

# Introduction to Matlab

Lesson 03 — Basics of Root Finding, Numerical Differentiation, and Integration

Rafael Serrano-Quintero

Department of Economics  
University of Barcelona

# Basics of Root Finding

# Root Finding — The Problem

Say we want to find all  $x$  such that  $f(x) = 0$ . If:

- $f(x) = ax^2 + bx + c \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$  solves the problem for any  $\{a, b, c\} \in \mathbb{R}$  (and  $a \neq 0$ ).
- Sometimes, we cannot solve explicitly for  $x$  even in the realm of polynomials!
- If  $f(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$  where  $\{a, b, c, d, e, f\} \in \mathbb{R}$  we do not have an explicit formula to solve for  $f(x) = 0$ . Neither for polynomials of order  $\geq 5$ . But we know **such roots do exist!** see Fundamental Theorem of Algebra
- Then... what should we do?

# Root Finding — Intuition

Say we have a nonlinear  $f(x)$  for which we know at least one root exists.

- Suppose we know the root is *somewhere around*  $x_0$ .
- Starting from  $x_0$  construct a sequence of  $\{x_k\}$  hoping it converges to a root  $x^*$  such that  $f(x^*) = 0$ .

This highlights two important features of root finding *algorithms*:

- **Convergence:** is the sequence  $\{f(x_k)\}$  getting *closer* to  $f(x^*) = 0$ ?
- **Stopping criteria:** how do we know we are *close enough*?

We can discuss convergence and stopping criteria in each method studied.

## Bisection Method

# Bisection Method — Intuition

Consider  $f : \mathbb{R} \mapsto \mathbb{R}$  where the problem is  $f(x^*) = 0$ . Suppose  $f(x)$  is continuous in  $[a, b]$  and we know  $\exists x^* \in [a, b]$  The ideas presented generalize to  $n$  dimensions

- **Bisect** the interval  $[a, b]$  and take the middle point  $c = \frac{a+b}{2}$ . If  $f(c) = 0$  we are done. Otherwise,  $x^*$  must be in either  $[a, c]$  or  $[c, b]$ .
- Find the one that contains  $x^*$  and bisect again. **How?!?**
- Continue until the interval is as small as the accuracy desired.

# Bisection Method — Intuition

## Theorem 1 (The Intermediate Value Theorem (IVT))

Suppose  $f(x)$  is continuous on  $[a, b] \subseteq \mathbb{R}$  and  $M$  in between  $f(a)$  and  $f(b)$ . Then, there is at least one  $c \in (a, b)$  such that  $f(c) = M$ . If  $f(a) < 0 < f(b)$  then, there is a root  $x = c$  such that  $f(c) = 0$ .

- If  $f(c) < 0$ , by the IVT, the root of  $f(x)$  must be in  $[c, b]$
- Otherwise, if  $f(c) > 0$ , by the IVT, the root must be in  $[a, c]$
- Note that this algorithm finds **a zero, not all zeros** of  $f(x)$

# Bisection Method — the Algorithm

1. Initialize and bracket a zero. (Initial guess)
  - Find  $x^L < x^R$  such that  $f(x^L)f(x^R) < 0$
  - Choose stopping rule parameters
2. Compute  $x^M = \frac{x^L + x^R}{2}$
3. Test if  $x^M$  is a root. If so, stop. (Test if it is an acceptable solution)
4. If  $x^M$  is not a root, refine interval. (Iterate)
  - If  $f(x^M)f(x^L) < 0$  let  $x^R = x^M$  and leave  $x^L$  unchanged.
  - Else,  $x^L = x^M$  and leave  $x^R$  unchanged.
5. Repeat until the stopping rule tells us to stop.



## Bisection Method — Remarks

- The algorithm **always converges** (we always find a solution).
- Convergence can be very slow but it is a very **reliable** method.
- We have not yet defined proper stopping criteria.

### Stopping Criteria:

1. The value of the function is lower than or equal to the tolerance  $|f(x^M)| \leq \delta$ .
2. The length of the interval is very small.  $(x^R - x^L) / (1 + |x^L| + |x^R|) \leq \varepsilon$ .
3. The number of iterations is larger than a predetermined number  $N$ .

# Bisection Method — Stopping Criteria

## Criterion 1:

- Parameter  $\delta$  controls the “acceptable error”.
- Sometimes, computing  $f(x)$  involves other numerical operations that add errors to the computation of  $f$ .
- We should take that into account when choosing a value for  $\delta$

# Bisection Method — Stopping Criteria

## Criterion 2:

- The size of the interval is too small whenever

$$\frac{x^R - x^L}{(1 + |x^L| + |x^R|)} \leq \varepsilon$$

- No sense in choosing  $\varepsilon = 0$ , unachievable. Same for  $\varepsilon = 10^{-130}$ , computers store finite digits.
- Bear in mind the sizes of  $x^L$  and  $x^R$ . If  $x^L$  and  $x^R$  are of the order  $10^{10}$ , convergence of  $x^R - x^L < 10^{-5}$  is infeasible.
- Note that adding 1 avoids problems when  $x^R \approx x^L \approx 0$

# Bisection Method — Stopping Criteria

## Criterion 3:

- We can compute the minimum number of iterations  $N$  to achieve accuracy  $\delta$
- We want the length of the interval after  $N$  iterations lower than  $\delta$

$$\frac{x^R - x^L}{2^N} \leq \delta \Rightarrow 2^N \geq \frac{x^R - x^L}{\delta}$$

taking logs on both sides and simplifying

$$N \geq \frac{\log(x^R - x^L) - \log(\delta)}{\log(2)}$$

which depends on the length of the interval  $[x^L, x^R]$  and the value of  $\delta$

## Bisection Method — Example

Compute the positive root of  $f(x) = x^3 - 6x^2 + 11x - 6$

1. Find the interval  $[x^L, x^R]$
2. The function has three positive roots, let's focus on the one on the interval  $[2.5, 4]$
3. Note  $f(2.5)f(4) = -2.25 < 0 \Rightarrow$  by IVT there is a root in  $[2.5, 4]$
4. Choose  $\delta = 10^{-4}$  and  $\varepsilon = 10^{-8}$
5. The minimum number of iterations needed to achieve accuracy  $\delta$  is  $N^{min} = 13.8$ , choose  $N = N^{min} + 50$

# Bisection Method — The Actual Code

- Initialize with a `while` loop with three conditionals

```
1 while (error_f > delta) && (error_i > epsil) && (it < maxit)
2 % Actual algorithm in here
3 end
```

- Middle steps

```
1 % Compute xm
2 xm = x1 + (xr - x1) / 2; % Slightly improves performance
3
4 % Compute value of f at xm
5 fxm = myf(xm);
6
7 % Compute bounds values of f
8 fxl = myf(xl);
9 fxr = myf(xr);
```

# Bisection Method — The Actual Code

- Compute errors

```
1 % Compute errors
2 error_f = abs(fxm);
3 error_i = (xr - xl)./(1 + abs(xl) + abs(xr));
```

- Update iteration counter and refine interval

```
1 % Update iteration counter
2 it = it + 1;
3
4 % Update if necessary
5 if fxm*fxl < 0
6     xr = xm;
7 else
8     xl = xm;
9 end
```

## Newton-Raphson Method



# Newton-Raphson Method — Intuition

- When using bisection, we have only assumed continuity of  $f(x)$
- However, bisection can be slow. Newton-Raphson's method uses properties of **smooth** functions.
- This method is faster but may not always converge.
- **Key idea:** approximate  $f(x)$  by a succession of linear functions. Approximate zeros with the zeros of the linear approximations.

# Newton-Raphson — Preliminaries

## Definition 1

A number  $c$  is a **fixed point** of  $g(x)$  if  $g(c) = c$ .

- Note then that  $f(x) = 0 \Rightarrow f(x) = x - g(x) = 0$  and any fixed point  $c$  of  $g(x)$  is a root of  $f(x)$  because

$$f(c) = c - g(c) = c - c = 0$$

- Finding a root of  $f(x) \equiv$  find a fixed point of  $x = g(x)$  such that  $f(x) = 0$
- **Problem:** How to rewrite  $f(x) = 0$  as  $x = g(x)$ ?

# Newton-Raphson — Existence, Uniqueness, and Convergence

## Theorem 2 (Fixed-Point Iteration Theorem)

Let  $f(x) = 0$  be written as  $x = g(x)$ . Let  $g(x)$  satisfy:

1.  $\forall x \in [a, b], g(x) \in [a, b]$
2.  $g'(x) \in (a, b)$  and, for  $q \in (0, 1)$ ,  $g'(x)$  satisfies  $|g'(x)| < q$  for all  $x \in (a, b)$

then

1.  $\exists! c \in (a, b) : g(c) = c$  **(Existence of a unique solution)**
2. For any  $x_0 \in [a, b]$ , the sequence  $\{x_k\}$  defined by

$$x_{k+1} = g(x_k), \quad k = 0, 1, \dots$$

converges to the fixed point  $c = g(c)$ , that is to the root  $c$  of  $f(x) = 0$ .

# Newton-Raphson — Preliminaries

- In practice, verifying Assumption 1 in Theorem 2 is not easy.
- This method chooses  $g(x)$  as  $g(x) = x - \frac{f(x)}{f'(x)}$  and the iteration scheme is:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad (1)$$

- For this choice, we can state sufficient conditions for convergence.

## Theorem 3

Suppose  $f(x)$  is  $\mathcal{C}^2$  and that  $f(x^*) = 0$ . If  $x_0$  is sufficiently close to  $x^*$ ,  $f'(x) \neq 0$ , and  $|f''(x)/f'(x)| < \infty$ , the iteration scheme defined by (1) converges to  $x^*$ .

# Newton-Raphson — Preliminaries

Where does the choice for  $g(x)$  come from? Recall **Taylor's Theorem** and linearly approximate  $f(x)$  around  $x_k$ :

$$p(x) = f(x_k) + (x - x_k)f'(x_k)$$

- $p(x)$  and  $f(x)$  are tangent at  $x_k$  and close in the neighborhood of  $x_k$
- Solving for a zero of  $p(x) \Rightarrow x = x_k - \frac{f(x_k)}{f'(x_k)}$  which is the iteration scheme (1)
- Which yields our new guess for  $x_{k+1}$

# Newton-Raphson — The Algorithm

1. Choose stopping criterion parameters  $\{\epsilon, \delta, N\}$ , and a starting point  $x_0$ . Set the iteration counter  $k = 0$ .
2. Compute next iteration using (1)
3. Test for convergence. If either one of the following:
  - $|x_k - x_{k+1}| \leq \epsilon(1 + |x_{k+1}|)$
  - $|f(x_{k+1})| \leq \delta$
  - $k > N$

**Stop**

4. Repeat until one of the convergence criteria is satisfied.

## Newton-Raphson — Remarks

- The method is not guaranteed to converge. If after  $N$  iterations we have not found a root, the method failed.
- Note that we rely on our initial guess being **close** to the root. If we are far, we may very well fail.
- For some functions and starting points, we may enter an infinite loop. The sequence of iterations will oscillate without converging to any value.
- Even when passing both  $\varepsilon$  and  $\delta$  tests, we may not have found a zero.

# Newton-Raphson — Some Issues

- Consider  $f(x) = x^6$
- $x_{k+1} = (5/6)x_k$
- Still far after 100 iterations!

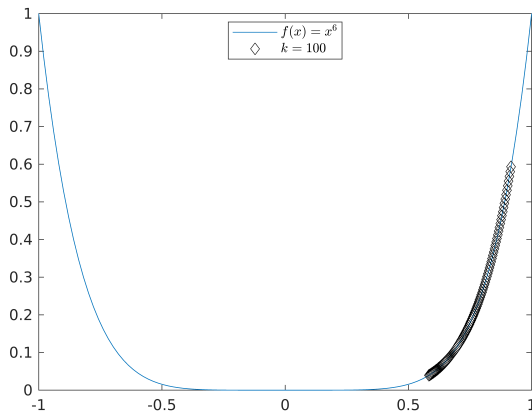


Figure 1: Convergence Issues with Newton-Raphson



## Newton-Raphson Method — Example

As with the bisection method, let's compute the root of  $f(x) = x^3 - 6x^2 + 11x - 6$  in the interval  $[2.5, 4]$ . Keep the same stopping criteria parameters.

1. Choose  $N = 64$ ,  $\delta = 10^{-4}$ , and  $\varepsilon = 10^{-8}$
2. Choose  $x_0 = 2.65$ .

# Newton-Raphson Method — The Actual Code

- Initialize with a `while` loop with three conditionals as in bisection

```
1 while (error_f > delta) && (error_i > epsil) && (it < maxit)
2 % Actual algorithm in here
3 end
```

- Middle steps

```
1 % Compute next guess
2 xkp = xk - myf(xk) ./ fprime(xk);
3
4 % Compute the errors at current guess
5 error_f = abs(myf(xk));
6 error_i = abs(xk - xkp) ./ (1 + abs(xkp));
7
8 % Update iteration counter and guess
9 it = it+1;
10 xk = xkp;
```

# Newton-Raphson vs Bisection

- In previous example, it took 14 iterations for the bisection method to find a solution.
- It took Newton-Raphson **half** of those. In 7 iterations it was done.
- Both achieved the same solution.
- Note that the first guess for bisection yields  $x^M = 3.25$ . If we start Newton-Raphson with that guess, only 4 iterations are needed.
- If we give Newton-Raphson  $x_0 = 2.5$ , we end up in the root  $x^* = 1$ .
- Both methods have pros and cons. It depends on the problem which one to choose.

# Root Finding in Matlab

- We have learned two very powerful methods to find roots
- Matlab has implemented **fzero** for such problems.
- This function combines bisection with secant and inverse quadratic interpolation (we have not seen this but many ideas apply).
- Another function is **roots** which finds the roots of polynomials.
- Familiarize yourselves with these functions and compare them with the methods we have implemented.

# Basics of Numerical Differentiation

# Numerical Differentiation — Why?

- Numerical evaluation of the derivatives in economics is **crucial!**
  - Newton-Raphson, optimization, ODEs ...
- Sometimes, it is difficult or cumbersome to compute derivatives analytically. In those cases, we turn to *numerical* evaluation of derivatives.
- Recall the definition of the derivative of a function

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (\text{can recall why?})$$

we can approximate  $f'(x)$  by choosing a *step size*  $h$ .

- That is called, *finite differences*.

# Numerical Differentiation — An Example

Take the function

$$f(x) = x^3 - 6x^2 + 11x - 6$$

- The analytical derivative is

$$f'(x) = 3x^2 - 12x + 11$$

- Let's approximate the derivative by using finite differences in Matlab with on  $x \in [-5, 5]$
- Compute  $dy$  as  $f(x + h) - f(x)$  for a fixed  $h$  and  $dx = h$ .
- The numerical derivative is given by  $f'(x) = \frac{dy}{dx}$ .

# Numerical Differentiation — An Example

As  $h \rightarrow 0$ , the derivative converges.

```
1 N = 10;  
2 x = linspace(-5,5,N);  
3 h = 1;  
4 % Numerical derivative  
5 dy = myf(x+h) - myf(x);  
6 dx = h.*ones(size(dy));  
7 fp_num = dy./dx;
```

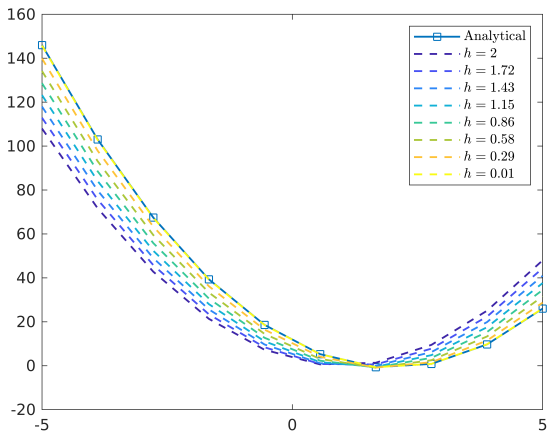


Figure 2: Accuracy of Finite Difference



# Numerical Differentiation — An Application to ODEs

- The Solow-Swan growth model finds an equilibrium solution by solving the ODE

$$\dot{k} = \frac{dk}{dt} = sf(k) - \delta k \quad (2)$$

where  $k$  is physical capital,  $f(k)$  is the production function,  $s$  is an exogenous savings rate, and  $\delta$  is the depreciation rate of capital.

- Approximate  $k(t)$  at  $N$  discrete points in the time dimension  $t^n, n = 1, \dots, N$  and denote the distance between the points by  $h$ .
- Approximate  $\dot{k}$  using finite differences  $\dot{k}(t^n) \approx \frac{k^{n+1} - k^n}{h}$
- We can compute  $k(t^{n+1})$  recursively given  $h$  and  $k(0) = k_0$

$$k(t^{n+1}) = k(t^n) + h (sf(k^n) - \delta k^n)$$

# Numerical Differentiation — An Application to ODEs

- Horizon  $T = 70$
- $N = 10$  points.
- Compare with `ode45` and analytical solution.

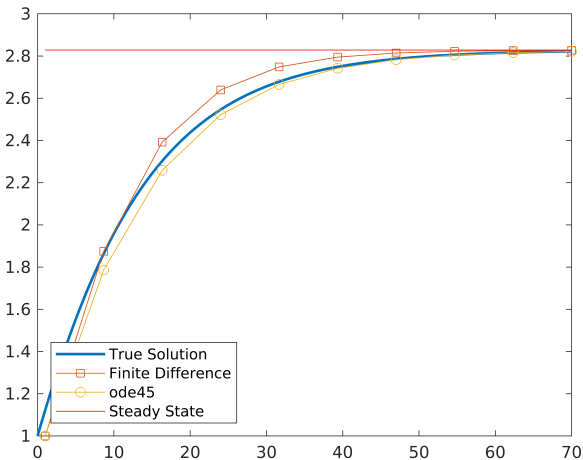


Figure 3: Trajectory for  $k(t)$

# Symbolic and Automatic Differentiation

- **Symbolic differentiation** applies the rules of derivatives to symbolic expressions. What we humans do.
- **Automatic differentiation** breaks codes into smaller sections and applies the chain rule (Kwon 2019, See Chapter 4).
- The advantage of automatic differentiation is that computes **analytical** derivatives at the same cost. No error!
- Automatic differentiation evaluates expressions numerically early, symbolic differentiation at the end.
- Thus, symbolic is more costly. Check **Matlab's documentation**

# Basics of Numerical Integration

# Numerical Integration — Why?

- Numerical evaluation of integrals in economics is ubiquitous.
  - Expectations, posteriors in Bayesian statistics, ODEs (again...)
- Again, sometimes it is computationally expensive to compute integrals.
- The definite integral of  $f(x)$  is the area under its graph.
- Furthermore, the definition of the definite integral involves an infinite sum (Riemann Sum).
- We will approximate integrals by computing sums in particular ways.

# Numerical Integration — Preliminaries

## Definition 2

Let  $f : [a, b] \mapsto \mathbb{R}$ . Let  $\mathcal{P}$  and  $\mathcal{T}$  be a **partition pair** such that  $\mathcal{P}, \mathcal{T} \subset [a, b]$  where  $\mathcal{P} = \{x_0, \dots, x_n\}$ ,  $\mathcal{T} = \{t_1, \dots, t_n\}$  and

$$a = x_0 \leq t_1 \leq x_1 \leq t_2 \leq x_2 \leq \dots \leq t_n \leq x_n = b$$

where we assume the points  $\{x_0, \dots, x_n\}$  are distinct. The **Riemann sum** corresponding to  $f, \mathcal{P}, \mathcal{T}$  is

$$\mathcal{R}(f, \mathcal{P}, \mathcal{T}) = \sum_{i=1}^n f(t_i) \Delta x_i = f(t_1) \Delta x_1 + f(t_2) \Delta x_2 + \dots + f(t_n) \Delta x_n$$

and  $\Delta x_i = x_i - x_{i-1}$ .

Notice this is just the area of the rectangles of base  $\Delta x_i$  under the curve of  $f$ .

# Numerical Integration — Preliminaries

## Definition 3

The **mesh** of the partition  $\mathcal{P}$  is the length of the largest subinterval  $[x_{i-1}, x_i]$ .

## Definition 4

A real number  $I$  is the **Riemann Integral** of  $f$  over  $[a, b]$  if it satisfies  $\forall \epsilon > 0, \exists \delta > 0$  such that if  $\mathcal{P}, \mathcal{T}$  is any partition pair, then

$$\text{mesh}(\mathcal{P}) < \delta \Rightarrow |\mathcal{R}(f, \mathcal{P}, \mathcal{T}) - I| < \epsilon$$

If such an  $I$  exists it is unique and we denote it by

$$\int_a^b f(x)dx = I = \lim_{\text{mesh}(\mathcal{P}) \rightarrow 0} \mathcal{R}(f, \mathcal{P}, \mathcal{T})$$

and we say that  $f$  is **Riemann integrable** with Riemann integral  $I$ .

# Numerical Integration — Intuition

- Notice the definition of the **Riemann integral** is just a weighted sum of the values of  $f$  at certain points.
- If the subintervals  $[x_{i-1}, x_i]$  are all the same length, the weights are all equal. But we do not need to choose equal weights.
- The problem of numerical integration is also called *cuadrature*.
- There are many quadrature methods, we will only study a couple of **Newton-Cotes formulas**. But the general idea is to use a formula such as (3)

$$\int_a^b f(x)dx \approx \sum_{i=1}^n \omega_i f(x_i) \quad (3)$$

where  $\omega_i$  are the weights, and  $x_i$  the points.



# Numerical Integration — Newton-Cotes Quadratures

## General Idea:

- Evaluate  $f(x)$  at a finite number of points.
- Construct a piece-wise polynomial approximation of  $f$  based on those points.
- Integrate the approximation of  $f$  to approximate  $\int_a^b f(x)dx$

## Numerical Integration — Newton-Cotes Quadratures

- $aUQVb \Rightarrow$  midpoint rule.
- $aPRb \Rightarrow$  trapezoid rule.
- Parabola  $PQSR$

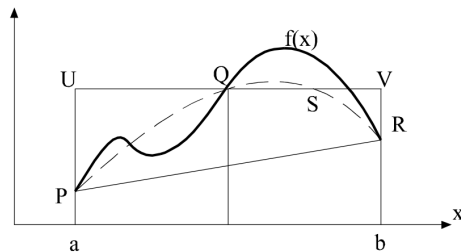


Figure 4: See Judd (1998)

# Numerical Integration — Newton Cotes Quadratures

## Midpoint Rule:

- The simplest quadrature formula with one point given by

$$\int_a^b f(x)dx = \underbrace{(b-a)f\left(\frac{a+b}{2}\right)}_{\text{Rule}} + \underbrace{\frac{(b-a)^3}{24}f''(\xi)}_{\text{Error}}$$

for some  $\xi \in [a, b]$ . Based on Taylor's theorem and the IVT.

- Let  $n \geq 1$  be the number of subintervals,  $h = (b-a)/n$ , and  $x_j = a + (j - \frac{1}{2})h$ ,  $j = 1, 2, \dots, n$ . The **composite midpoint rule** (omitting the error) is given by

$$\int_a^b f(x)dx = h \sum_{j=1}^n f(x_j) \Rightarrow \text{Same as the Riemann Sum!!}$$

# Numerical Integration — Newton-Cotes Quadratures

## Example 4

Compute  $\int_1^5 x^2 dx = \frac{124}{3} \approx 41.333$ . Errors decline with the number of points  $n$ .

```
1 function [value] =  
    midpoint_rule(a,b,n,myfunc)  
2 % Numerical integration with  
    midpoint rule  
3 xpts = zeros(n,1);  
4 h = (b-a)/n;  
5 for jj=1:n  
6     xpts(jj,1) = a + (jj -  
        1/2)*h;  
7 end  
8 value = h*sum(myfunc(xpts));  
9 end
```

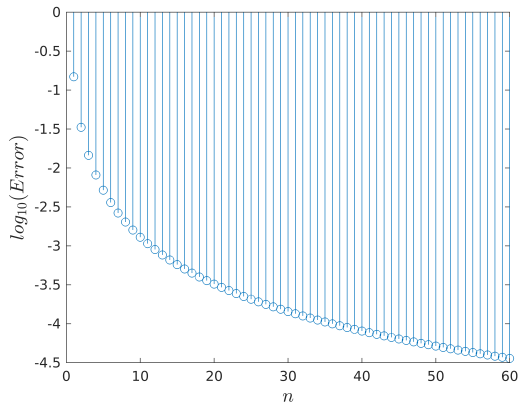


Figure 5: Relative errors in  $\log_{10}$  units

# Numerical Integration — Newton-Cotes Quadratures

## Trapezoid Rule:

- Use linear approximations of  $f$  using the end-points of  $[a, b]$ .
- The rule is

$$\int_a^b f(x)dx = \frac{b-a}{2}(f(a) + f(b)) - \frac{(b-a)^3}{12}f''(\xi)$$

for some  $\xi \in [a, b]$ .

- Composite trapezoid rule letting  $h = (b-a)/n$  and  $x_i = a + ih$

$$\int_a^b f(x)dx = \frac{h}{2} (f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n))$$

# Numerical Integration in Matlab

- As with root finding, Matlab has routines for numerical integration.
- The trapezoid rule is implemented in `trapz` (you'll work with this on the Problem Set).
- Another routine is `integral`.
- The latter uses adaptive quadratures. Basically, adapts the subintervals refining the process. But the quadrature rules can still be Newton-Cotes.
- For double and triple integrals, Matlab has `integral2` and `integral3` respectively.