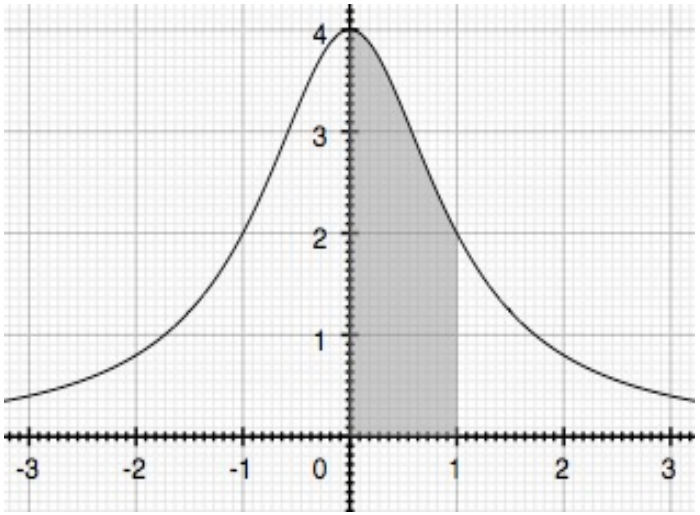
An example: computing the value of Pi

- One way to approximate the value of the constant *Pi* is to compute the area between 0 and 1 of the function $f(x) = \frac{4}{1+x^2}$

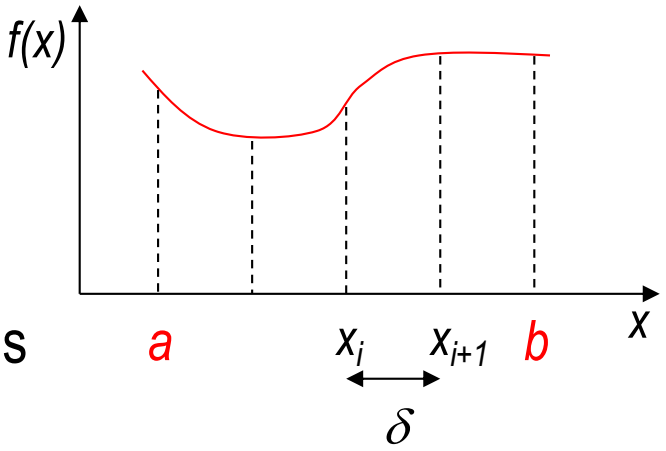


- $\int_0^1 \frac{4}{1+x^2} = \pi$
- The integral can be approximated by dividing the area under the curve into a number of intervals and calculating the area of each interval

Numerical integration

- Compute the area under a curve *f(x)*

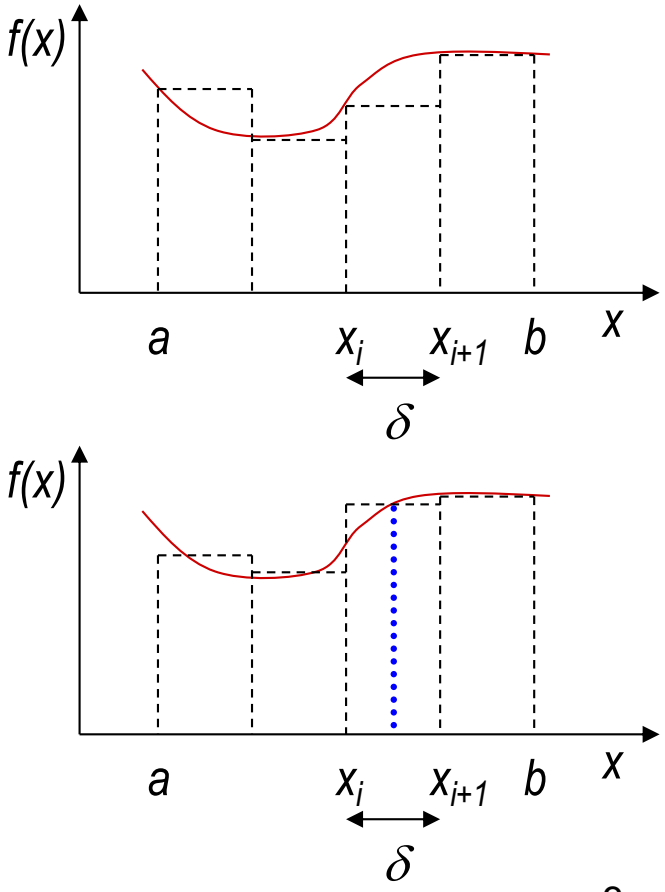
$$\int_a^b f(x)dx$$



- Divide the area between *a* and *b* into intervals of width δ
  - compute the area of each interval
  - sum the areas of all intervals
- The accuracy of the solution increases if we use a smaller δ
  - that is, if we use a large number of intervals

Using rectangles

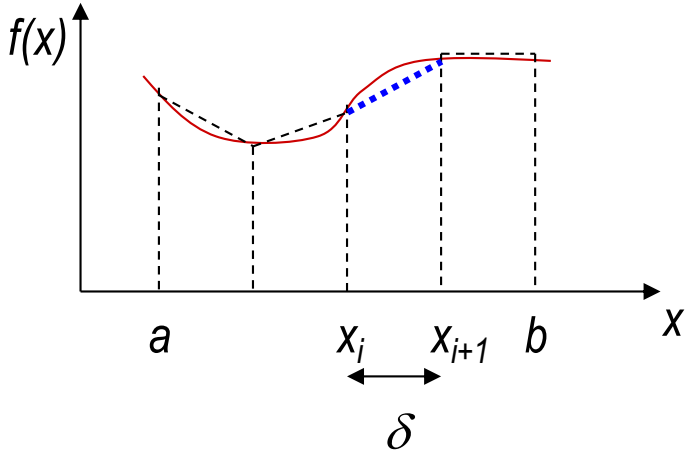
- We can approximate the area under the curve *f(x)* using rectangles of width $\delta=(x_{i+1} - x_i)$ whose height is such that their left upper point lies on the curve *f(x)*
  - the area of rectangle *i* is $\delta * f(x_i)$
- Choosing the left point as the height of the rectangles introduces a bias
  - same problem if we choose the right point
- We get a better approximation if we choose the midpoint $(x_i+x_{i+1})/2$ to compute the height of the rectangle



Using trapezoids

- We get an even better approximation if we use a trapezoid to compute the area of an interval
  - we approximate the function *f(x)* on the interval by a straight line from $(x_i , f(x_i))$ to $(x_{i+1} , f(x_{i+1}))$
  - the area of trapezoid *i* is then

$$\frac{\delta}{2}\left(f\left(x_i\right) + f\left(x_{i+1}\right)\right)$$



  - this can also be written as *0.5 * (f(x)+f(x+*δ*))*

Sequential code

Using rectangles and midpoint

```
n is the number of intervals
d = 1.0/n;

for (i=0; i<n; i++) {
    x = d*(i+0.5);
    sum +=f(x);
  }
pi = d*sum;
```

Using trapezoids

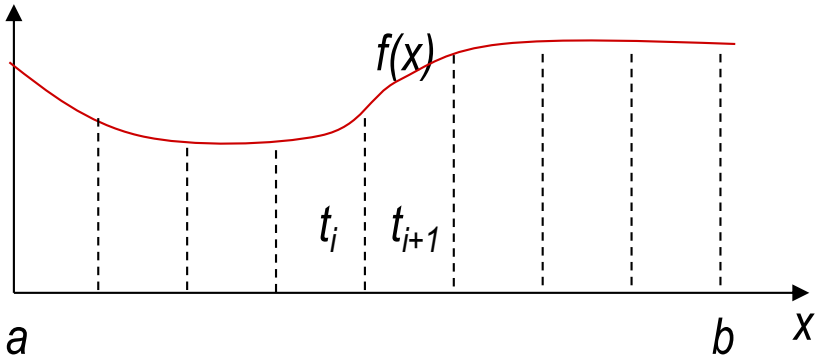
```
n is the number of intervals
d = 1.0/n;

for (i=0; i<n; i++) {
  x = d*i;
  sum += f(x) + f(x+d);
}
pi = d*sum*0.5;
```

The multiplication with *d* and division by 2 can be moved out of the loop

Parallel implementation with OpenMP

- The iterations of the for-loop are divided between the threads
  - a thread computes the sum of the areas for the trapezoids that it is responsible for
  - the *sum* and the function argument *x* are local variables
  - the number of intervals *n*, and the interval length *d*, are shared
- When the area of all trapezoids have been computed, the private variable *sum* in all threads must be added together
  - this is done by declaring *sum* to be a reduction variable



Åbo Akademi University

OpenMP implementation

```
n is the number of intervals
d = 1.0/(double)n;

#pragma omp parallel default(shared) private(nthreads, tid, x)
{
   ... print nr of threads, etc ...

#pragma omp for reduction(+:sum)
   for (i=0; i<n; i++) {
       x = d*(double)i;
       sum += f(x) + f(x+d);
   }
} /* End of parallel region */

pi = d*sum*0.5;
... print the value of Pi
```