IMPRS - BBL

Numerical methods

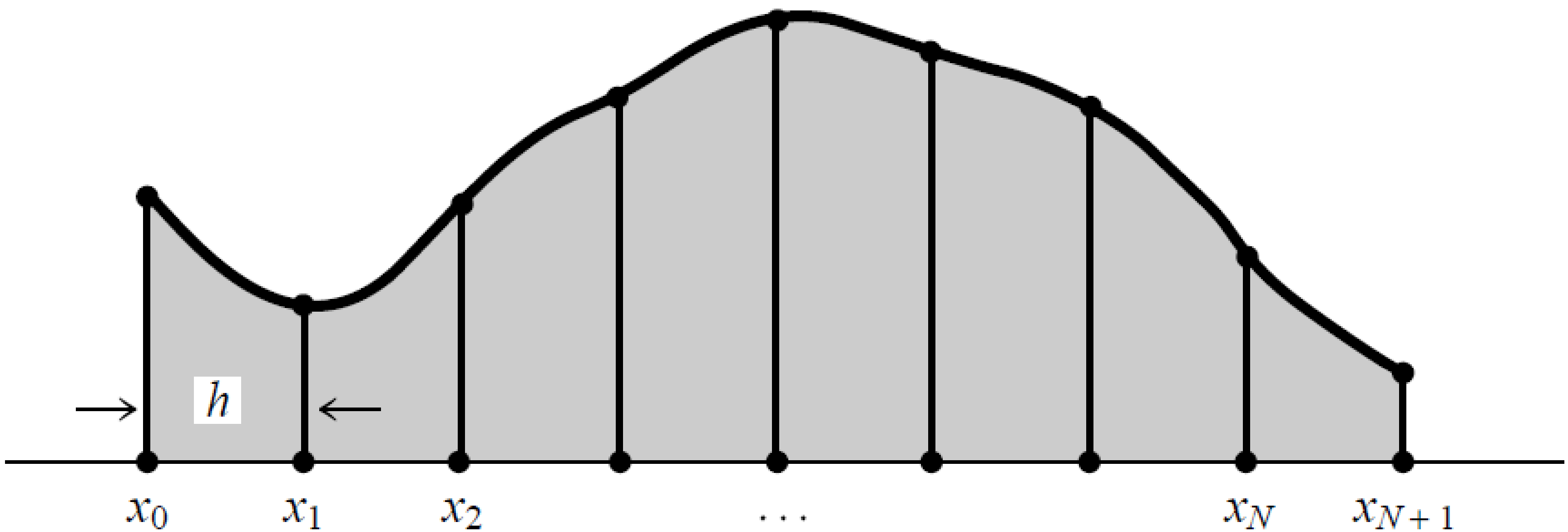
Lecture 3

Michael Marks

Topics:

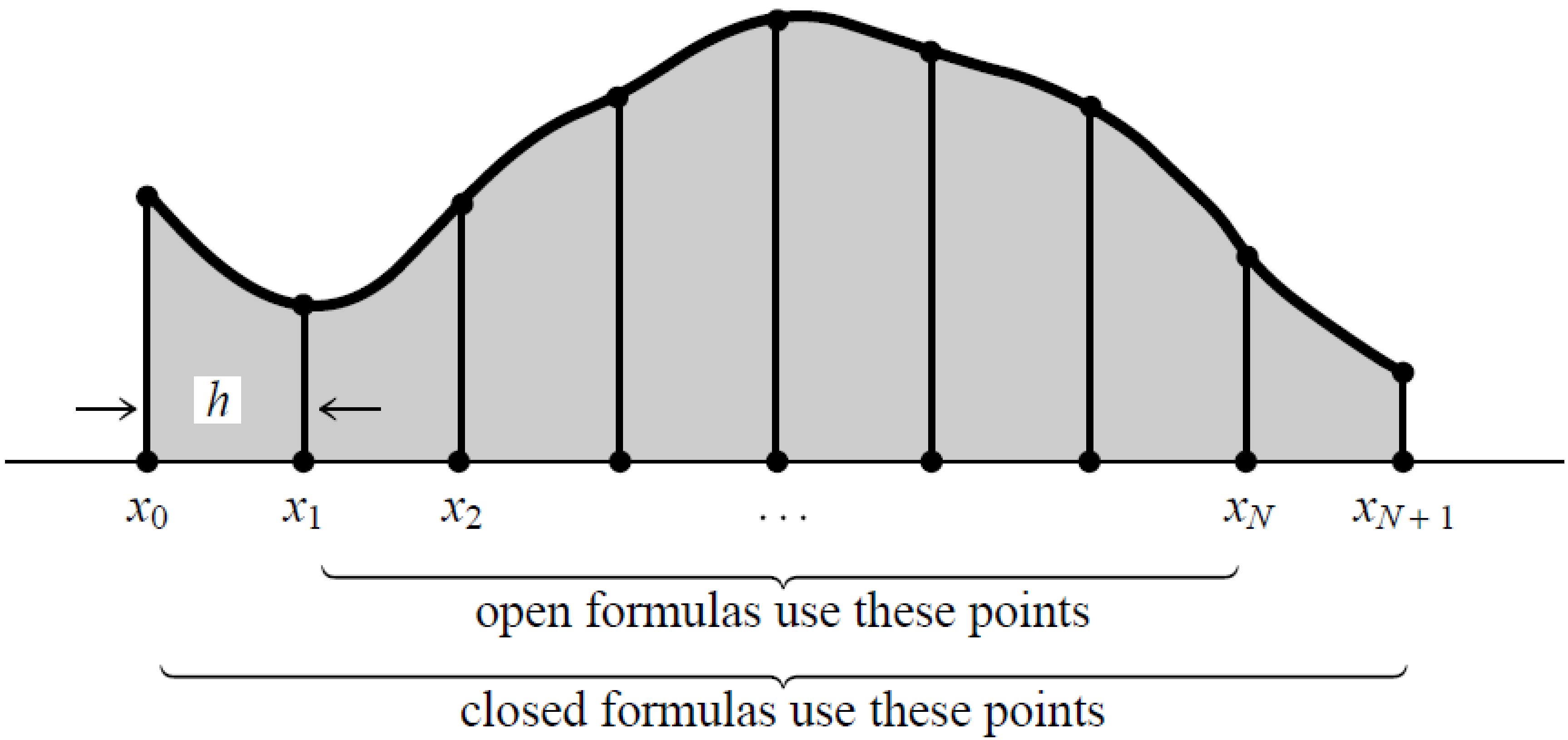
Day 1: Linear algebraic equations
Day 2: Inter- and Extrapolation
Day 3: Integration
Day 4: Random numbers and distribution functions
Day 5: Root finding, Minimization and Maximization
Day 6: Differentiation

Integration



Numerical Recipes (Press et al.)

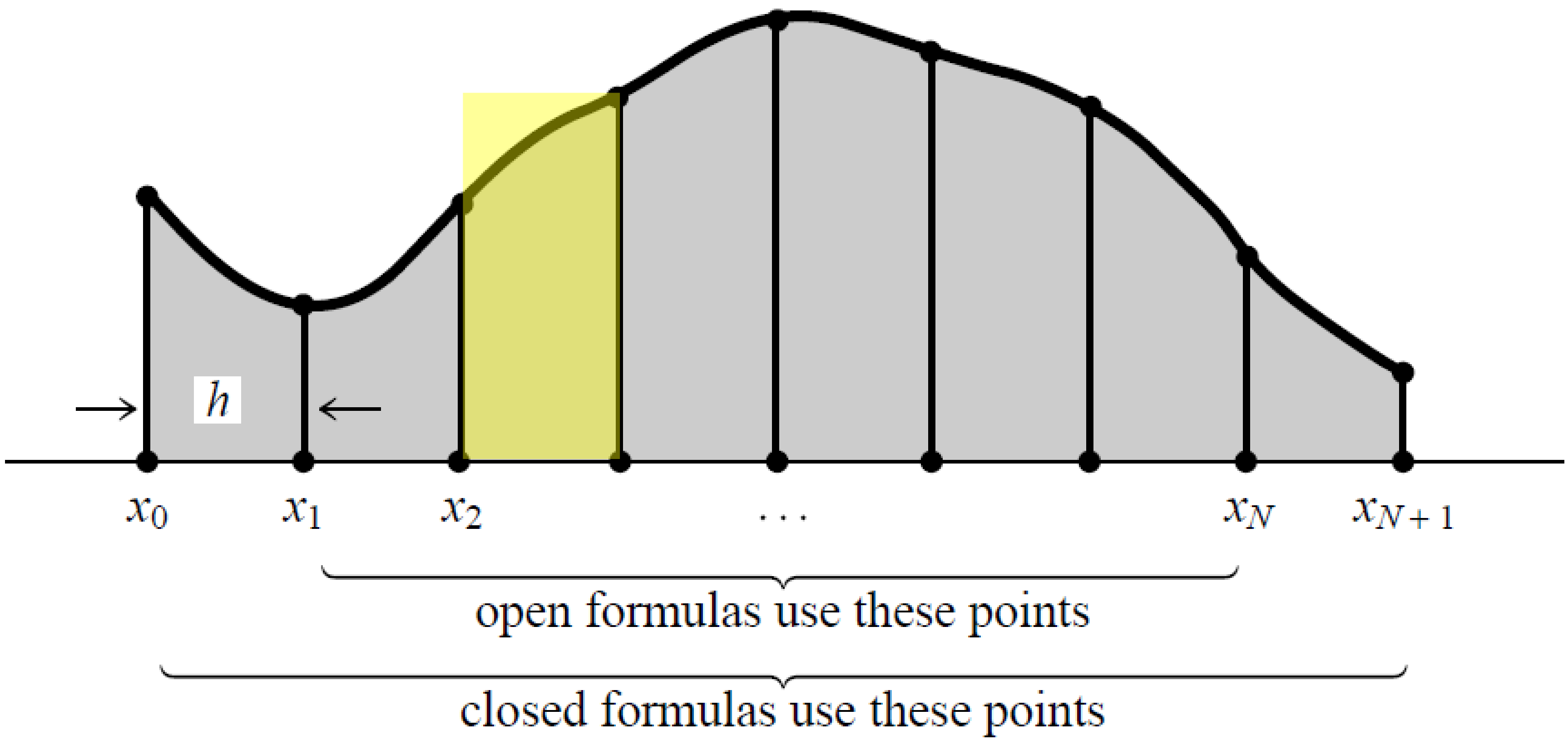
Integration



Numerical Recipes (Press et al.)

Integration

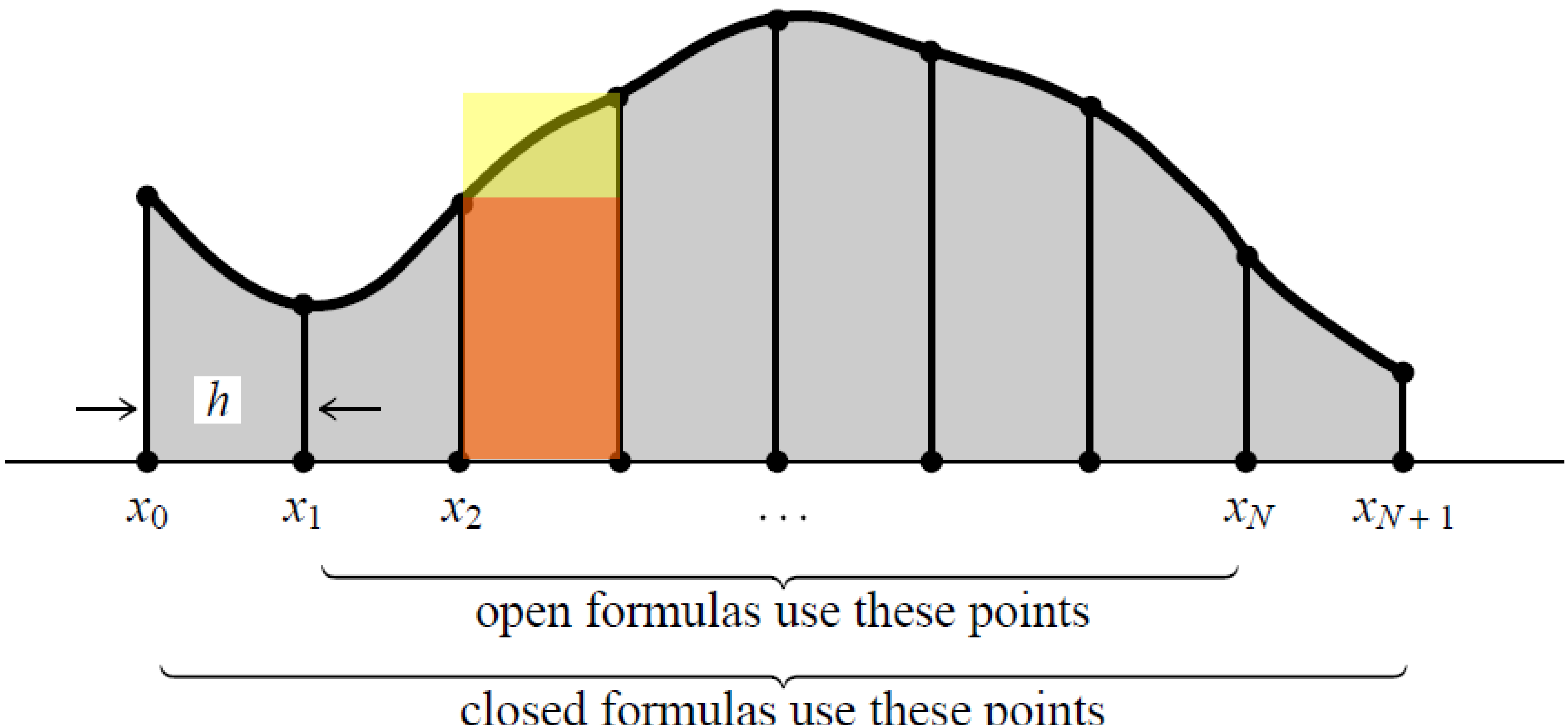
Trapezoidal rule



Numerical Recipes (Press et al.)

Integration

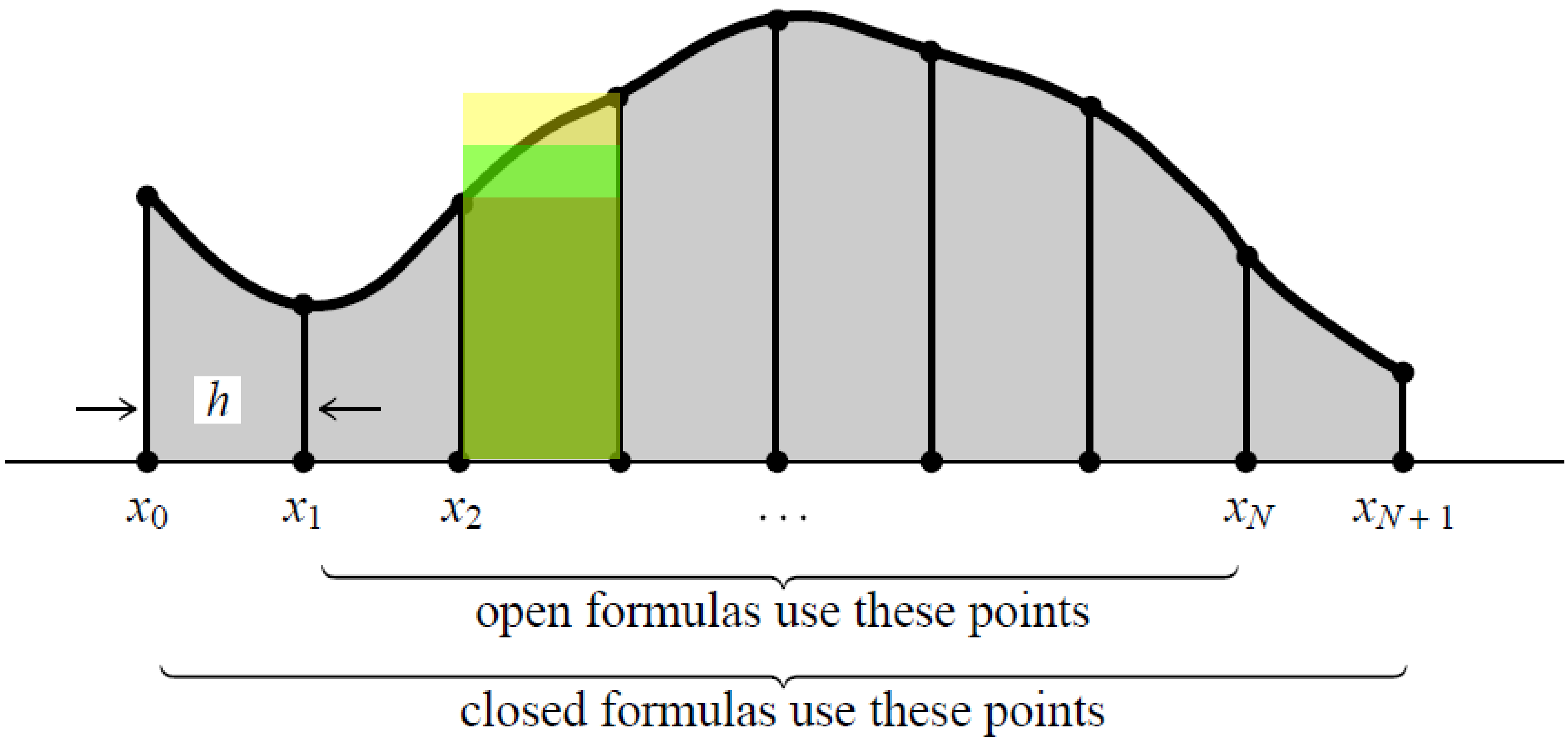
Trapezoidal rule



Numerical Recipes (Press et al.)

Integration

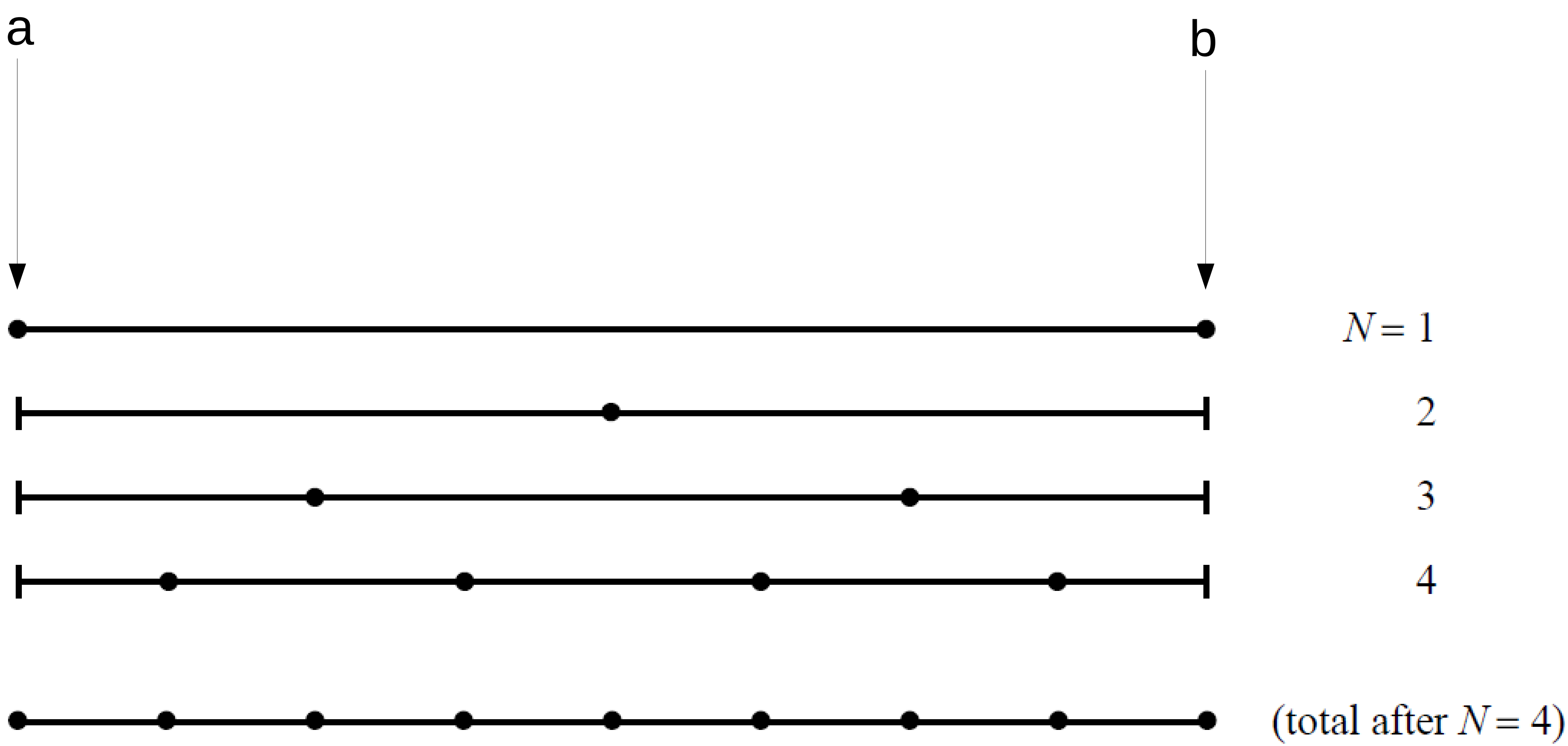
Trapezoidal rule



Numerical Recipes (Press et al.)

Integration

Trapezoidal rule



Numerical recipes (Press et al.)

Integration

Trapezoidal rule in C

```
#define FUNC(x) ((*func)(x))

float trapzd(float (*func)(float), float a, float b, int n)
```

This routine computes the `n`th stage of refinement of an extended trapezoidal rule. `func` is input as a pointer to the function to be integrated between limits `a` and `b`, also input. When called with `n`=1, the routine returns the crudest estimate of $\int_a^b f(x)dx$. Subsequent calls with `n`=2,3,... (in that sequential order) will improve the accuracy by adding $2^{\mathtt{n}-2}$ additional interior points.

```
{
    float x,tnm,sum,del;
    static float s;
    int it,j;

    if (n == 1) {
        return (s=0.5*(b-a)*(FUNC(a)+FUNC(b)));
    } else {
        for (it=1,j=1;j<n-1;j++) it <<= 1;
        tnm=it;
        del=(b-a)/tnm;                          This is the spacing of the points to be added.
        x=a+0.5*del;
        for (sum=0.0,j=1;j<=it;j++,x+=del) sum += FUNC(x);
        s=0.5*(s+(b-a)*sum/tnm);                This replaces s by its refined value.
        return s;
    }
}
```

Numerical recipes (Press et al.)

Integration

Trapezoidal rule

```
#define FUNC(x) ((*func)(x))

float trapzd(float (*func)(float), float a, float b, int n)
```

This routine computes the `n`th stage of refinement of an extended trapezoidal rule. `func` is input as a pointer to the function to be integrated between limits `a` and `b`, also input. When called with `n`=1, the routine returns the crudest estimate of $\int_a^b f(x)dx$. Subsequent calls with `n`=2,3,... (in that sequential order) will improve the accuracy by adding $2^{\mathtt{n}-2}$ additional interior points.

```
{
    float x,tnm,sum,del;
    static float s;
```

```
for(j=1;j<=m+1;j++) s=trapzd(func,a,b,j);
```

```
        return (s=0.5*(b-a)*(FUNC(a)+FUNC(b)));
    } else {
        for (it=1,j=1;j<n-1;j++) it <<= 1;
        tnm=it;
        del=(b-a)/tnm;                          This is the spacing of the points to be added.
        x=a+0.5*del;
        for (sum=0.0,j=1;j<=it;j++,x+=del) sum += FUNC(x);
        s=0.5*(s+(b-a)*sum/tnm);                This replaces s by its refined value.
        return s;
    }
}
```

Numerical recipes (Press et al.)

Integration

Trapezoidal rule

```
#include <math.h>
#define EPS 1.0e-5
#define JMAX 20

float qtrap(float (*func)(float), float a, float b)
```

Returns the integral of the function `func` from `a` to `b`. The parameters `EPS` can be set to the desired fractional accuracy and `JMAX` so that 2 to the power `JMAX-1` is the maximum allowed number of steps. Integration is performed by the trapezoidal rule.

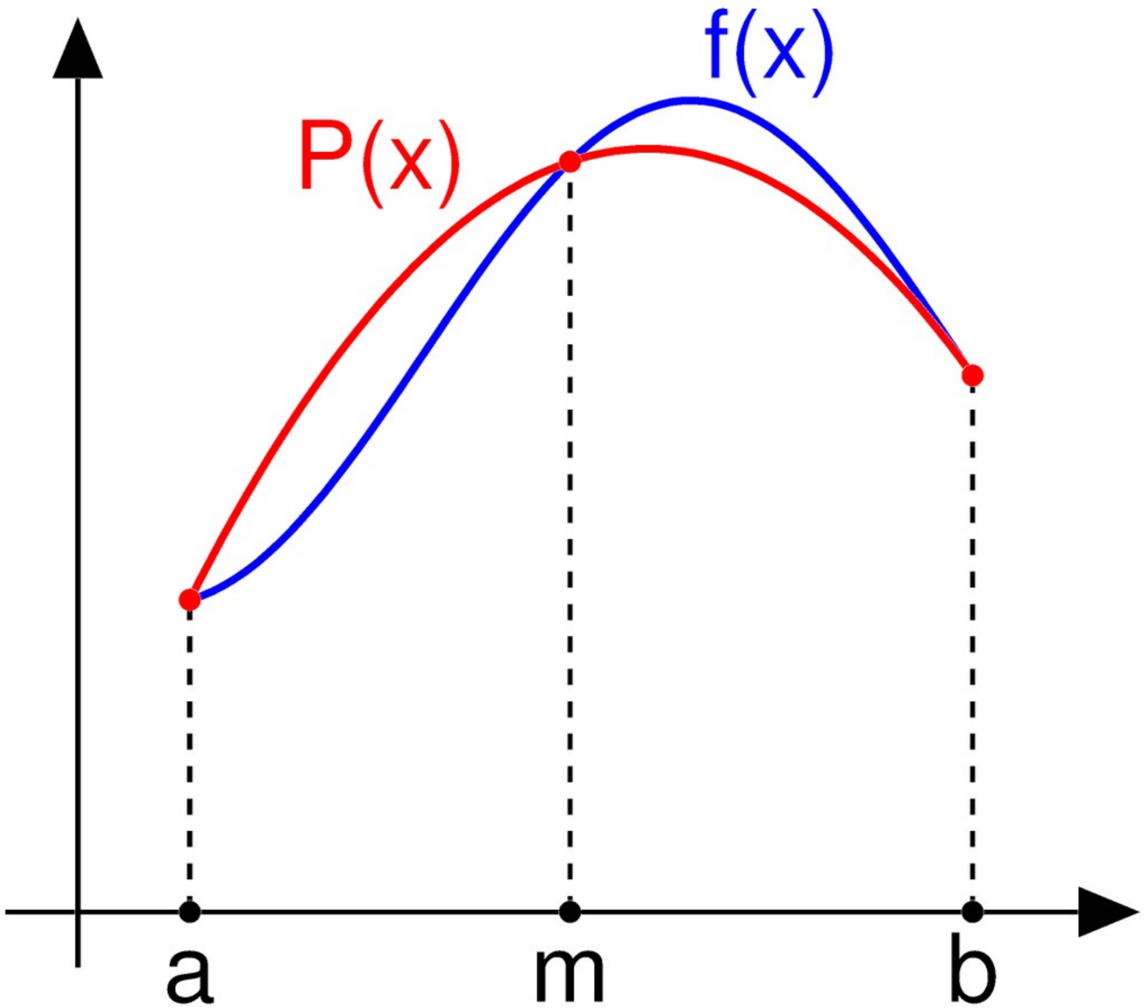
```
{
    float trapzd(float (*func)(float), float a, float b, int n);
    void nrerror(char error_text[]);
    int j;
    float s,olds=0.0;              Initial value of olds is arbitrary.

    for (j=1;j<=JMAX;j++) {
        s=trapzd(func,a,b,j);
        if (j > 5)                 Avoid spurious early convergence.
            if (fabs(s-olds) < EPS*fabs(olds) ||
                (s == 0.0 && olds == 0.0)) return s;
        olds=s;
    }
    nrerror("Too many steps in routine qtrap");
    return 0.0;                    Never get here.
}
```

Numerical recipes (Press et al.)

Integration

Simpson's rule



Wikipedia

Integration

Simpson's rule

```
/* Program int_simps - Numerical integration with Simpsons rule */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

#define NSTEP 10
#define PI 3.1415927

#define xa 0
#define xe PI

double func(double x) {
 return(sin(x));
}

int main() {
  int i;
  double xlow,xhigh,area;

  area = 0.0;

  for (i=0;i<NSTEP;i++) {
   xlow = xa+i*(xe-xa)/NSTEP;
   xhigh = xa+(i+1)*(xe-xa)/NSTEP;

   area+= (func(xlow)+4*func((xlow+xhigh)/2.0)+func(xhigh))/3.0;
  }
  area*=(xe-xa)/NSTEP/2.0;

  printf("Value of integral: %lf\n",area);
}
```

Newton-Cotes formulas

Open formulas

Trapezium rule:

$$\int_{x_1}^{x_2} f(x)dx = h\left[\frac{1}{2}f_1 + \frac{1}{2}f_2\right] \quad + O(h^3 f'')$$

Simpson's 1/3-rule:

$$\int_{x_1}^{x_3} f(x)dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right] \quad + O(h^5 f^{(4)})$$

Simpson's 3/8-rule:

$$\int_{x_1}^{x_4} f(x)dx = h\left[\frac{3}{8}f_1 + \frac{9}{8}f_2 + \frac{9}{8}f_3 + \frac{3}{8}f_4\right] \quad + O(h^5 f^{(4)})$$

Bode's rule:

$$\int_{x_1}^{x_5} f(x)dx = h\left[\frac{14}{45}f_1 + \frac{64}{45}f_2 + \frac{24}{45}f_3 + \frac{64}{45}f_4 + \frac{14}{45}f_5\right] \quad + O(h^7 f^{(6)})$$

Integration

Romberg Integration

```
#include <math.h>
#define EPS 1.0e-6
#define JMAX 20
#define JMAXP (JMAX+1)
#define K 5
```

Here EPS is the fractional accuracy desired, as determined by the extrapolation error estimate; JMAX limits the total number of steps; K is the number of points used in the extrapolation.

```
float qromb(float (*func)(float), float a, float b)
```

Returns the integral of the function func from a to b. Integration is performed by Romberg's method of order 2K, where, e.g., K=2 is Simpson's rule.

```
{
    void polint(float xa[], float ya[], int n, float x, float *y, float *dy);
    float trapzd(float (*func)(float), float a, float b, int n);
    void nrerror(char error_text[]);
    float ss,dss;
    float s[JMAXP],h[JMAXP+1];          These store the successive trapezoidal approxi-
    int j;                                  mations and their relative stepsizes.

    h[1]=1.0;
    for (j=1;j<=JMAX;j++) {
        s[j]=trapzd(func,a,b,j);
        if (j >= K) {
            polint(&h[j-K],&s[j-K],K,0.0,&ss,&dss);
            if (fabs(dss) <= EPS*fabs(ss)) return ss;
        }
        h[j+1]=0.25*h[j];
```

This is a key step: The factor is 0.25 even though the stepsize is decreased by only 0.5. This makes the extrapolation a polynomial in h^2 as allowed by equation (4.2.1), not just a polynomial in h.

```
    }
    nrerror("Too many steps in routine qromb");
    return 0.0;                         Never get here.
}
```

extrapolated value at desired x (h=0)

Numerical recipes (Press et al.)

Excercise

Integrate the following function in the interval [0...π] by using, e.g., the trapezoidal rule.

$$\sin(x) / x^{(3/2)}$$

Note the pole of the function at $x = 0$. (Result: 2.651469)

<u>Steps:</u>

1) Calculate an analytic expression for f(x) if x-->0 and add it to your integration at the end.
2) In order to better sample the strongly changing function values as x-->0 use, e.g., logarithmically increasing stepsizes as you move to larger x (i.e. equidistant steps in log).

Excercise

solution

```
/* Program int_log - Numerical integration in logarithm */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

#define NSTEP 10000
#define PI 3.1415927

#define xa -10.0
#define xe 0.49714987
```

$=\lg(\pi)$

```
double func(double x) {
 x = pow(10.0,x);
 return(sin(x)/x/sqrt(x));
}

int main() {
  int i;
  double xlow,xhigh,area;

  area = 0.0;

  for (i=0;i<NSTEP;i++) {
   xlow = xa+i*(xe-xa)/NSTEP;
   xhigh = xa+(i+1)*(xe-xa)/NSTEP;

   area+= func((xhigh+xlow)/2.0)*(pow(10.0,xhigh)-pow(10.0,xlow));
  }
  area+=sqrt(pow(10.0,xa))*2.0;

  printf("Value of integral: %le\n",area);
}
```

for $x \to 0$

$$\sin(x) * x^{-1.5}$$

$$= x * x^{-1.5} = x^{-0.5}$$

$$\Rightarrow \text{Integral}(x^{-0.5}) = 2x^{0.5}$$