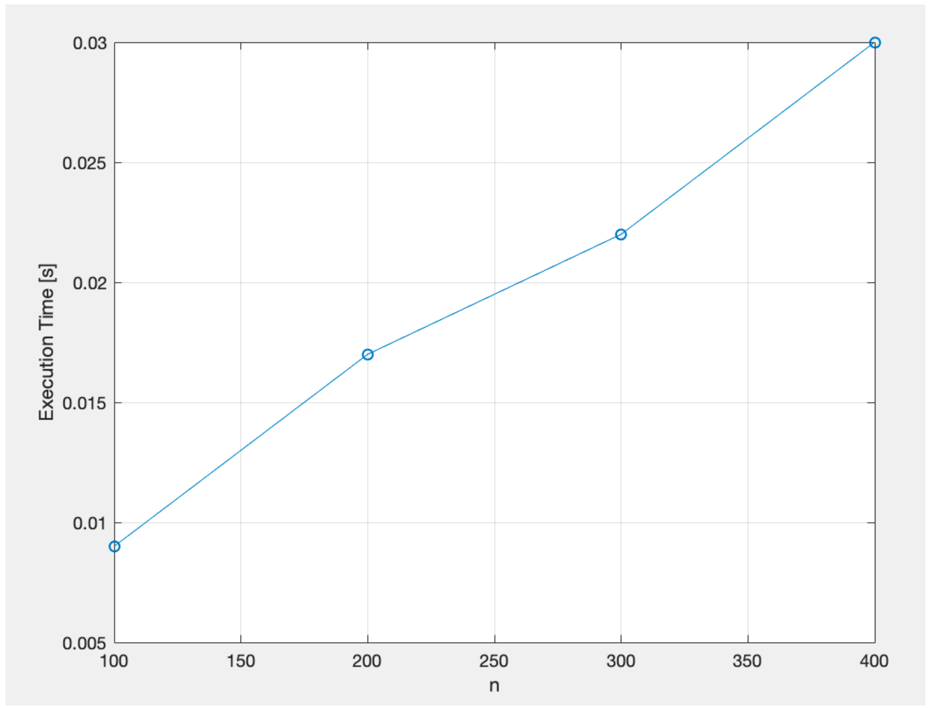


Profile Summary

Generated 01-Nov-2018 12:58:08 using performance time.

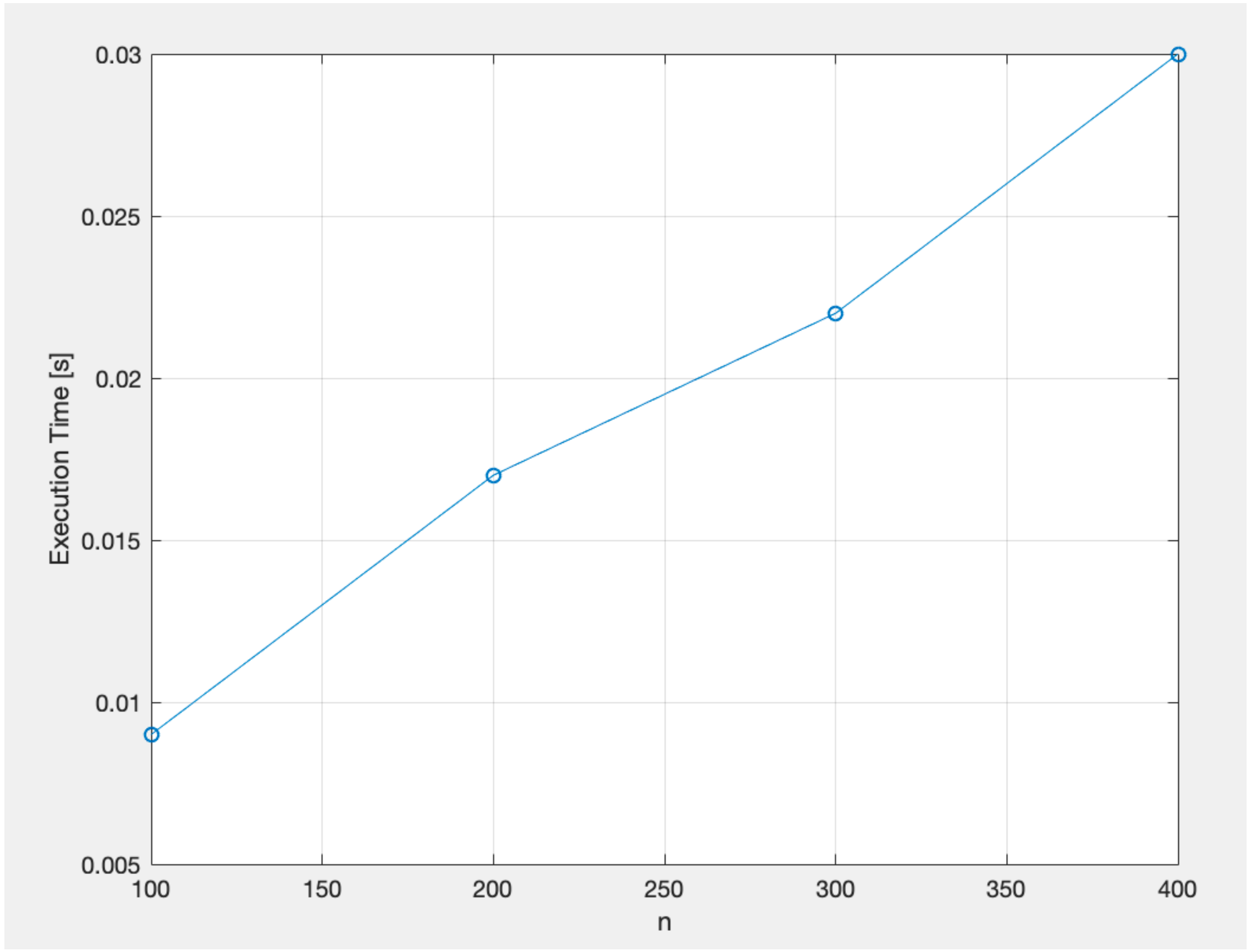
Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
integr	1	0.011 s	0.011 s	▬

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.



Matlab
9.3.0.713579 (R2017b)
.9 GHz Intel Core i5
16 GB 2133 MHz LPDDR3

Can we say something about complexity?



Trapezoidal Rule

$$\int_x^{x+h} f(x)dx \approx \frac{h}{6}\left[f(x) + 4f\left(x + \frac{h}{2}\right) + f(x+h)\right]$$

- It is possible to use **higher order** discretization scheme: lower truncation error but higher number of computations

```
intSquare = 0;
for i = 1:n
  x1 = x(i)^2; x2 = (x(i) + h/2)^2; x3 = (x(i) + h)^2;
  y1(1:n) = y(1:n).^2;
  y2(1:n) = (y(1:n) + h/2).^2;
  y3(1:n) = (y(1:n) + h).^2;
  intSquare = intSquare + sum(  1./sqrt(x1+y1+zs) + 1./sqrt(x1+y3+zs) ..
                  + 1./sqrt(x3+y1+zs) + 1./sqrt(x3+y3+zs)...
                  + 4./sqrt(x2+y1+zs) + 4./sqrt(x2+y3+zs)...
                  + 4./sqrt(x1+y2+zs) + 4./sqrt(x3+y2+zs)...
                  + 16./sqrt(x2+y2+zs))/36;

end
```

Singularity Case

z=0 → Integrand is singular but integral still converging

```
>> pot400 = integr(0,1,400,'midpoint')

pot400 =

   1.761737828182028
```

```
>> pot400 = integr(0,1,400,'simpson')

pot400 =

   Inf
```

$I(0,1) = 2\ln(1+\sqrt{2}) \approx 1.762747$ Analytical result

Low-order methods are better in dealing with singularity; they are more “resilient”.