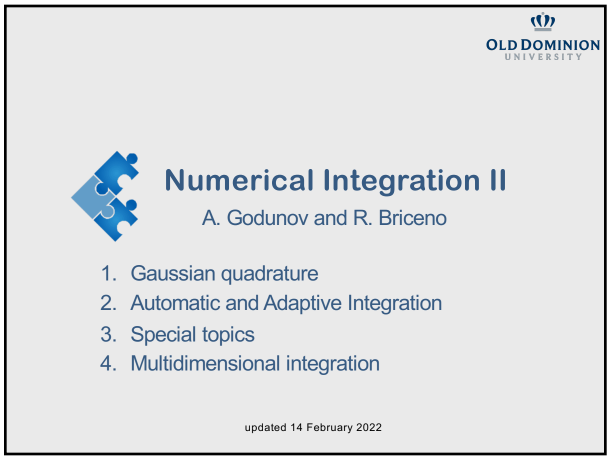


OLD DOMINION UNIVERSITY

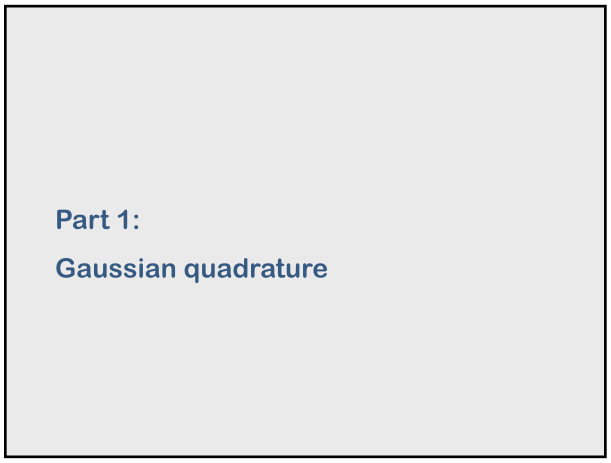
Numerical Integration II

A. Godunov and R. Briceno

1. Gaussian quadrature
2. Automatic and Adaptive Integration
3. Special topics
4. Multidimensional integration

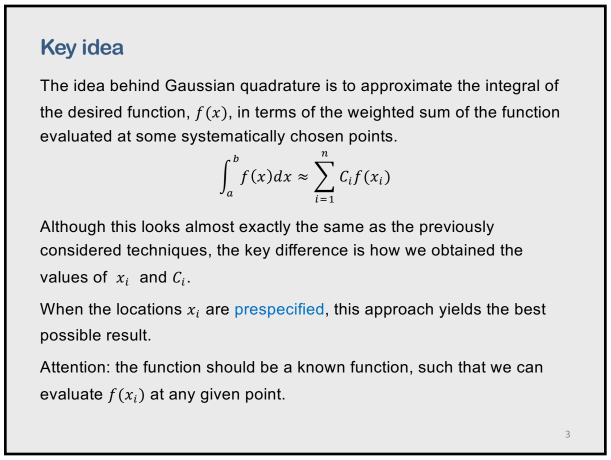
updated 14 February 2022

1



Part 1: Gaussian quadrature

2



Key idea

The idea behind Gaussian quadrature is to approximate the integral of the desired function, $f(x)$, in terms of the weighted sum of the function evaluated at some systematically chosen points.

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

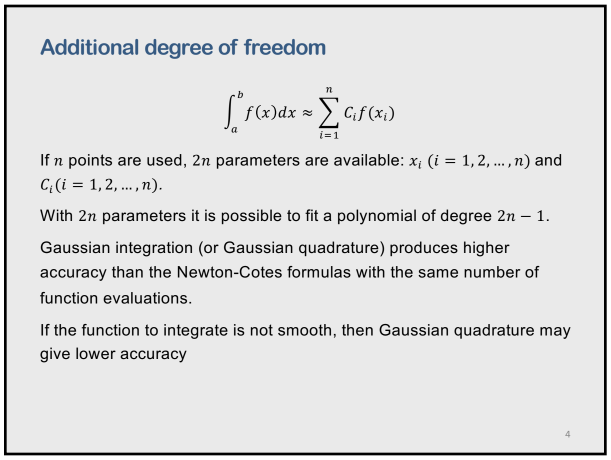
Although this looks almost exactly the same as the previously considered techniques, the key difference is how we obtained the values of x_i and C_i.

When the locations x_i are prespecified, this approach yields the best possible result.

Attention: the function should be a known function, such that we can evaluate $f(x_i)$ at any given point.

3

3



Additional degree of freedom

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

If n points are used, $2n$ parameters are available: x_i $(i = 1, 2, \dots, n)$ and $C_i (i = 1, 2, \dots, n)$.

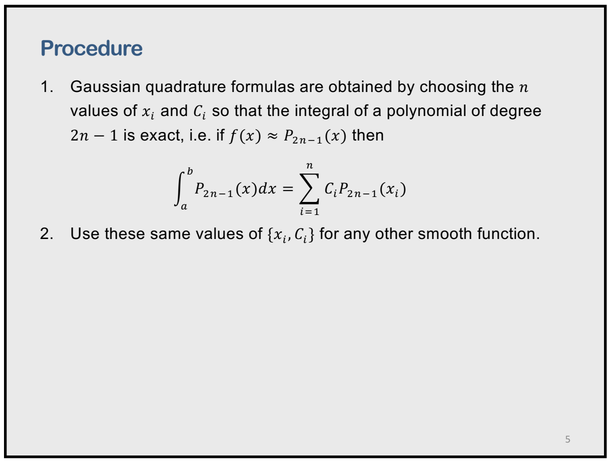
With $2n$ parameters it is possible to fit a polynomial of degree $2n - 1$.

Gaussian integration (or Gaussian quadrature) produces higher accuracy than the Newton-Cotes formulas with the same number of function evaluations.

If the function to integrate is not smooth, then Gaussian quadrature may give lower accuracy

4

4



Procedure

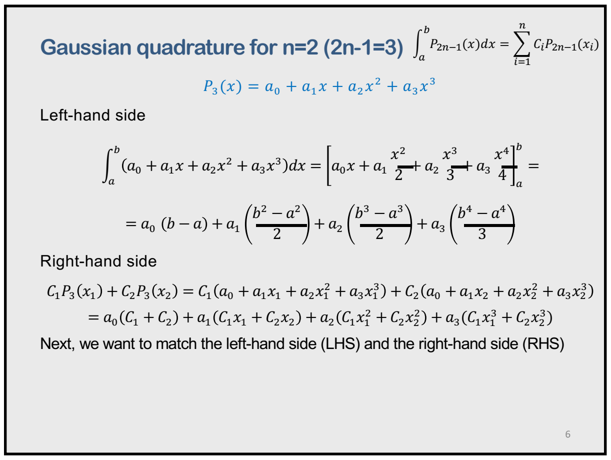
1. Gaussian quadrature formulas are obtained by choosing the n values of x_i and C_i so that the integral of a polynomial of degree $2n - 1$ is exact, i.e. if $f(x) \approx P_{2n-1}(x)$ then

$$\int_a^b P_{2n-1}(x)dx = \sum_{i=1}^{n} C_i P_{2n-1}(x_i)$$

2. Use these same values of $\{x_i, C_i\}$ for any other smooth function.

5

5



Gaussian quadrature for n=2 (2n-1=3) $\int_a^b P_{2n-1}(x)dx = \sum_{i=1}^{n} C_i P_{2n-1}(x_i)$

$$P_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

Left-hand side

$$\int_a^b (a_0 + a_1 x + a_2 x^2 + a_3 x^3)dx = \left[a_0 x + a_1 \frac{x^2}{2} + a_2 \frac{x^3}{3} + a_3 \frac{x^4}{4}\right]_a^b =$$

$$= a_0\ (b - a) + a_1 \left(\frac{b^2 - a^2}{2}\right) + a_2 \left(\frac{b^3 - a^3}{2}\right) + a_3 \left(\frac{b^4 - a^4}{3}\right)$$

Right-hand side

$$C_1 P_3(x_1) + C_2 P_3(x_2) = C_1(a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3) + C_2(a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3)$$
$$= a_0(C_1 + C_2) + a_1(C_1 x_1 + C_2 x_2) + a_2(C_1 x_1^2 + C_2 x_2^2) + a_3(C_1 x_1^3 + C_2 x_2^3)$$

Next, we want to match the left-hand side (LHS) and the right-hand side (RHS)

6

6

Gaussian quadrature for n=2 (cont.)

$$a_0(b-a)+a_1\left(\frac{b^2-a^2}{2}\right)+a_2\left(\frac{b^3-a^3}{2}\right)+a_3\left(\frac{b^4-a^4}{3}\right)=$$

$$=a_0\,(C_1+C_2)+a_1(C_1x_1+C_2x_2)+a_2(C_1x_1^2+C_2x_2^2)+a_3(C_1x_1^3+C_2x_2^3)$$

Four unknowns and four equations

$b-a=C_1+C_2$

$\frac{b^2-a^2}{2}=C_1x_1+C_2x_2$

$\frac{b^3-a^3}{3}=C_1x_1^2+C_2x_2^2$

$\frac{b^4-a^4}{4}=C_1x_1^3+C_2x_2^3$

and the solutions

$C_1=\frac{b-a}{2}$

$C_2=\frac{b-a}{2}$

$x_1=\left(\frac{b-a}{2}\right)\left(-\frac{1}{\sqrt{3}}\right)+\frac{b+a}{2}$

$x_1=\left(\frac{b-a}{2}\right)\left(+\frac{1}{\sqrt{3}}\right)+\frac{b+a}{2}$

7

7

Parameterization

Gaussian quadrature is typically written in terms of a parameter t, defined via

$x=\frac{b-a}{2}t+\frac{b+a}{2}.$

This replaces the integration range from $x\in[a,b]$ to $t\in[-1,+1]$.

If we define $F(t)\equiv f[x(t)]$ and redefine C_i as $C_i\Rightarrow C_i(b-a)/2$ then the Gaussian quadrature can be written as

$$\int_a^b f(x)\,dx=\frac{b-a}{2}\int_{-1}^{1}F(t)dt=\frac{b-a}{2}\sum_{i=1}^{n}C_iF(t_i)$$

There is no inherent advantage of this form, since they are mathematically identical, but it is common in the literature.

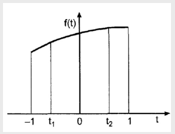
8

8

Re-deriving the previous result

$$\int_{-1}^{1}F(t)dx=\sum_{i=1}^{n}C_iF(t_i)$$

With this, we can re-derive the values of $\{t_1,t_2,C_1C_2\}$ by asserting that the integrals are exact for $F(t)=1+\;t+t^2+t^3$. This gives four different constraints,



$\int_{-1}^{1}1dt=2=C_1+C_2$

$\int_{-1}^{1}tdt=0=C_1t_1+C_2t_2$

$\int_{-1}^{1}t^2dt=\frac{2}{3}=C_1t_1^2+C_2t_2^2$

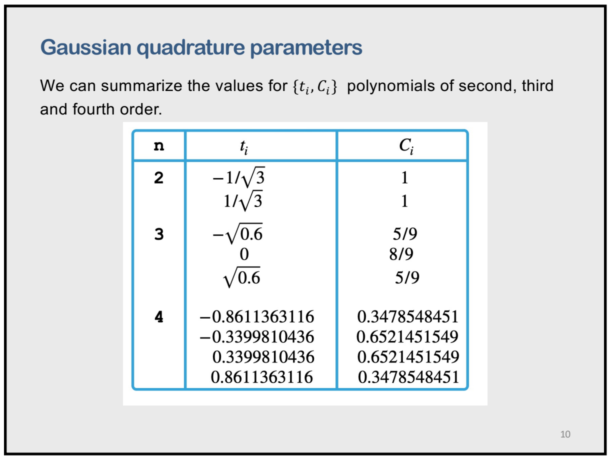
$\int_{-1}^{1}t^3dt=0=C_1t_1^3+C_2t_2^3$

$$\int_{-1}^{1}F(t)dt=F\left(-\frac{1}{\sqrt{3}}\right)+F\left(+\frac{1}{\sqrt{3}}\right)$$

Solving the system … $C_1=1,C_2=1,t_1=-t_2=1/\sqrt{3}$

9

9



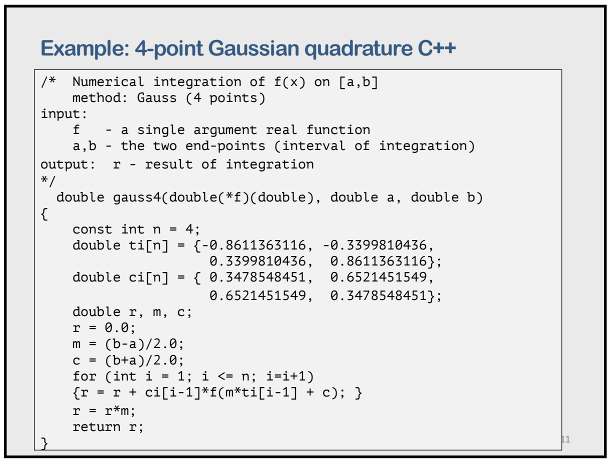
Gaussian quadrature parameters

We can summarize the values for $\{t_i, C_i\}$ polynomials of second, third and fourth order.

n	t_i	C_i
2	$-1/\sqrt{3}$	1
	$1/\sqrt{3}$	1
3	$-\sqrt{0.6}$	5/9
	0	8/9
	$\sqrt{0.6}$	5/9
4	−0.8611363116	0.3478548451
	−0.3399810436	0.6521451549
	0.3399810436	0.6521451549
	0.8611363116	0.3478548451

10

10

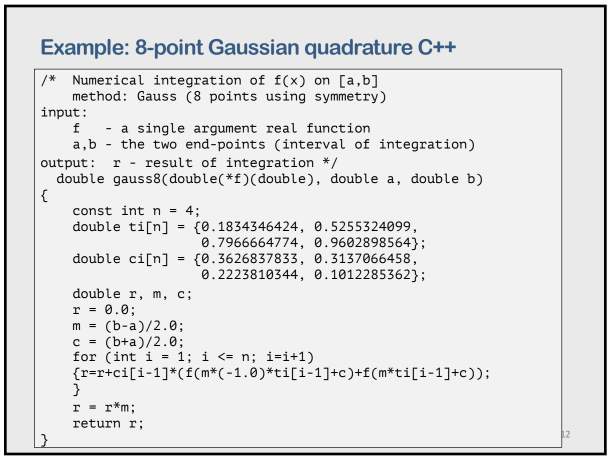


Example: 4-point Gaussian quadrature C++

```
/*  Numerical integration of f(x) on [a,b]
    method: Gauss (4 points)
input:
    f   - a single argument real function
    a,b - the two end-points (interval of integration)
output:  r - result of integration
*/
  double gauss4(double(*f)(double), double a, double b)
{
    const int n = 4;
    double ti[n] = {-0.8611363116, -0.3399810436,
                     0.3399810436,  0.8611363116};
    double ci[n] = { 0.3478548451,  0.6521451549,
                     0.6521451549,  0.3478548451};
    double r, m, c;
    r = 0.0;
    m = (b-a)/2.0;
    c = (b+a)/2.0;
    for (int i = 1; i <= n; i=i+1)
    {r = r + ci[i-1]*f(m*ti[i-1] + c); }
    r = r*m;
    return r;
}
```

11

11

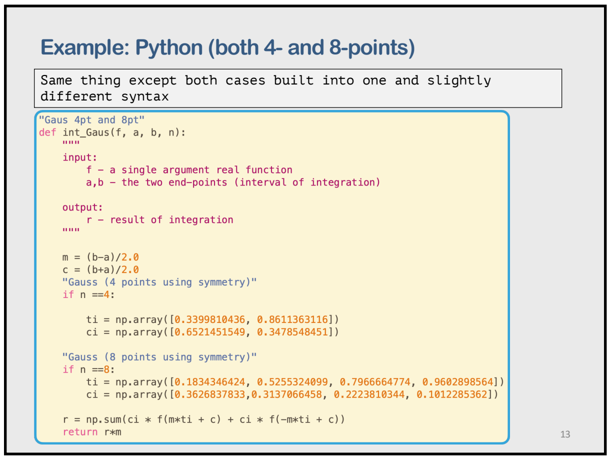


Example: 8-point Gaussian quadrature C++

```
/*  Numerical integration of f(x) on [a,b]
    method: Gauss (8 points using symmetry)
input:
    f   - a single argument real function
    a,b - the two end-points (interval of integration)
output:  r - result of integration */
  double gauss8(double(*f)(double), double a, double b)
{
    const int n = 4;
    double ti[n] = {0.1834346424, 0.5255324099,
                    0.7966664774, 0.9602898564};
    double ci[n] = {0.3626837833, 0.3137066458,
                    0.2223810344, 0.1012285362};
    double r, m, c;
    r = 0.0;
    m = (b-a)/2.0;
    c = (b+a)/2.0;
    for (int i = 1; i <= n; i=i+1)
    {r=r+ci[i-1]*(f(m*(-1.0)*ti[i-1]+c)+f(m*ti[i-1]+c));
    }
    r = r*m;
    return r;
}
```

12

12



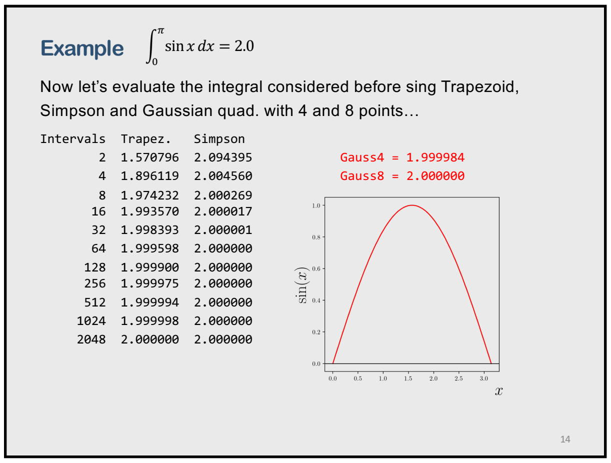
Example: Python (both 4- and 8-points)

```
Same thing except both cases built into one and slightly
different syntax
```

```
"Gaus 4pt and 8pt"
def int_Gaus(f, a, b, n):
    """
    input:
        f - a single argument real function
        a,b - the two end-points (interval of integration)

    output:
        r - result of integration
    """

    m = (b-a)/2.0
    c = (b+a)/2.0
    "Gauss (4 points using symmetry)"
    if n ==4:

        ti = np.array([0.3399810436, 0.8611363116])
        ci = np.array([0.6521451549, 0.3478548451])

    "Gauss (8 points using symmetry)"
    if n ==8:
        ti = np.array([0.1834346424, 0.5255324099, 0.7966664774, 0.9602898564])
        ci = np.array([0.3626837833,0.3137066458, 0.2223810344, 0.1012285362])

    r = np.sum(ci * f(m*ti + c) + ci * f(-m*ti + c))
    return r*m
```

13

13

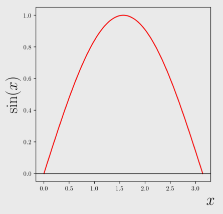


Example $\int_0^{\pi} \sin x \, dx = 2.0$

Now let's evaluate the integral considered before sing Trapezoid, Simpson and Gaussian quad. with 4 and 8 points…

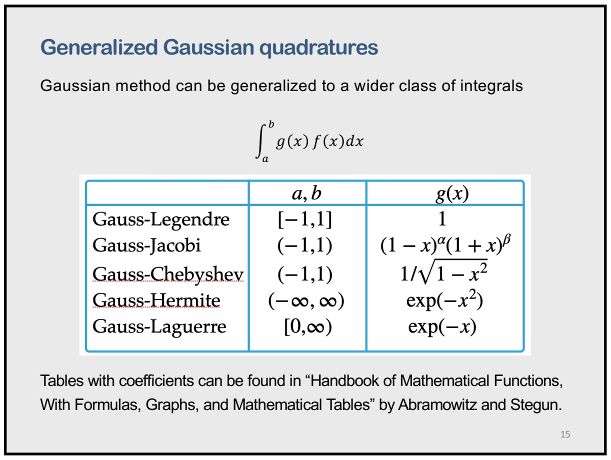
```
Intervals  Trapez.   Simpson
        2  1.570796  2.094395
        4  1.896119  2.004560
        8  1.974232  2.000269
       16  1.993570  2.000017
       32  1.998393  2.000001
       64  1.999598  2.000000
      128  1.999900  2.000000
      256  1.999975  2.000000
      512  1.999994  2.000000
     1024  1.999998  2.000000
     2048  2.000000  2.000000
```

```
Gauss4 = 1.999984
Gauss8 = 2.000000
```



14

14



Generalized Gaussian quadratures

Gaussian method can be generalized to a wider class of integrals

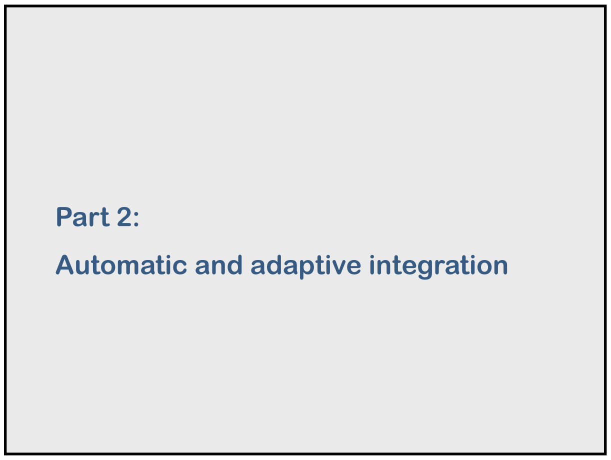
$$\int_a^b g(x)\, f(x)\, dx$$

	a, b	$g(x)$
Gauss-Legendre	$[-1,1]$	1
Gauss-Jacobi	$(-1,1)$	$(1-x)^{\alpha}(1+x)^{\beta}$
Gauss-Chebyshev	$(-1,1)$	$1/\sqrt{1-x^2}$
Gauss-Hermite	$(-\infty,\infty)$	$\exp(-x^2)$
Gauss-Laguerre	$[0,\infty)$	$\exp(-x)$

Tables with coefficients can be found in "Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables" by Abramowitz and Stegun.

15

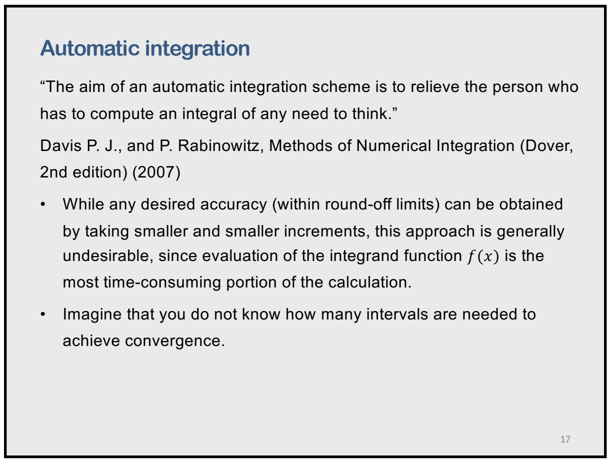
15



Part 2:

Automatic and adaptive integration

16



Automatic integration

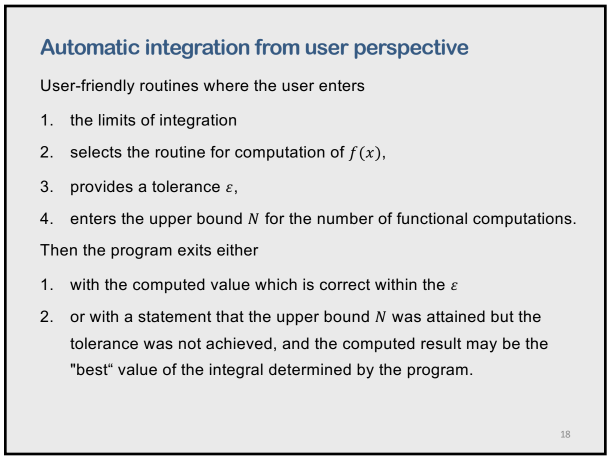
"The aim of an automatic integration scheme is to relieve the person who has to compute an integral of any need to think."

Davis P. J., and P. Rabinowitz, Methods of Numerical Integration (Dover, 2nd edition) (2007)

- While any desired accuracy (within round-off limits) can be obtained by taking smaller and smaller increments, this approach is generally undesirable, since evaluation of the integrand function $f(x)$ is the most time-consuming portion of the calculation.
- Imagine that you do not know how many intervals are needed to achieve convergence.

17

17



Automatic integration from user perspective

User-friendly routines where the user enters

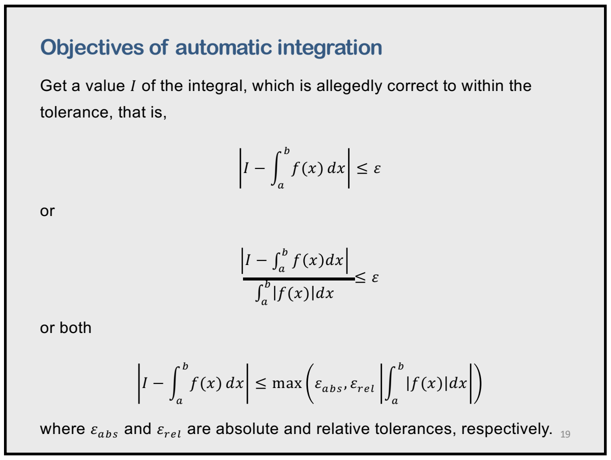
1. the limits of integration
2. selects the routine for computation of $f(x)$,
3. provides a tolerance ε,
4. enters the upper bound N for the number of functional computations.

Then the program exits either

1. with the computed value which is correct within the ε
2. or with a statement that the upper bound N was attained but the tolerance was not achieved, and the computed result may be the "best" value of the integral determined by the program.

18

18



Objectives of automatic integration

Get a value I of the integral, which is allegedly correct to within the tolerance, that is,

$$\left| I - \int_a^b f(x)\,dx \right| \le \varepsilon$$

or

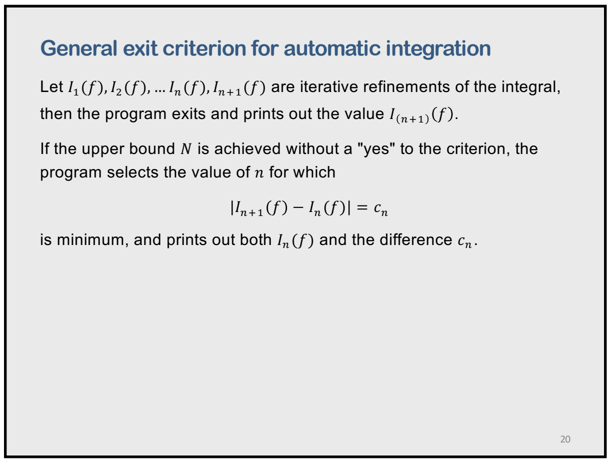
$$\frac{\left| I - \int_a^b f(x)dx \right|}{\int_a^b |f(x)|dx} \le \varepsilon$$

or both

$$\left| I - \int_a^b f(x)\,dx \right| \le \max\left(\varepsilon_{abs}, \varepsilon_{rel} \left| \int_a^b |f(x)|dx \right| \right)$$

where ε_{abs} and ε_{rel} are absolute and relative tolerances, respectively. 19

19



General exit criterion for automatic integration

Let $I_1(f), I_2(f), \dots I_n(f), I_{n+1}(f)$ are iterative refinements of the integral, then the program exits and prints out the value $I_{(n+1)}(f)$.

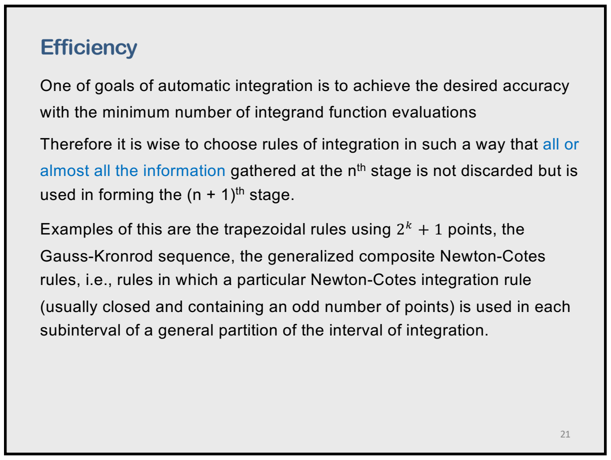
If the upper bound N is achieved without a "yes" to the criterion, the program selects the value of n for which

$$|I_{n+1}(f) - I_n(f)| = c_n$$

is minimum, and prints out both $I_n(f)$ and the difference c_n.

20

20



Efficiency

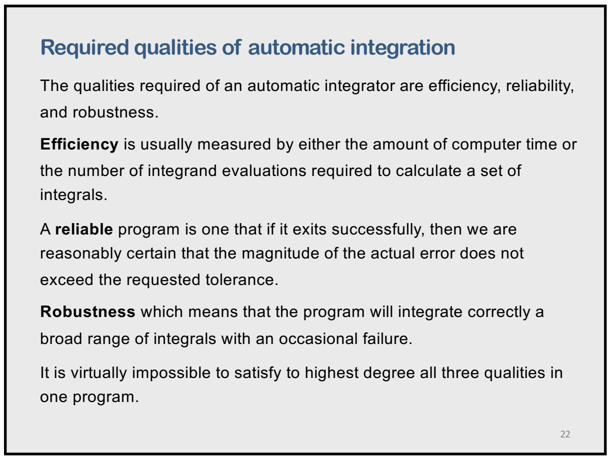
One of goals of automatic integration is to achieve the desired accuracy with the minimum number of integrand function evaluations

Therefore it is wise to choose rules of integration in such a way that all or almost all the information gathered at the nth stage is not discarded but is used in forming the (n + 1)th stage.

Examples of this are the trapezoidal rules using $2^k + 1$ points, the Gauss-Kronrod sequence, the generalized composite Newton-Cotes rules, i.e., rules in which a particular Newton-Cotes integration rule (usually closed and containing an odd number of points) is used in each subinterval of a general partition of the interval of integration.

21

21



Required qualities of automatic integration

The qualities required of an automatic integrator are efficiency, reliability, and robustness.

Efficiency is usually measured by either the amount of computer time or the number of integrand evaluations required to calculate a set of integrals.

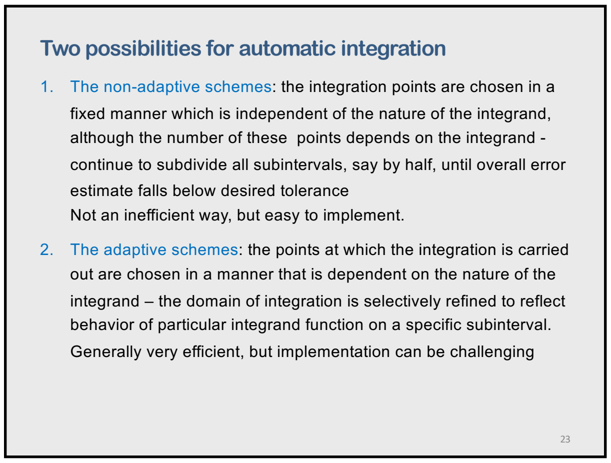
A **reliable** program is one that if it exits successfully, then we are reasonably certain that the magnitude of the actual error does not exceed the requested tolerance.

Robustness which means that the program will integrate correctly a broad range of integrals with an occasional failure.

It is virtually impossible to satisfy to highest degree all three qualities in one program.

22

22

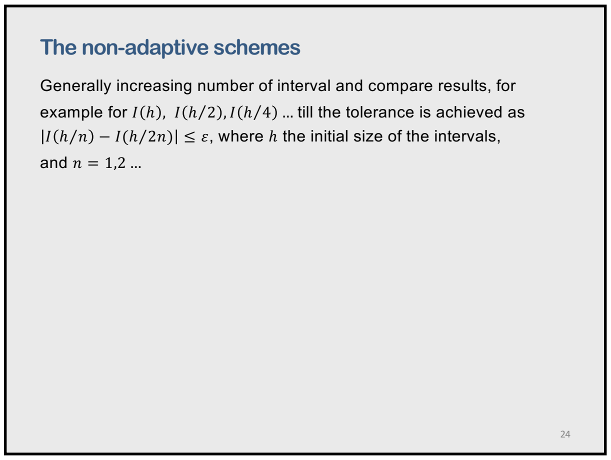


Two possibilities for automatic integration

1. The non-adaptive schemes: the integration points are chosen in a fixed manner which is independent of the nature of the integrand, although the number of these points depends on the integrand - continue to subdivide all subintervals, say by half, until overall error estimate falls below desired tolerance
   Not an inefficient way, but easy to implement.
2. The adaptive schemes: the points at which the integration is carried out are chosen in a manner that is dependent on the nature of the integrand – the domain of integration is selectively refined to reflect behavior of particular integrand function on a specific subinterval.
   Generally very efficient, but implementation can be challenging

23

23



The non-adaptive schemes

Generally increasing number of interval and compare results, for example for $I(h),\ I(h/2), I(h/4)$ … till the tolerance is achieved as $|I(h/n) - I(h/2n)| \leq \varepsilon$, where h the initial size of the intervals, and $n = 1,2$ …

24

24

Example: Non-adaptive Simpson rule

```
Subroutine simpson2(f,a,b,eps,integral,nint)
!==========================================================
! Integration of f(x) on [a,b]
! Method: Simpson rule with doubling number of intervals
!         till  error = coeff*|I_n - I_2n| < eps
… here we declare data types
nmax=16384
coeff = 1.0/15.0
h = (b-a)/2.0
sn = (1.0/3.0)*h*(f(a)+4.0*f(a+h)+f(b))
! loop over number of intervals (starting from 4 intervals)
n=4
do while (n <= nmax)
   s2n = 0.0
   h = (b-a)/dfloat(n)
   do i=2, n-2, 2
      x   = a+dfloat(i)*h
      s2n = s2n + 2.0*f(x) + 4.0*f(x+h)
   end do
   s2n = (s2n + f(a) + f(b) + 4.0*f(a+h))*h/3.0
   if(coeff*abs(s2n-sn) <= eps) then
      integral = s2n + coeff*(s2n-sn)
      nint = n
      exit
   end if
   sn = s2n
   n = n*2
end do
return
end subroutine simpson2
```

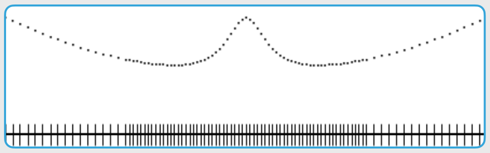
25

25

The adaptive schemes

Adaptive integration is a generic name denoting a strategy to achieve the desired accuracy with the minimum number of integrand function evaluations.

The overall range of integration is broken into several subranges, and each subrange is evaluated to the desired accuracy by subdividing each individual subrange as required until the desired accuracy is obtained.

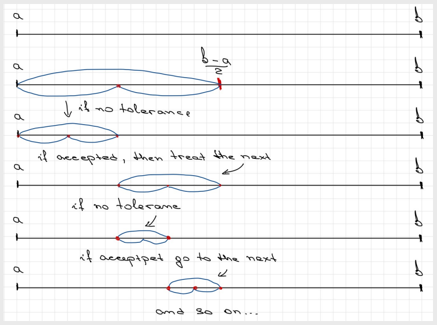


26

26

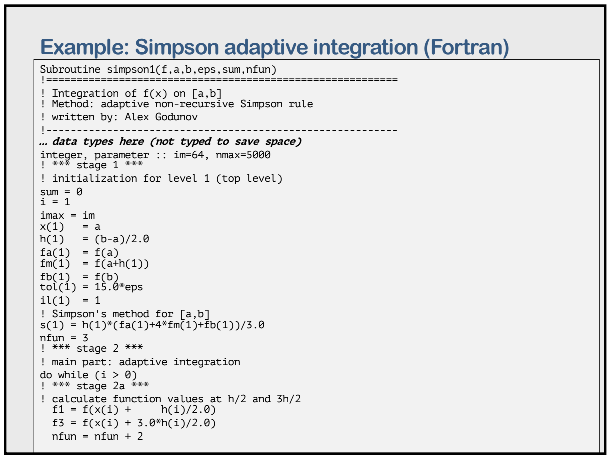
Strategies for adaptive algorithms

There are very many approaches to adaptive integration. Here is an example using doubling of subintervals



27

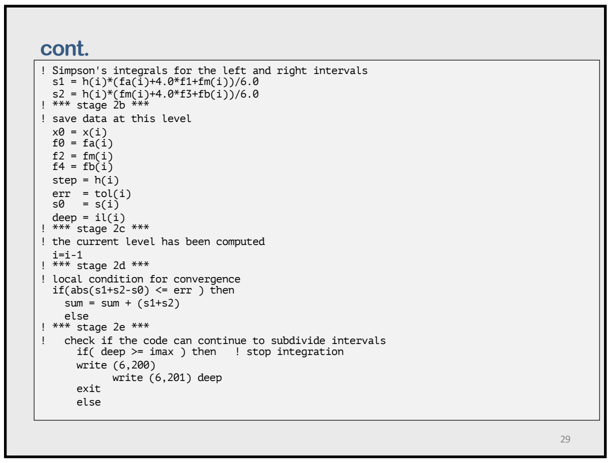
27



Example: Simpson adaptive integration (Fortran)

```fortran
Subroutine simpson1(f,a,b,eps,sum,nfun)
!============================================================
! Integration of f(x) on [a,b]
! Method: adaptive non-recursive Simpson rule
! written by: Alex Godunov
!------------------------------------------------------------
```
… data types here (not typed to save space)
```fortran
integer, parameter :: im=64, nmax=5000
! *** stage 1 ***
! initialization for level 1 (top level)
sum = 0
i = 1
imax = im
x(1)   = a
h(1)   = (b-a)/2.0
fa(1)  = f(a)
fm(1)  = f(a+h(1))
fb(1)  = f(b)
tol(1) = 15.0*eps
il(1)  = 1
! Simpson's method for [a,b]
s(1) = h(1)*(fa(1)+4*fm(1)+fb(1))/3.0
nfun = 3
! *** stage 2 ***
! main part: adaptive integration
do while (i > 0)
! *** stage 2a ***
! calculate function values at h/2 and 3h/2
  f1 = f(x(i) +     h(i)/2.0)
  f3 = f(x(i) + 3.0*h(i)/2.0)
  nfun = nfun + 2
```

28

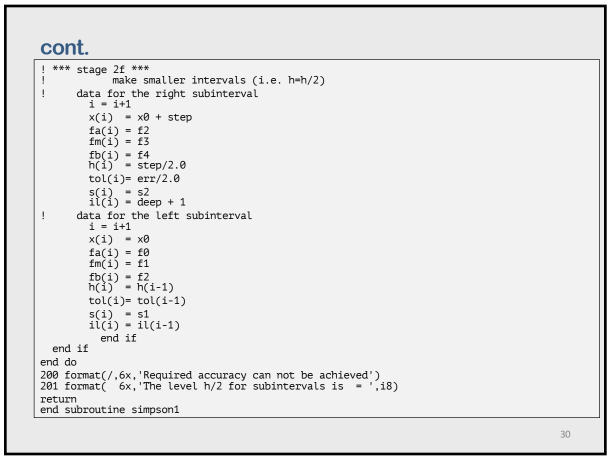


cont.

```fortran
! Simpson's integrals for the left and right intervals
  s1 = h(i)*(fa(i)+4.0*f1+fm(i))/6.0
  s2 = h(i)*(fm(i)+4.0*f3+fb(i))/6.0
! *** stage 2b ***
! save data at this level
  x0 = x(i)
  f0 = fa(i)
  f2 = fm(i)
  f4 = fb(i)
  step = h(i)
  err  = tol(i)
  s0   = s(i)
  deep = il(i)
! *** stage 2c ***
! the current level has been computed
  i=i-1
! *** stage 2d ***
! local condition for convergence
  if(abs(s1+s2-s0) <= err ) then
    sum = sum + (s1+s2)
    else
! *** stage 2e ***
!   check if the code can continue to subdivide intervals
      if( deep >= imax ) then   ! stop integration
      write (6,200)
            write (6,201) deep
      exit
      else
```

29

29

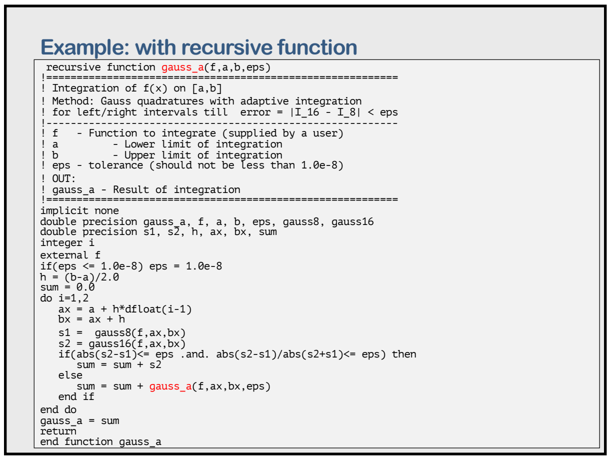


cont.

```fortran
! *** stage 2f ***
!           make smaller intervals (i.e. h=h/2)
!     data for the right subinterval
        i = i+1
        x(i)  = x0 + step
        fa(i) = f2
        fm(i) = f3
        fb(i) = f4
        h(i)  = step/2.0
        tol(i)= err/2.0
        s(i)  = s2
        il(i) = deep + 1
!     data for the left subinterval
        i = i+1
        x(i)  = x0
        fa(i) = f0
        fm(i) = f1
        fb(i) = f2
        h(i)  = h(i-1)
        tol(i)= tol(i-1)
        s(i)  = s1
        il(i) = il(i-1)
          end if
  end if
end do
200 format(/,6x,'Required accuracy can not be achieved')
201 format(  6x,'The level h/2 for subintervals is  = ',i8)
return
end subroutine simpson1
```

30

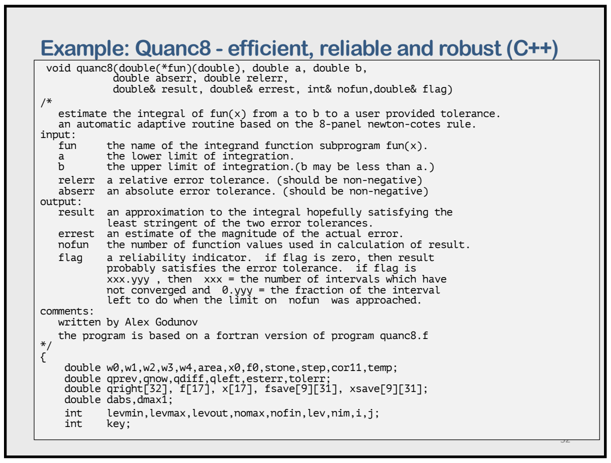
30



Example: with recursive function

```
 recursive function gauss_a(f,a,b,eps)
!==========================================================
! Integration of f(x) on [a,b]
! Method: Gauss quadratures with adaptive integration
! for left/right intervals till  error = |I_16 - I_8| < eps
!----------------------------------------------------------
! f   - Function to integrate (supplied by a user)
! a         - Lower limit of integration
! b         - Upper limit of integration
! eps - tolerance (should not be less than 1.0e-8)
! OUT:
! gauss_a - Result of integration
!==========================================================
implicit none
double precision gauss_a, f, a, b, eps, gauss8, gauss16
double precision s1, s2, h, ax, bx, sum
integer i
external f
if(eps <= 1.0e-8) eps = 1.0e-8
h = (b-a)/2.0
sum = 0.0
do i=1,2
   ax = a + h*dfloat(i-1)
   bx = ax + h
   s1 =  gauss8(f,ax,bx)
   s2 = gauss16(f,ax,bx)
   if(abs(s2-s1)<= eps .and. abs(s2-s1)/abs(s2+s1)<= eps) then
      sum = sum + s2
   else
      sum = sum + gauss_a(f,ax,bx,eps)
   end if
end do
gauss_a = sum
return
end function gauss_a
```

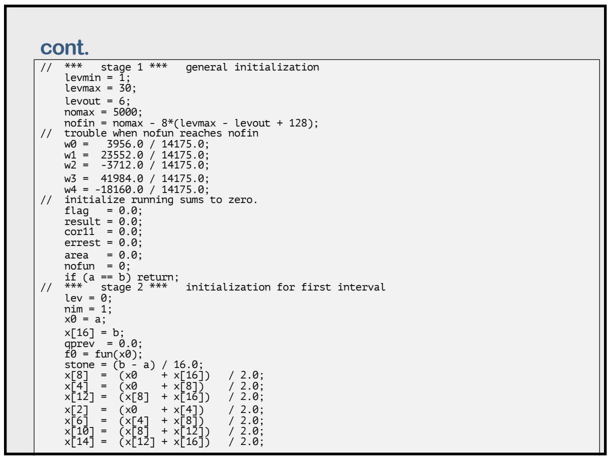
31



Example: Quanc8 - efficient, reliable and robust (C++)

```
 void quanc8(double(*fun)(double), double a, double b,
           double abserr, double relerr,
           double& result, double& errest, int& nofun,double& flag)
/*
   estimate the integral of fun(x) from a to b to a user provided tolerance.
   an automatic adaptive routine based on the 8-panel newton-cotes rule.
input:
   fun     the name of the integrand function subprogram fun(x).
   a       the lower limit of integration.
   b       the upper limit of integration.(b may be less than a.)
   relerr  a relative error tolerance. (should be non-negative)
   abserr  an absolute error tolerance. (should be non-negative)
output:
   result  an approximation to the integral hopefully satisfying the
           least stringent of the two error tolerances.
   errest  an estimate of the magnitude of the actual error.
   nofun   the number of function values used in calculation of result.
   flag    a reliability indicator.  if flag is zero, then result
           probably satisfies the error tolerance.  if flag is
           xxx.yyy , then  xxx = the number of intervals which have
           not converged and  0.yyy = the fraction of the interval
           left to do when the limit on  nofun  was approached.
comments:
   written by Alex Godunov
   the program is based on a fortran version of program quanc8.f
*/
{
    double w0,w1,w2,w3,w4,area,x0,f0,stone,step,cor11,temp;
    double qprev,qnow,qdiff,qleft,esterr,tolerr;
    double qright[32], f[17], x[17], fsave[9][31], xsave[9][31];
    double dabs,dmax1;
    int    levmin,levmax,levout,nomax,nofin,lev,nim,i,j;
    int    key;
```

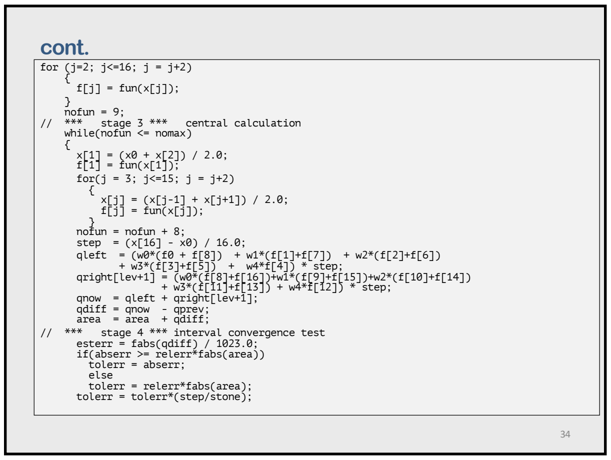
32



cont.

```
//  ***   stage 1 ***   general initialization
    levmin = 1;
    levmax = 30;
    levout = 6;
    nomax = 5000;
    nofin = nomax - 8*(levmax - levout + 128);
//  trouble when nofun reaches nofin
    w0 =   3956.0 / 14175.0;
    w1 =  23552.0 / 14175.0;
    w2 =  -3712.0 / 14175.0;
    w3 =  41984.0 / 14175.0;
    w4 = -18160.0 / 14175.0;
//  initialize running sums to zero.
    flag   = 0.0;
    result = 0.0;
    cor11  = 0.0;
    errest = 0.0;
    area   = 0.0;
    nofun  = 0;
    if (a == b) return;
//  ***   stage 2 ***   initialization for first interval
    lev = 0;
    nim = 1;
    x0 = a;
    x[16] = b;
    qprev  = 0.0;
    f0 = fun(x0);
    stone = (b - a) / 16.0;
    x[8]  =  (x0    + x[16])   / 2.0;
    x[4]  =  (x0    + x[8])    / 2.0;
    x[12] =  (x[8]  + x[16])   / 2.0;
    x[2]  =  (x0    + x[4])    / 2.0;
    x[6]  =  (x[4]  + x[8])    / 2.0;
    x[10] =  (x[8]  + x[12])   / 2.0;
    x[14] =  (x[12] + x[16])   / 2.0;
```

33

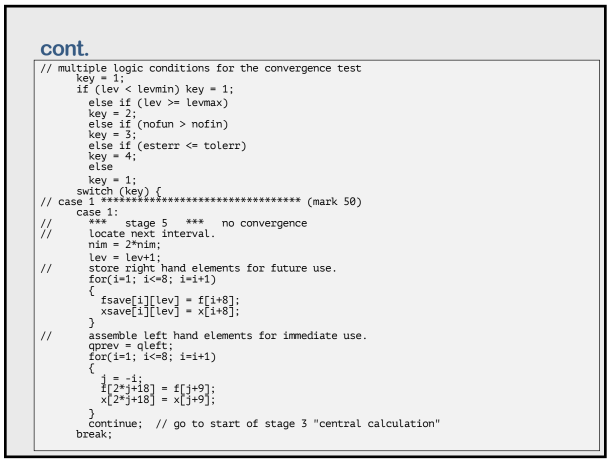


cont.

```
for (j=2; j<=16; j = j+2)
    {
      f[j] = fun(x[j]);
    }
    nofun = 9;
//  ***   stage 3 ***   central calculation
    while(nofun <= nomax)
    {
      x[1] = (x0 + x[2]) / 2.0;
      f[1] = fun(x[1]);
      for(j = 3; j<=15; j = j+2)
        {
          x[j] = (x[j-1] + x[j+1]) / 2.0;
          f[j] = fun(x[j]);
        }
      nofun = nofun + 8;
      step  = (x[16] - x0) / 16.0;
      qleft  = (w0*(f0 + f[8])  + w1*(f[1]+f[7])  + w2*(f[2]+f[6])
             + w3*(f[3]+f[5])  +  w4*f[4]) * step;
      qright[lev+1] = (w0*(f[8]+f[16])+w1*(f[9]+f[15])+w2*(f[10]+f[14])
                    + w3*(f[11]+f[13]) + w4*f[12]) * step;
      qnow  = qleft + qright[lev+1];
      qdiff = qnow  - qprev;
      area  = area  + qdiff;
//  ***   stage 4 *** interval convergence test
      esterr = fabs(qdiff) / 1023.0;
      if(abserr >= relerr*fabs(area))
        tolerr = abserr;
        else
        tolerr = relerr*fabs(area);
      tolerr = tolerr*(step/stone);
```

34

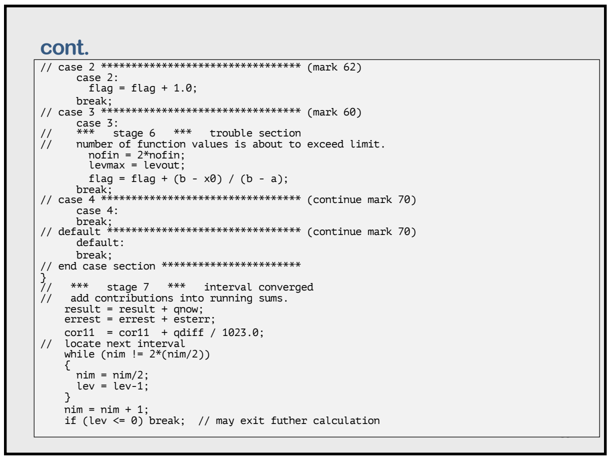
34



cont.

```
// multiple logic conditions for the convergence test
      key = 1;
      if (lev < levmin) key = 1;
        else if (lev >= levmax)
        key = 2;
        else if (nofun > nofin)
        key = 3;
        else if (esterr <= tolerr)
        key = 4;
        else
        key = 1;
      switch (key) {
// case 1 ********************************* (mark 50)
      case 1:
//      ***   stage 5   ***   no convergence
//      locate next interval.
        nim = 2*nim;
        lev = lev+1;
//      store right hand elements for future use.
        for(i=1; i<=8; i=i+1)
        {
          fsave[i][lev] = f[i+8];
          xsave[i][lev] = x[i+8];
        }
//      assemble left hand elements for immediate use.
        qprev = qleft;
        for(i=1; i<=8; i=i+1)
        {
          j = -i;
          f[2*j+18] = f[j+9];
          x[2*j+18] = x[j+9];
        }
        continue;  // go to start of stage 3 "central calculation"
      break;
```

35



cont.

```
// case 2 ********************************* (mark 62)
      case 2:
        flag = flag + 1.0;
      break;
// case 3 ********************************* (mark 60)
      case 3:
//    ***   stage 6   ***   trouble section
//    number of function values is about to exceed limit.
        nofin = 2*nofin;
        levmax = levout;
        flag = flag + (b - x0) / (b - a);
      break;
// case 4 ********************************* (continue mark 70)
      case 4:
      break;
// default ******************************** (continue mark 70)
      default:
      break;
// end case section ***********************
}
//   ***   stage 7   ***   interval converged
//   add contributions into running sums.
    result = result + qnow;
    errest = errest + esterr;
    cor11  = cor11  + qdiff / 1023.0;
//  locate next interval
    while (nim != 2*(nim/2))
    {
      nim = nim/2;
      lev = lev-1;
    }
    nim = nim + 1;
    if (lev <= 0) break;  // may exit futher calculation
```

36

cont.

```
//  assemble elements required for the next interval.
    qprev = qright[lev];
    x0 = x[16];
    f0 = f[16];
    for (i =1; i<=8; i=i+1)
    {
      f[2*i] = fsave[i][lev];
      x[2*i] = xsave[i][lev];
    }
}
//  *** end stage 3 ***   central calculation
//  ***   stage 8   ***   finalize and return
    result = result + cor11;
//  make sure errest not less than roundoff level.
    if (errest == 0.0) return;
    do
    {
     temp = fabs(result) + errest;
     errest = 2.0*errest;
    }
    while (temp == fabs(result));
    return;
}
```

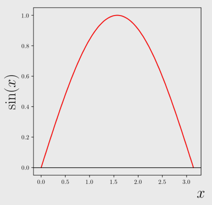
37

37

Example: eps=10^{-8}

$$\int_0^\pi \sin x \, dx$$

```
Adaptive Simpson =   2.00000001
Tol. estimated   =   0.00000001
Function calls   =          121

Integral Quanc8  =   2.00000000
Tol. estimated   =   0.00000000
Function calls   =           33

Adaptive G7K15   =   2.00000000
Tol. estimated   =   0.00000000
Function calls   =           15

Adaptive G10K21  =   2.00000000
Tol. estimated   =   0.00000000
Function calls   =           21
```



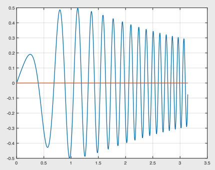
38

38

Example: eps=10^{-8}

$$\int_0^\pi \frac{x\cos(10x^2)}{x^2+1} dx$$

```
Adaptive Simpson =   0.00031559
Tol. estimated   =   0.00000000
Function calls   =         3605

Integral Quanc8  =   0.00031559
Tol. estimated   =   0.00000000
Function calls   =          705

Adaptive G7K15   =   0.00031559
Tol. estimated   =   0.00000000
Function calls   =          615

Adaptive G10K21  =   0.00031559
Tol. estimated   =   0.00000001
Function calls   =          399
```



39

39

What to do if …?

Suppose that an automatic integration scheme is applied to a function $f(x)$ over an interval $[a, b]$ with a given error tolerance ε and exits with an indication of failure. What is one to do next?

1. Increase, if possible, the number of functional evaluations allowed.
2. Raise the error tolerance ε.
3. Subdivide the interval $[a, b]$ into two or more subintervals, preferably in a random manner, and apply the integration scheme separately on each subinterval.
4. Try a different automatic integration scheme, perhaps one with special features suitable for the integrand in question.
5. Try to locate the interior singularities of the integrand and integrate between them, thus converting interior singularities to endpoint singularities which are much more tractable.

40

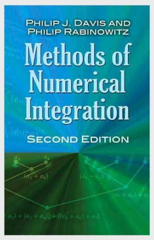
40

Attention

Adaptive programs tend to be effective in practice … but it can be fooled.
Interval of integration may be very wide but "interesting" behavior of integrand is confined to narrow range

Sampling by automatic routine may miss interesting part of integrand behavior, and resulting value for integral may be completely wrong.

Such examples can be found in the excellent book by Davis and Rabinowitz, chapter 6.


Philip J. Davis and Philip Rabinowitz
Methods of Numerical Integration
Second Edition

41

41

Libraries

All respected numerical libraries have routines for adaptive integration.

Adaptive programs can be found also at
https://ww2.odu.edu/~agodunov/computing.html

Fortran:
adaptive integration based on Simpson rule (simpson2.f90),
based on Gauss quadratures (gauss2.f90), adaptive integration using recursive calls (gaussA.f90), adaptive integration based on Newton-Cotes quadrature (quanc8.f)

C++
adaptive integration based on Newton-Cotes quadrature (quanc8.cpp)

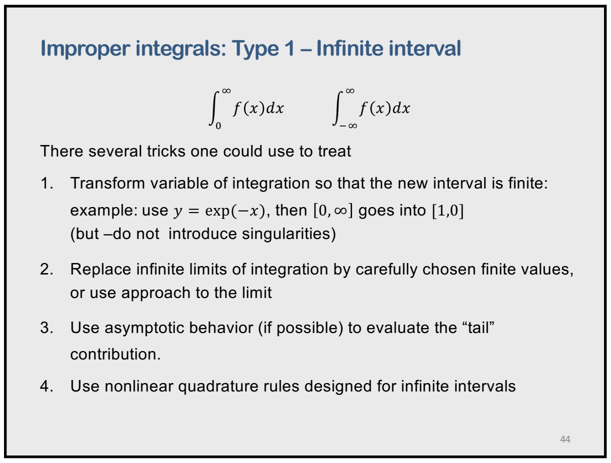
42

42

Part 3:

Special topics

43



Improper integrals: Type 1 – Infinite interval

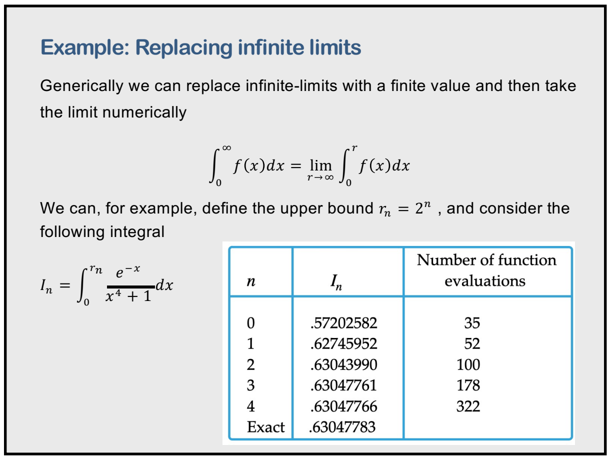
$$\int_0^\infty f(x)dx \qquad \int_{-\infty}^\infty f(x)dx$$

There several tricks one could use to treat

1. Transform variable of integration so that the new interval is finite: example: use $y = \exp(-x)$, then $[0, \infty]$ goes into $[1,0]$ (but –do not introduce singularities)
2. Replace infinite limits of integration by carefully chosen finite values, or use approach to the limit
3. Use asymptotic behavior (if possible) to evaluate the "tail" contribution.
4. Use nonlinear quadrature rules designed for infinite intervals

44

44



Example: Replacing infinite limits

Generically we can replace infinite-limits with a finite value and then take the limit numerically

$$\int_0^\infty f(x)dx = \lim_{r\to\infty}\int_0^r f(x)dx$$

We can, for example, define the upper bound $r_n = 2^n$, and consider the following integral

$$I_n = \int_0^{r_n} \frac{e^{-x}}{x^4+1}dx$$

n	I_n	Number of function evaluations
0	.57202582	35
1	.62745952	52
2	.63043990	100
3	.63047761	178
4	.63047766	322
Exact	.63047783	

45

Example: Using asymptotic behavior

$$\int_0^\infty \frac{\sqrt{x}}{x^2+1}dx = \int_0^a \frac{\sqrt{x}}{x^2+1}dx + \int_a^\infty \frac{\sqrt{x}}{x^2+1}dx$$

for$a \gg 1$ we use the asymptotic behavior of the function

$$\int_a^\infty \frac{\sqrt{x}}{x^2+1}dx \approx \int_a^\infty \frac{\sqrt{x}}{x^2}dx = \int_a^\infty \frac{1}{x^{\frac{3}{2}}}dx = \frac{2}{\sqrt{a}}$$

Then

$$\int_0^\infty \frac{\sqrt{x}}{x^2+1}dx \approx \int_0^a \frac{\sqrt{x}}{x^2+1}dx + \frac{2}{\sqrt{a}}$$

46

46

Improper integrals: Type 2 – discontinuous integrands

Example

$$\int_0^1 f(x)dx$$

when $f(x)$ is discontinuous at 0.

Formal definition

$$\int_0^1 f(x)dx = \lim_{t\to 0}\int_t^1 f(x)dx$$

Proceeding to the limit

$$\int_0^1 f(x)dx\, R = \int_{R_1}^1 f(x)dx + \int_{R_2}^{R_1} f(x)dx + \cdots \qquad R_n = 2^{-n}$$

Other methods: change variables, elimination of singularity, Gauss type quadratures, …

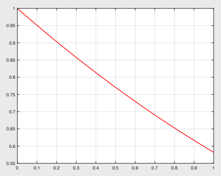
47

47

Example:

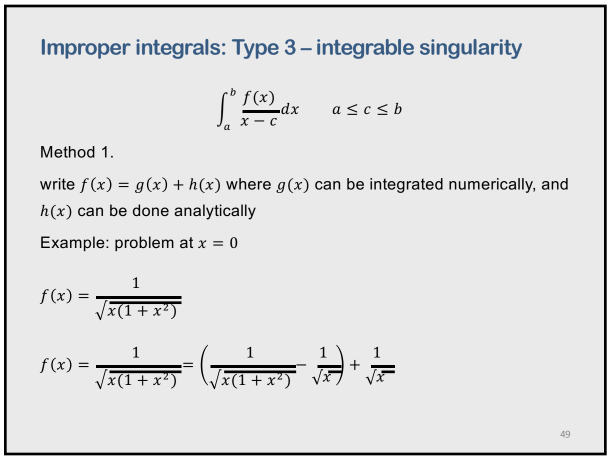
$$\int_0^1 \frac{x}{e^x-1}dx = 0.77750463$$

```
Simpson          =          -NaN

Integral Quanc8  =          -NaN
Tolerance        =    0.00000001
Tol. estimated   =           NaN
Function calls   =          4113

Adaptive G7K15   =    0.77750463
Tolerance        =    0.00000001
Tol. estimated   =    0.00000000
Function calls   =            15

Adaptive G10K21  =    0.77750463
Tolerance        =    0.00000001
Tol. estimated   =    0.00000000
Function calls   =            21
```



48

48



Improper integrals: Type 3 – integrable singularity

$$\int_a^b \frac{f(x)}{x-c}dx \qquad a \le c \le b$$

Method 1.

write $f(x) = g(x) + h(x)$ where $g(x)$ can be integrated numerically, and $h(x)$ can be done analytically

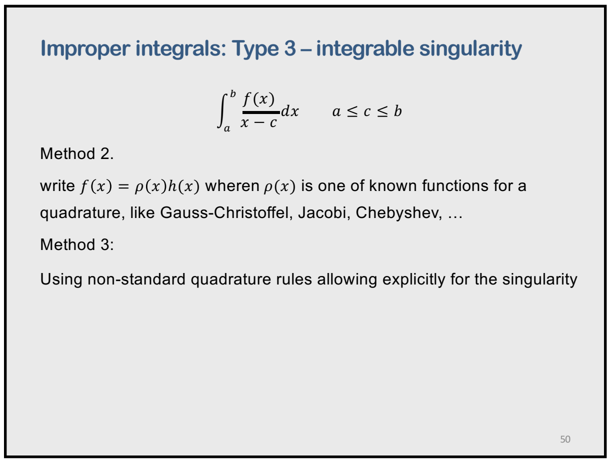
Example: problem at $x = 0$

$$f(x) = \frac{1}{\sqrt{x(1+x^2)}}$$

$$f(x) = \frac{1}{\sqrt{x(1+x^2)}} = \left(\frac{1}{\sqrt{x(1+x^2)}} - \frac{1}{\sqrt{x}}\right) + \frac{1}{\sqrt{x}}$$

49

49



Improper integrals: Type 3 – integrable singularity

$$\int_a^b \frac{f(x)}{x-c}dx \qquad a \le c \le b$$

Method 2.

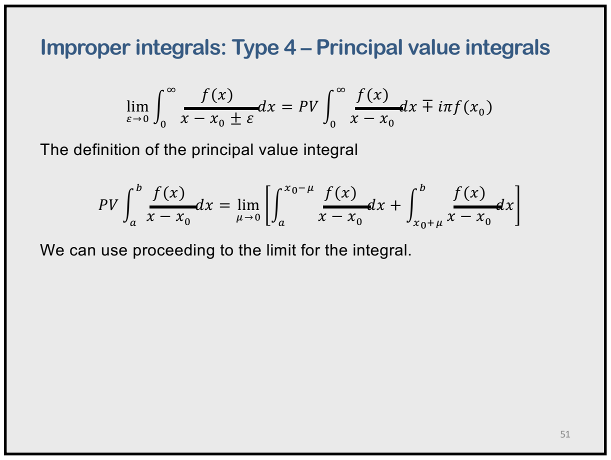
write $f(x) = \rho(x)h(x)$ wheren $\rho(x)$ is one of known functions for a quadrature, like Gauss-Christoffel, Jacobi, Chebyshev, …

Method 3:

Using non-standard quadrature rules allowing explicitly for the singularity

50

50



Improper integrals: Type 4 – Principal value integrals

$$\lim_{\varepsilon\to 0}\int_0^\infty \frac{f(x)}{x-x_0 \pm \varepsilon}dx = PV\int_0^\infty \frac{f(x)}{x-x_0}dx \mp i\pi f(x_0)$$

The definition of the principal value integral

$$PV\int_a^b \frac{f(x)}{x-x_0}dx = \lim_{\mu\to 0}\left[\int_a^{x_0-\mu} \frac{f(x)}{x-x_0}dx + \int_{x_0+\mu}^b \frac{f(x)}{x-x_0}dx\right]$$

We can use proceeding to the limit for the integral.

51

51

Improper integrals: Type 4 – Principal value integrals

Suppose $f(x)$ is unbounded in the neighborhood of $x = c$ so that the principal value of the integral is

$$PV\int_a^b f(x)dx = \lim_{\mu\to 0}\left[\int_a^{x_0-\mu} f(x)dx + \int_{x_0+\mu}^b f(x)dx\right]$$

Let us take $c = 0$, and the limits of the integral are from $-a$ to a.

$$g(x) = \frac{1}{2}[f(x) - f(-x)], \qquad h(x) = \frac{1}{2}[f(x) + f(-x)]$$

It is possible to show that (see Davis and Rabinowitz, page 182)

$$PV\int_{-a}^{a} f(x)dx = \int_{-a}^{a} (f(x) + f(-x))dx$$

52

52

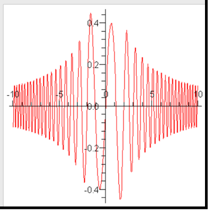
Integration of rapidly oscillatory functions

Example

$$\int_a^b f(x)\cos nx\, dx$$

Recommendation: use methods or programs specially designed to calculate integrals with oscillating functions, e.g., Filon's method, Clenshaw-Curtis method, …

Methods: integration between zeros,
use of Gauss rules for Fourier coefficients,
use of Chebyshev and Legendre expansions
and many more



53

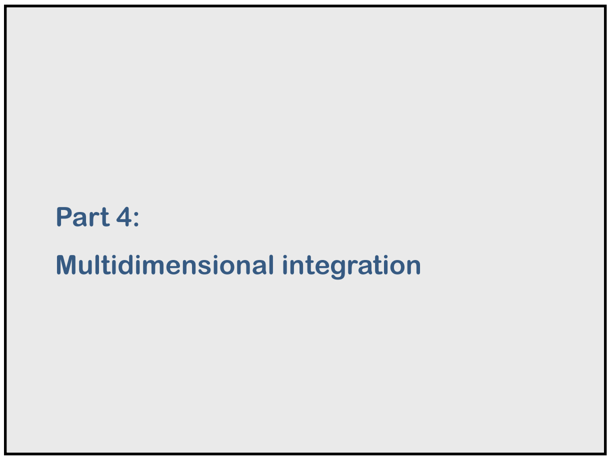
Other cases

- Integration of periodic functions*
- Slowly convergent integrals
- Singular integrals
- Integrating tabular data
- Contour integrals
- Indefinite integrals (integration via differential equations)

*see also Fast Fourier Transform (FFT)

54

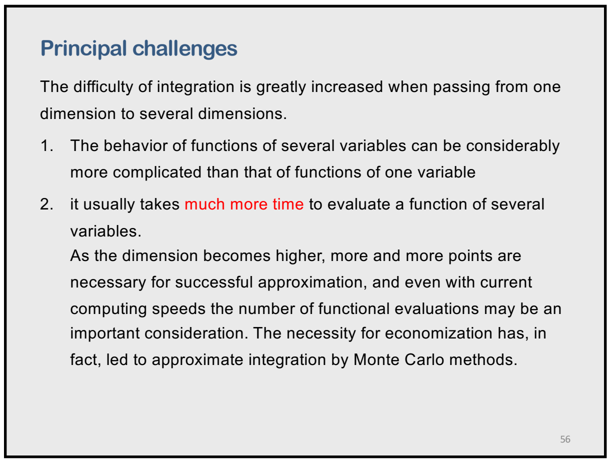
54



Part 4:

Multidimensional integration

55



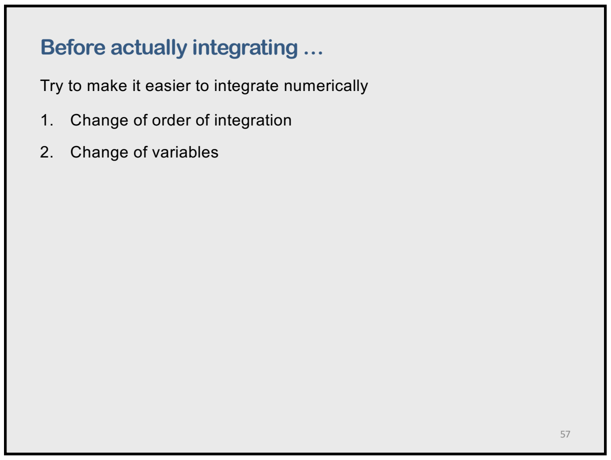
Principal challenges

The difficulty of integration is greatly increased when passing from one dimension to several dimensions.

1. The behavior of functions of several variables can be considerably more complicated than that of functions of one variable
2. it usually takes much more time to evaluate a function of several variables.
   As the dimension becomes higher, more and more points are necessary for successful approximation, and even with current computing speeds the number of functional evaluations may be an important consideration. The necessity for economization has, in fact, led to approximate integration by Monte Carlo methods.

56

56



Before actually integrating …

Try to make it easier to integrate numerically

1. Change of order of integration
2. Change of variables

57

57

Principal strategies for nD integration

1. Use automatic one-dimensional quadrature routine for each dimension, one for outer integral another for inner integral, etc.
2. Use Monte Carlo method
3. For 2D integrals it's possible to use cubature integration (e.g. cubature trapezoid rule as a product of one-dimensional rules)

Some numerical libraries have routines for 2D and 3D integration

For example, Matlab has integral2, integral3.

58

58

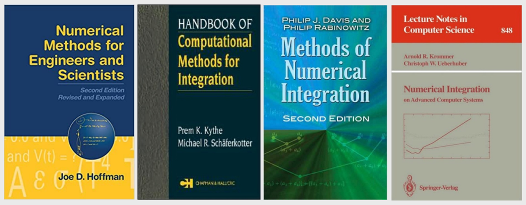
Example: 2D sequential integration

$$\int_0^1 dx \int_0^{\sin(x)} (x+y)^2\, dy = 0.48258512$$

```
! MAIN PART
a = 0.0
b = 1.0
eps = 1.0e-7
integral = simpson2D(f,a,b,eps)
stop
End

Function f(x)
common/fxy/x1
external g
c = 0.0
d =  sin(x)
x1 = x                     ! x1 passes the value of x to g(y)
 f = simpson2D(g,c,d,eps)
return
end
Function g(y)
common/fxy/x
g = (x+y)**2
return
end
```

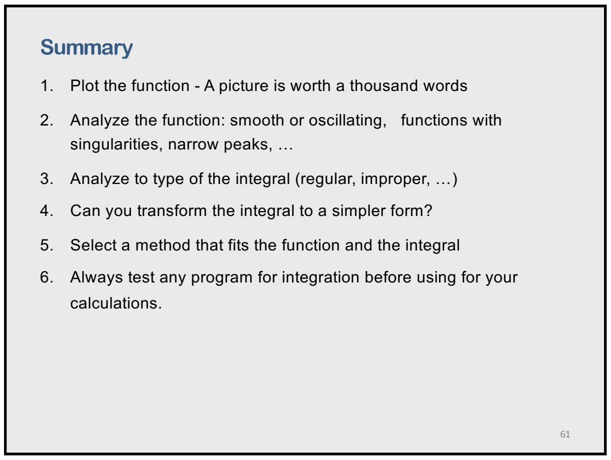
59

Very practical books


Numerical Methods for Engineers and Scientists
Second Edition Revised and Expanded
Joe D. Hoffman
HANDBOOK OF Computational Methods for Integration
Prem K. Kythe
Michael R. Schäferkotter
PHILIP J. DAVIS AND PHILIP RABINOWITZ
Methods of Numerical Integration
SECOND EDITION
Lecture Notes in Computer Science 848
Arnold R. Krommer
Christoph W. Ueberhuber
Numerical Integration on Advanced Computer Systems
Springer-Verlag

60

60



Summary

1. Plot the function - A picture is worth a thousand words
2. Analyze the function: smooth or oscillating,   functions with singularities, narrow peaks, …
3. Analyze to type of the integral (regular, improper, …)
4. Can you transform the integral to a simpler form?
5. Select a method that fits the function and the integral
6. Always test any program for integration before using for your calculations.

61

61