Interpolation and Quadrature

Stefan Baur

14th September 2004

Efficient Algorithms for Interpolation and Quadrature - Basics and more...

Overview I

- Interpolation
  - Definition
  - Polynomial Interpolation
  - Various Approaches
    - Lagrange
    - Inductive Calculation (Neville Tableau)
    - Newton Interpolation
  - Error estimate
  - Improvements

Overview II

- Quadrature
  - Basic Idea
  - Examples
    - Trapezoidal Rule, Simpson Rule
  - More Efficent Algorithms
    - Gaussian Quadrature
  - Extrapolation
  - Adaptive
    - Archimedes Quadrature
  - Additional Methods
    - Transformation, Monte-Carlo Quadrature

Interpolation

Needed when:

- values of a function only known on some sample points
- calculate values of the function between those points

Examples:

- sine tables
- physical measurements

Definition - Interpolation

- given $n + 1$ points $x_0, \ldots, x_n$
  and coresponding values $y_i = f(x_i)$

- basic functions $g_0(x), \ldots, g_n(x)$
  (e.g. x^k, $cos(kx)$, $\ldots$)

- find coefficients $c_0, \ldots, c_n$:

$$\sum_{k=0}^{n} c_k g_k(x_j) = y_j \ (j = 0, 1, \ldots, n)$$

Polynomial Interpolation

Here we have:

- given $n + 1$ points (x_i, y_i) $(i = 0, \ldots, n)$
- find polynomial $p(x)$ of degree n with:
  $p(x_i) = y_i$ for $i = 0, \ldots, n$

Approach

Use basic functions x^k with coefficients c_k:

$$\sum_{k=0}^{n} c_k x^k$$

leads to linear equation:

$$\begin{pmatrix} 1 & x_0^1 & \dots & x_0^n \\ \dots & & & \dots \\ 1 & x_n^1 & \dots & x_n^n \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ \dots \\ c_n \end{pmatrix} = \begin{pmatrix} y_0 \\ \dots \\ y_n \end{pmatrix}$$

Solution?

Solve linear equation. . .

Easier: take Lagrange's classical formula:

$$p(x) = \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)}y_0 +$$

$$\frac{(x-x_0)(x-x_2)\dots(x-x_n)}{(x_1-x_0)(x_1-x_2)\dots(x_1-x_n)}y_1 + \dots$$

$$\dots + \frac{(x-x_0)(x-x_1)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\dots(x_n-x_{n-1})}y_n$$

each term constructed to be zero for all x_i except one, for which it is y.

Lagrange Polynomials

We define the Lagrange Polynomials L_j:

$$L_j := \prod_{i=0, i\neq j}^{n} \frac{x - x_i}{x_j - x_i}$$

now we have $L_j(x_j) = 1$ and $L_j(x_i) = 0$ for $i \neq j$

$$\Rightarrow p(x) = \sum_{i=0}^{n} f(x_i) \cdot L_i(x)$$

This makes $p(x)$ a polynomial of degree n with
$p(x_i) = f(x_i) = y_i$ □

One solution - ambiguous solutions?

- let $p_1(x)$ and $p_2(x)$ be polynomials of degree $\leq n$ through the $n+1$ points.
- now $q(x) := p_1(x) - p_2(x)$ is also a polynomial of degree $\leq n$ and is zero for all x_i ($n+1$ points).
- a nonzero polynomial of degree $\leq n$ can only have n points where it is zero. Therefore $q(x)$ has to be identical to 0. $\Rightarrow p_1 \equiv p_2$

So if there is a solution, it is the only one. $\square$

Make things easier

Lagrange polynomials

✔ always find a solution

✘ expensive calculations

✘ not flexible when changing sample points

✘ not easy to analyze rounding errors

We search for better ways to calculate the solution

Inductive Calculation

- $p_{i,l}(x)$: interpolation polynomial for the points $x_i, \ldots, x_{i+l}$:

then we get the recursive definition:

$$p_{i,k}(x) = p_{i,k-1}(x) \cdot \frac{x_{i+k} - x}{x_{i+k} - x_i} + p_{i+1,k-1}(x) \cdot \frac{x - x_i}{x_{i+k} - x_i}$$

$$p_{i,0}(x) = y_i$$

->proof: verify $p(x_i) = y_i$

Neville Tableau

Neville Tableau

- visualizes construction:

$$
\begin{array}{ccccccc}
p_{0,0} & & p_{1,0} & & p_{2,0} & & p_{3,0} \\
 & \backslash & / \quad \backslash & & / \quad \backslash & & / \\
 & p_{0,1} & & p_{1,1} & & p_{2,1} & \\
 & & \backslash & / \quad \backslash & / & & \\
 & & p_{0,2} & & p_{1,2} & & \\
 & & & \backslash \quad / & & & \\
 & & & p_{0,3} & & &
\end{array}
$$

->example

Algorithm

this leads to a $O(n^2)$ algorithm:

for i=0,..,n do
 $p_{i,0} = y_i$
od
for k=1,..,n do
 for i=0,..,n-k do
  $p_{i,k} = p_{i,k-1} \cdot \frac{x_{i+k}-x}{x_{i+k}-x_i} + p_{i+1,k-1} \cdot \frac{x-x_i}{x_{i+k}-x_i}$
 od
od

Faster Ways?

Faster ways, if we need $p(x)$ on different points x?

Yes: Newton Interpolation

- usual polynomial:
  $p(x) = c_n x^n + \cdots + c_1 x + c_0$

- transform to:
  $p(x) = (\ldots((c_n x + c_{n-1}) \cdot x + c_{n-2}) \cdot x + \cdots + c_1) \cdot x + c_0$

Faster Ways?

The expression

$$p(x) = (\ldots((c_n x + c_{n-1}) \cdot x + c_{n-2}) \cdot x + \cdots + c_1) \cdot x + c_0$$

can now be calculated with:

```
y=c[n];
for j=n-1,..,0 do
 y=y*x+c[j];
od
```

$\Rightarrow O(n)$ algorithm!

Newton Interpolation

Newton Interpolation consists of 2 steps:

- calculate coefficients (once)
- then use $O(n)$ algorithm to calculate values of $p(x)$

$\rightarrow$How to obtain the coefficients?

Newton Interpolation

We start with $p_{i,k}(x) = a_{i,k}x^k + b_{i,k}x^{k-1} + \ldots$
in the recursion
$p_{i,k}(x) = p_{i,k-1}(x) \cdot \frac{x_{i+k}-x}{x_{i+k}-x_i} + p_{i+1,k-1}(x) \cdot \frac{x-x_i}{x_{i+k}-x_i}$
gives for $a_{i,k}$:

$$a_{i,k} = \frac{a_{i+1,k-1} - a_{i,k-1}}{x_{i+k} - x_i} \quad \text{and} \quad a_{i,0} = y_i$$

- recursion similar to the one above (Neville Tableau)!
- it contains all necessary information!

->proof: coefficient comparison

Newton Interpolation

Now the expression

$$p(x) = a_{0,0} + a_{0,1} \cdot (x - x_0) + \cdots + a_{0,n} \cdot (x - x_0) \cdot \ldots (x - x_{n-1})$$

can be transformed to:

$$p(x) = (\ldots (a_{0,n} \cdot (x - x_{n-1}) + a_{0,n-1} + \ldots) \cdot (x - x_0) + a_{0,0}$$

which leads to the above $O(n)$ algorithm. □

How good are those interpolations?

We can estimate the error $f(x) - p(x)$ for smooth functions ($n+1$th derivative of f exists) as:

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0) \cdot \dots \cdot (x - x_n)$$

where $f^{(n+1)}(\xi)$ is the $n+1$th derivative of f at a point $\xi \in [min(x_i), max(x_i)]$

->proof: using mean value theorem

Error Estimate

now if there is a maximum of $f^{(n+1)}$ we can set:

$$M_{n+1} = max_{x \in [a,b]} |f^{(n+1)}(x)| < \infty$$

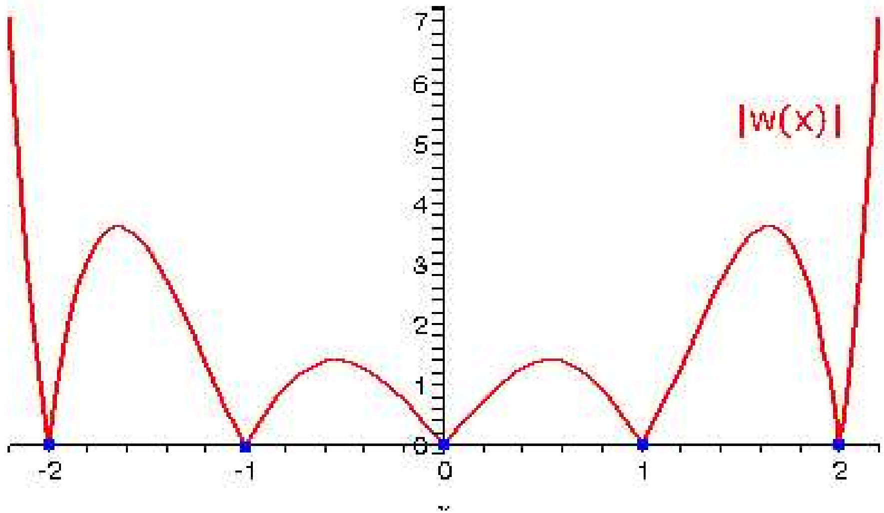
$$w(x) = |x - x_0| \cdot \dots \cdot |x - x_n|$$

then

$$\Rightarrow |f(x) - p(x)| \leq \frac{M_{n+1}}{(n+1)!} |w(x)|$$

Error Curve $|w(x)|$

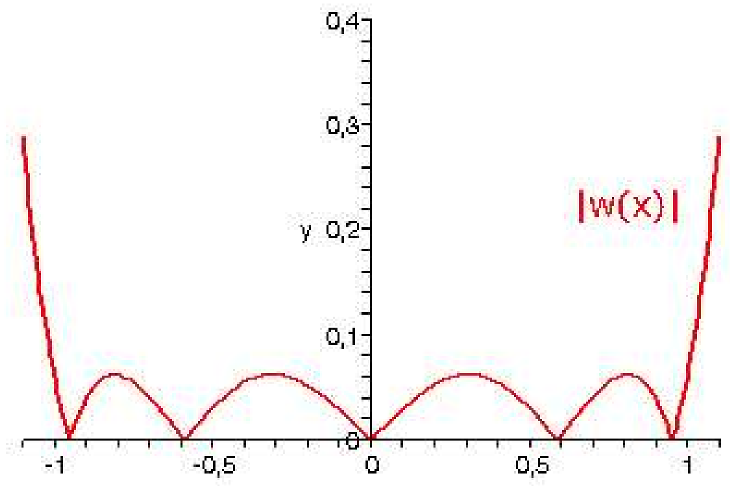
with $x_0 = -2$, $x_1 = -1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 2$



good in the middle - bad at borders

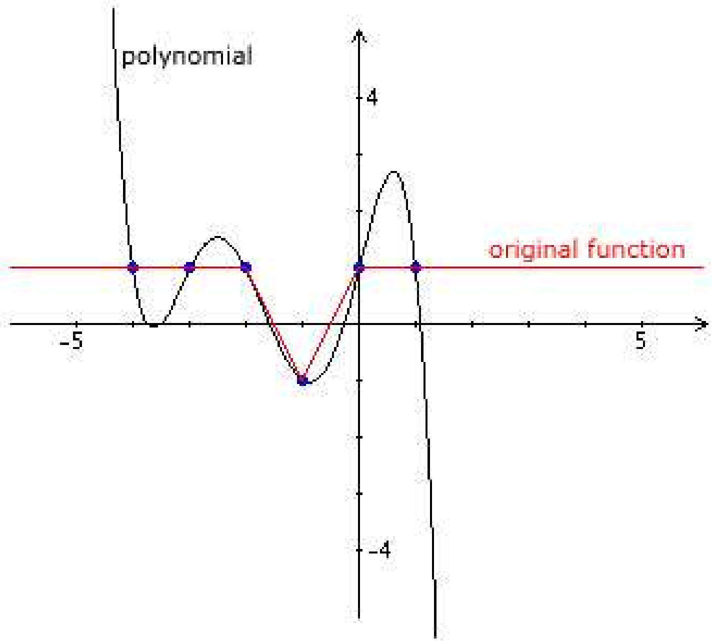
Improvements

- non-equidistant points x_i
  e.g. Chebycheff distributed for the interval $[-1, 1]$:
  $x_j = cos(\frac{(2j+1)\pi}{2n+2})$



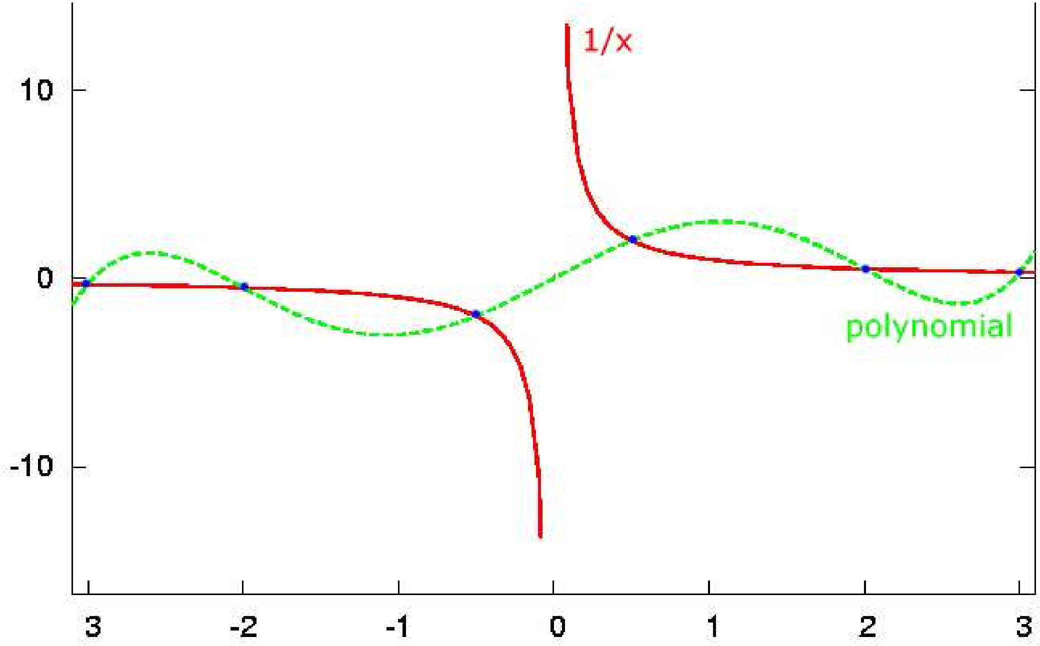
Is this already what we want?

Problem: Bad on sharp edged functions



Is this already what we want?

Problem: Singularities



Now, what to do?

- Choose other base functions:
  - Rational Functions
  - Trigonometric Functions -> Fourier Methods
- Hermite Interpolation
  (not only $y_i = f(x_i)$ but also $f'(x_i)$)
- Spline Interpolation
- ...

Enough Interpolation

Ok,
now we are ready for
Quadrature

Quadrature

Quadrature = numerical integration
calculating $\int_a^b f(x)dx$

Goals:

- as accurate as possible
- with only few function evaluations $f(x)$

Basic Idea

1. take $n+1$ points out of $[a,b]$
   (e.g. $x_i = \frac{i \cdot a + (n-i) \cdot b}{n}$ gives constant steps
   $h = x_i - x_{i-1} = \frac{1}{n}$)

2. interpolate $f(x)$ with a polynomial $p(x)$ through these sample points.

3. integrate $p(x)$:

$$\int_a^b f(x)dx \approx \int_a^b p(x)dx$$

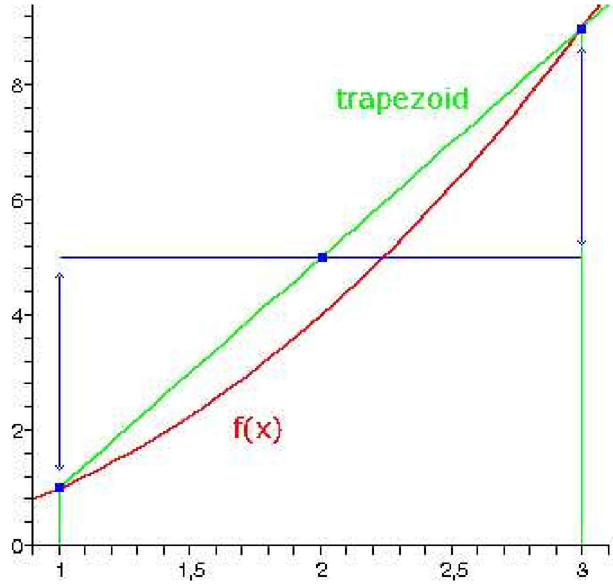
With Lagrange Polynomials

$$\int_a^b p(x)dx = \sum_{i=0}^{n} f(x_i) \cdot \int_a^b L_i(x)dx$$

Examples

Trapezoidal Rule $(n = 1)$

$$\int_a^b f(x)dx \approx (b-a) \cdot \left[\frac{1}{2}f(a) + \frac{1}{2}f(b)\right]$$



Error Estimate

From Interpolation:

$$|f(x) - p(x)| \leq \frac{M_{n+1}}{(n+1)!}|w(x)|$$

So we get for the error

$$\left|\int_a^b f(x) - p(x)dx\right|$$

$$\leq \frac{M_2}{2!}\int_a^b (x-a)(x-b)dx$$

$$= \frac{M_2}{12}(b-a)^3$$

Examples

Simpson-Rule $(n = 2)$

$$\int_a^b f(x)dx \approx \frac{b-a}{6} \cdot \left[f(a) + 4f(\frac{a+b}{2}) + f(b) \right]$$

and

$$\left| \int_a^b f(x) - p(x)dx \right| \leq \frac{M_4(b-a)^5}{2880}$$

Exact for polynomials of degree 3!
(due to good distribution of the sample points)

More Efficient Algorithms

Gaussian-Quadrature

non-equidistant points for better results
(see Interpolation)

Pro's - Con's

✔ better results with same number of points

✘ no reuse of values when increasing number of points

Extrapolation

To avoid problems with high polynomial degree

- divide $[a, b]$ into n parts
- use simple (trapezoidal) rule on each part

$h = (b - a)/n$, $x_i = a + ih$

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx$$

$$\approx \sum_{i=0}^{n-1} \frac{h}{2}(f(x_i) + f(x_{i+1})) =: T(h)$$

Extrapolation

$\Rightarrow T(h)$ trapezoidal sums as a function of h

Obviously $T(h) \to \int_a^b f(x)dx$ for $h \to 0$

So the goal is to calculate $T(0)$

Deeper analysis gives (Euler-McLaurin formula):

$$T(h) = \int_a^b f(x)dx + \sum_{k=1}^{N} c_k h^{2k} + O(h^{2N+2})$$

Extrapolation

Idea:

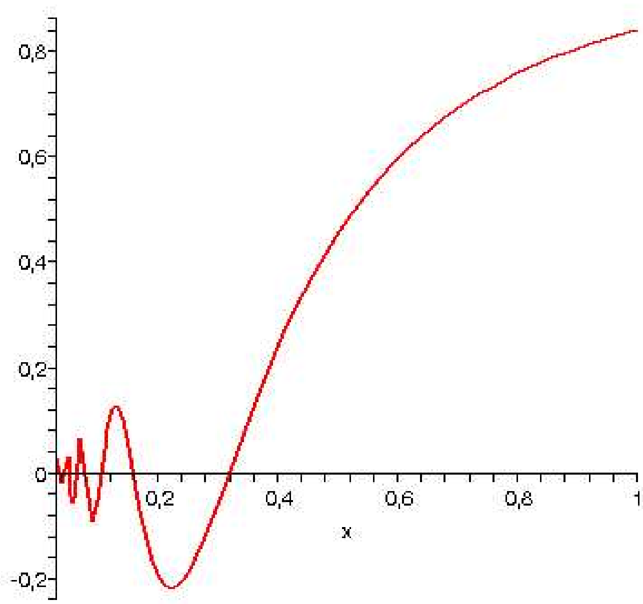
- interpret $T(h)$ as function in h^2
- interpolate $T(h)$ for sample points $h_0^2, h_1^2, \ldots, h_m^2$ $(h_i \to 0)$ with polynomial $\hat{T}(h)$
- evaluate $\hat{T}(0)$ as approximation of $T(0)$

$\Rightarrow$ error in $O(h_0^2 \cdot h_1^2 \cdot \cdots \cdot h_m^2)$

- good choice for (h_i): $h_i = (b-a)/n_i$, $n_i = 2n_{i-1}$ (makes reuse of function evaluations possible)

Adaptive Methods

Often hard to know good choice of sample points



⇒ increase number of sample points when needed

Archimedes Quadrature

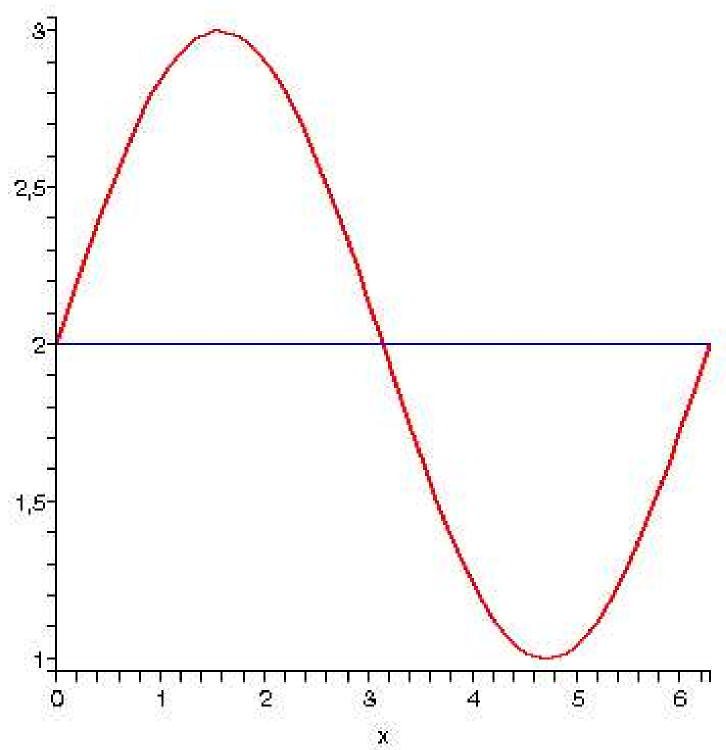
1. linear interpolation in $[a, b]$

2. check for differences

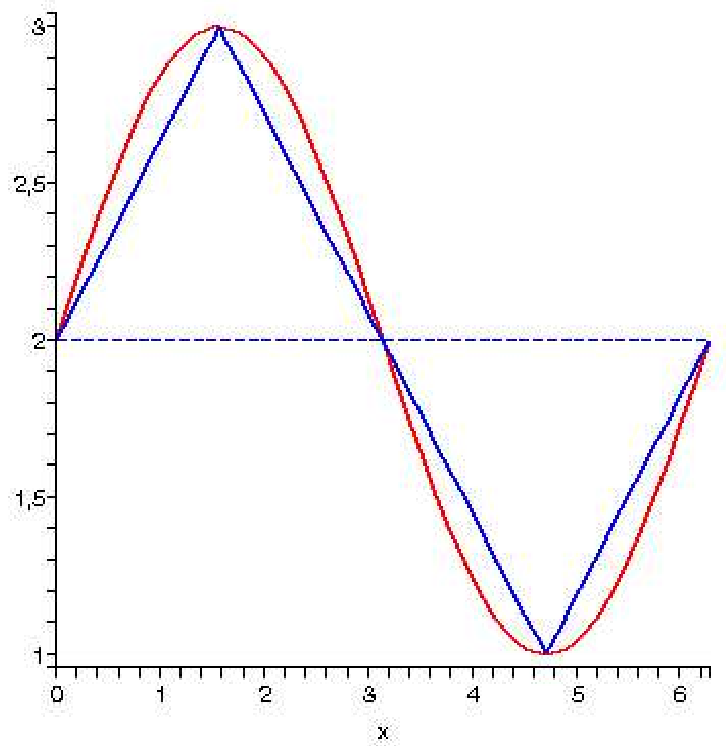
   ✔ small enough:
   - finish

   ✘ still too big:
   - divide interval into 2 parts
   - continue 1. linear interpolation for each part

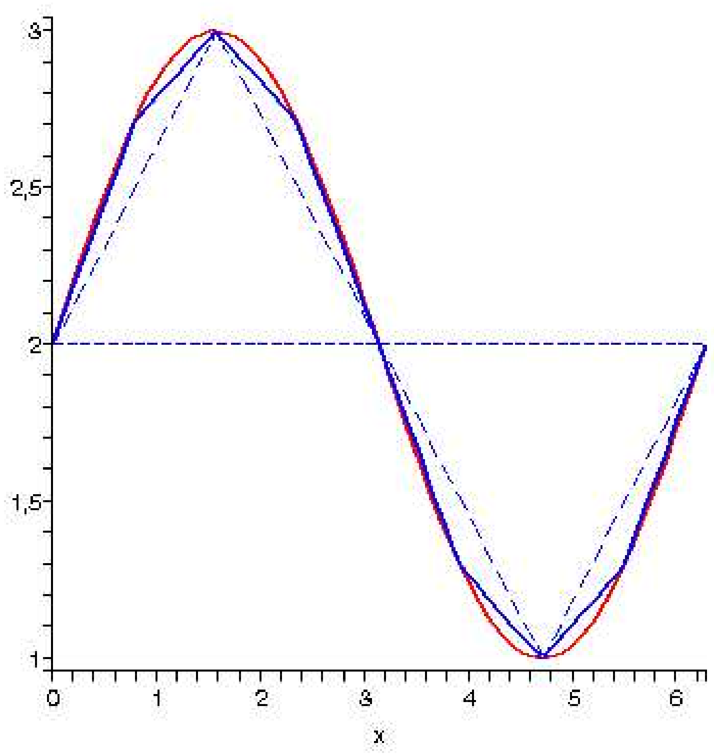
Archimedes - Example (step I)



Archimedes - Example (step II)



Archimedes - Example (step III)



Additional Methods

Sometimes good to transform $[a, b]$ to standard interval $[-1, 1]$ or $[0, 1]$

- set $x = g(y) = a + (b - a)y$
- $\Rightarrow g'(y) = b - a$, $y = \frac{x-a}{b-a}$

$$\int_a^b f(x)dx = \int_0^1 f(g(y))g'(y)dy$$

$$= (b - a)\int_0^1 f(a + (b - a)y)dy$$

Additional Methods

Especially useful when integrating to ∞:
transform $[1, \infty]$ using $x = \frac{1}{y}$:

$$\int_1^\infty f(x)dx = \int_1^0 \frac{1}{y^2} f(\frac{1}{y})dy$$

Other approach for $[a, \infty]$:

- calculate $[a, b]$ for big b
  as $\int_b^\infty f(x)dx \to 0$ for $b \to \infty$ if $\int_a^\infty f(x)dx$ exists

Monte Carlo

Completely different approach:

- choose random $x_i \in [a, b]$ $(i = 1, \ldots, n)$
- calculate $f(x_i)$
- weight each x_i with $(b - a)/n$

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \frac{f(x_i)}{n} \cdot (b - a)$$

Thank You

the end

...