Computational Physics (PHYS6350)

Lecture 9: Numerical Integration: Part 2

- High-order quadrature
- Gaussian quadrature

Instructor: Volodymyr Vovchenko (vvovchenko@uh.edu)

Course materials: https://github.com/vlvovch/PHYS6350-ComputationalPhysics
Online textbook: https://vovchenko.net/computational-physics/

Numerical integration so far

- Rectangle rule

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

- Trapezoidal rule

$$\int_a^b f(x)\,dx \approx (b-a)\,\frac{f(a)+f(b)}{2}$$

- Simpson's rule

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

All can be written as

$$\int_a^b f(x)\,dx \approx \sum_k w_k f(x_k)$$

Integrating the interpolating polynomial

There is a systematic way to derive a numerical integration scheme

$$\int_a^b f(x)\,dx \approx \sum_k w_k f(x_k)$$

which will give an exact result when *f(x)* is a polynomial up to a certain degree.

Recall the interpolating polynomial through *N+1* points where *f(x)* can be evaluated

$$f(x) \approx p_N(x) = \sum_{k=0}^{N} f(x_k)\, L_{N,k}(x)$$

$$L_{N,k}(x) = \prod_{j \neq k} \frac{x - x_j}{x_k - x_j}$$

Lagrange basis functions

Then, the integral reads

$$\int_a^b f(x)\,dx \approx \int_a^b p_N(x)\,dx = \sum_{k=0}^{N} w_k f(x_k)$$

where

$$w_k = \int_a^b L_{N,k}(x)dx$$

This expression is exact when *f(x)* is a polynomial up to degree *N*

Newton-Cotes quadratures

$$\int_a^b f(x)\,dx \approx \int_a^b p_N(x)\,dx = \sum_{k=0}^{N} w_k f(x_k)$$

with x_k distributed equidistantly

- Closed Newton-Cotes (include the endpoints)

$$x_k = a + hk, \qquad k = 0 \dots N, \qquad h = (b-a)/N$$

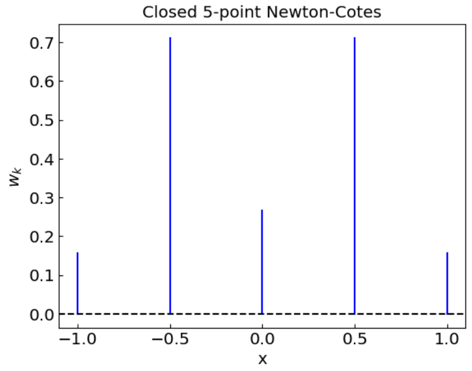
N = 1: trapezoidal

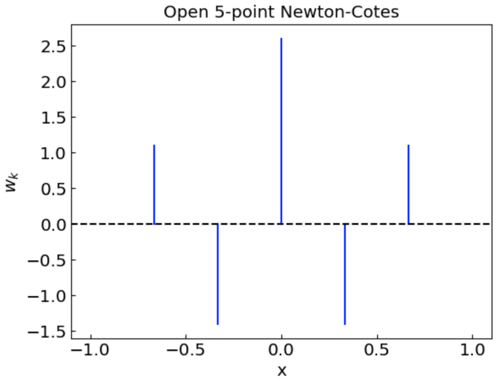
N = 2: Simpson

- Open Newton-Cotes (exclude the endpoints)

$$x_k = a + hk, \qquad k = 1 \dots N+1, \qquad h = (b-a)/(N+2)$$

N = 0: rectangle rule





Newton-Cotes quadratures

The weights can be computed just once using one of the earlier methods (e.g. Romberg)

$$w_k = \int_a^b L_{N,k}(x)dx$$

```python
# Calculating the weights using the Romberg method
# to requested accuracy for a given set of nodes x
# over the interval (a,b)
def compute_weights(x,
                    a,
                    b,
                    tol = 1.e-15):
    ret = []
    for k in range(0,len(x)):
        tx = x
        def f(t):
            return Lnj(t, len(x) - 1, k, x)
        ret.append(romberg(f, a, b, tol))
    return ret
```

```python
# Calculate the nodes and weights of either
# closed or open Newton-Cotes quadrature
# to requested accuracy
def newton_cotes(n,
                 a = -1.,
                 b = 1.,
                 isopen = False,
                 tol = 1.e-15):
    x = []
    if (isopen):
        h = (b - a) / (n + 2.)
        x = [a + (i+1)*h for i in range(0,n+1)]
    else:
        h = (b - a) / n
        x = [a + i*h for i in range(0,n+1)]
    return x, compute_weights(x, a, b, tol)
```

Newton-Cotes quadratures: example

$$I = \int_0^2 x^4 - 2x + 2 = 6.4$$

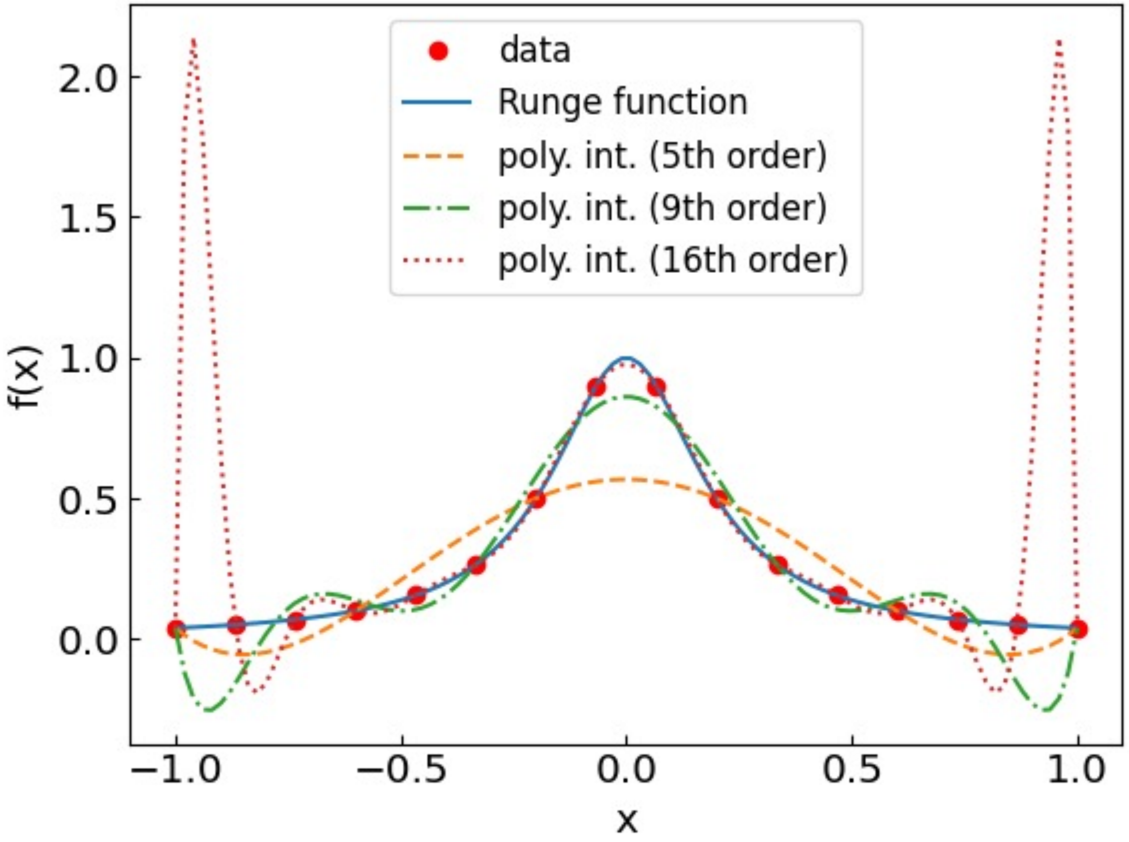
```
Computing the integral of x^4 - 2x + 2 over the interval ( 0.0 , 2.0 ) using open Newton-Cotes quadratures
         N                   I_N
         0    2.0000000000000000
         1    3.3580246913580254
         2    6.1666666666666661
         3    6.2378666666666671
         4    6.4000000000000039
         5    6.3999999999999986
         6    6.4000000000000021
         7    6.4000000000000039
```

```
Computing the integral of x^4 - 2x + 2 over the interval ( 0.0 , 2.0 ) using closed Newton-Cotes quadratures
         N                   I_N
         1   16.0000000000000000
         2    6.6666666666666661
         3    6.5185185185185182
         4    6.4000000000000004
         5    6.4000000000000012
         6    6.3999999999999986
         7    6.4000000000000004
```

Exact result (to machine precision) from N = 4

Newton-Cotes quadratures: Runge phenomenon

Recall the Runge function: $$f(x) = \frac{1}{1 + 25x^2}$$



Newton-Cotes quadratures: Runge phenomenon

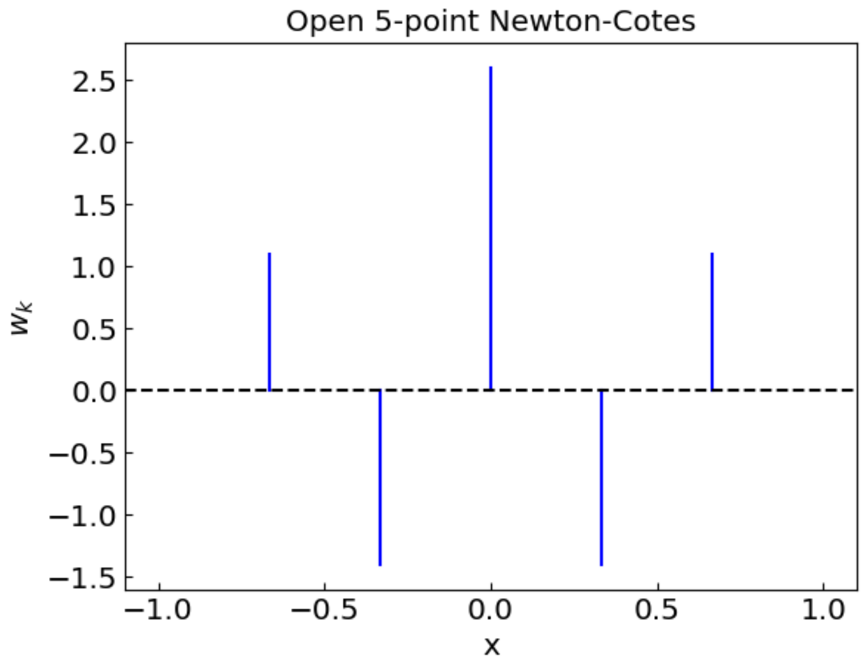
$$I = \int_{-1}^{1} \frac{dx}{1+25x^2} = 0.5493603\ldots$$

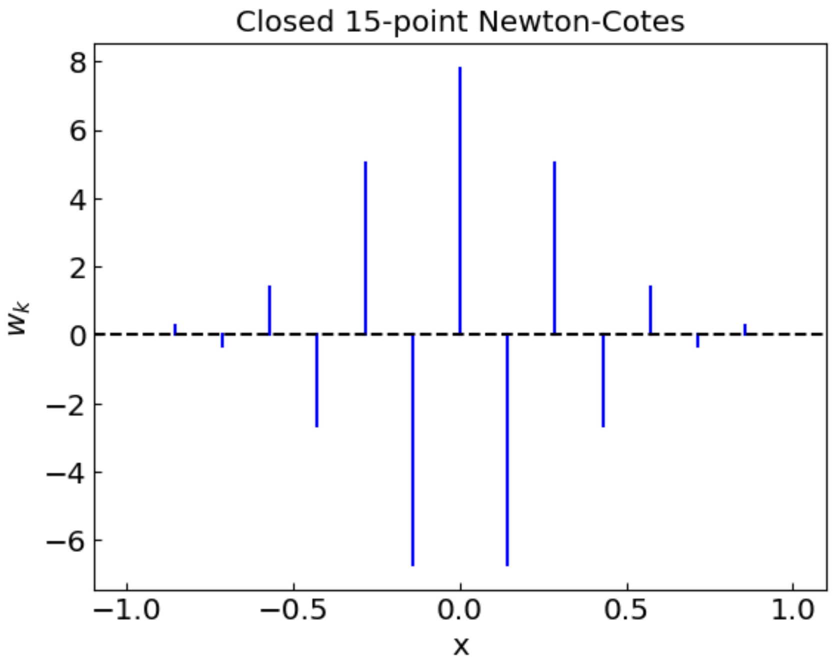
```
Computing the integral of Runge function over the interval ( -1.0 , 1.0 ) using open Newton-Cotes quadratures
         N                  I_N
         0    2.0000000000000000
         1    0.5294117647058825
         2   -0.2988505747126436
         3    0.2666666666666667
         4    2.0404749055585549
         5    0.9320668542657328
         6   -2.0045340869981669
         7   -0.1816307907657775

Computing the integral of Runge function over the interval ( -1.0 , 1.0 ) using closed Newton-Cotes quadratures
         N                  I_N
         1    0.0769230769230769
         2    1.3589743589743588
         3    0.4162895927601810
         4    0.4748010610079575
         5    0.4615384615384615
         6    0.7740897346941600
         7    0.5797988819496757
         8    0.3000977814255821
         9    0.4797235795683667
        10    0.9346601111306989
```

Romberg method

```
Computing the integral of Runge function over the interval ( -1.0 , 1.0 ) using Romberg method
0.549360306777909
```

Newton-Cotes quadratures: oscillating weights





For large N one has highly oscillatory weights

- Manifestation of the Runge phenomenon
- Another issue: round-off error due to large cancellations

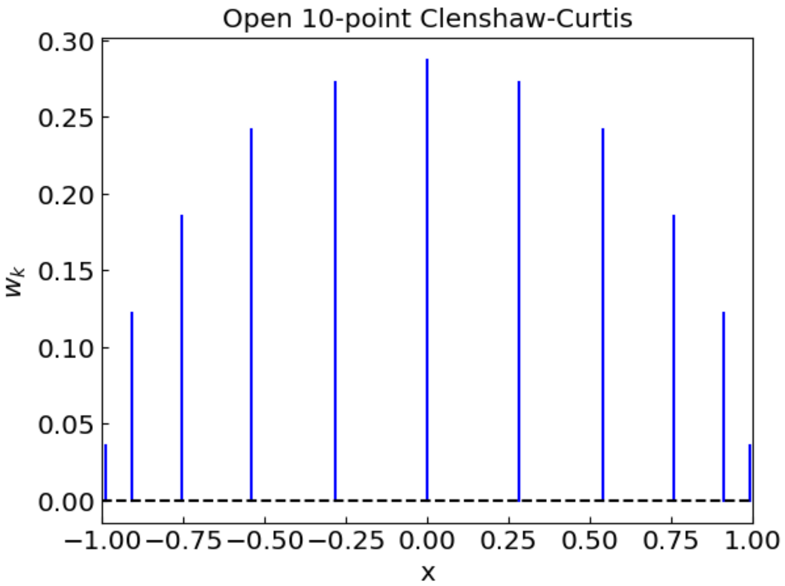
Clenshaw-Curtis quadrature

Chebyshev nodes minimize the Runge phenomenon

$$x_k = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{2k+1}{2n+2}\pi\right), \qquad k = 0, \dots, n,$$

The corresponding quadrature is called **Clenshaw-Curtis**

Weights*:



*For efficient calculation use discrete cosine transform

Clenshaw-Curtis quadrature

$$I = \int_{-1}^{1} \frac{dx}{1+25x^2} = 0.5493603\ldots$$

```
Computing the integral of Runge function over the interval ( -1.0 , 1.0 ) using closed Clenshaw-Curtis quadratures
         N                   I_N
         0    2.0000000000000000
         1    0.1481481481481482
         2    1.1561181434599159
         3    0.3393357342937174
         4    0.7366108212029662
         5    0.4422623071358261
         6    0.6363602552248223
         7    0.4995830749190563
         8    0.5839263513091471
         9    0.5259711610228502
        10    0.5661564732597759
        11    0.5388727075897808
        12    0.5562316021895978
        13    0.5445109449451719
        14    0.5527811219474377
        15    0.5472112438100144
        16    0.5507349751776419
        17    0.5483645031315995
        18    0.5500702958302579
        19    0.5489233775473977
        20    0.5496321498366133
        21    0.5491557069456035
        22    0.5495101923607436
        23    0.5492719294992719
        24    0.5494126772553229
```

Gaussian quadrature

We have seen that an *n*-point quadrature

$$\int_a^b f(x)\,dx \approx \sum_k w_k f(x_k)$$

gives the exact result when *f(x)* is a polynomial of degree up to *n-1.*

This is true *any* choice of distinct nodes x_k.

We have the freedom to choose the locations of nodes x_k,
which gives us additional n degrees of freedom.

It turns out this can be exploited to obtain a quadrature that is
exact when f(x) is a polynomial up to degree *2n-1*.

The corresponding quadrature is called **Gaussian quadrature**

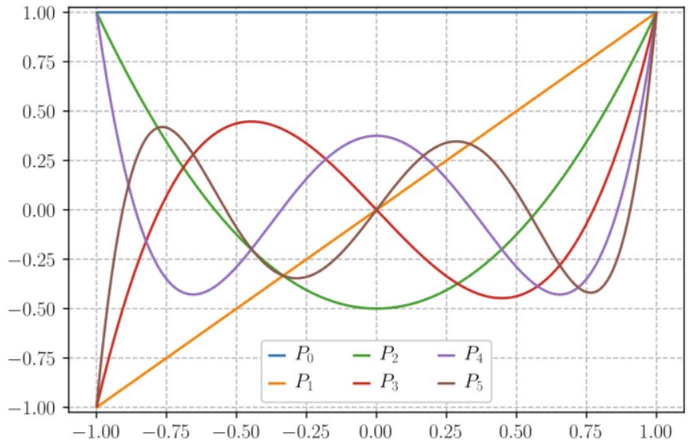
Gauss-Legendre quadrature

Let us focus on the interval *(-1,1)*. It can always be mapped to *(a,b)* by a transformation

$$x_k \to \frac{a+b}{2} + \frac{b-a}{2} x_k \,, \qquad w_k \to \frac{b-a}{2} w_k \,.$$

Gauss-Legendre quadrature:

$$\int_{-1}^{1} f(x)dx \approx \sum_{k=1}^{n} w_k f(x_k)$$



where x_k are the roots of the *Legendre polynomial* $P_n(x)$

and the weights are given by

$$w_k = \int_{-1}^{1} L_{n-1,k}(x)dx = \frac{2}{(1-x_k^2)[P_n'(x_k)]^2} \,.$$

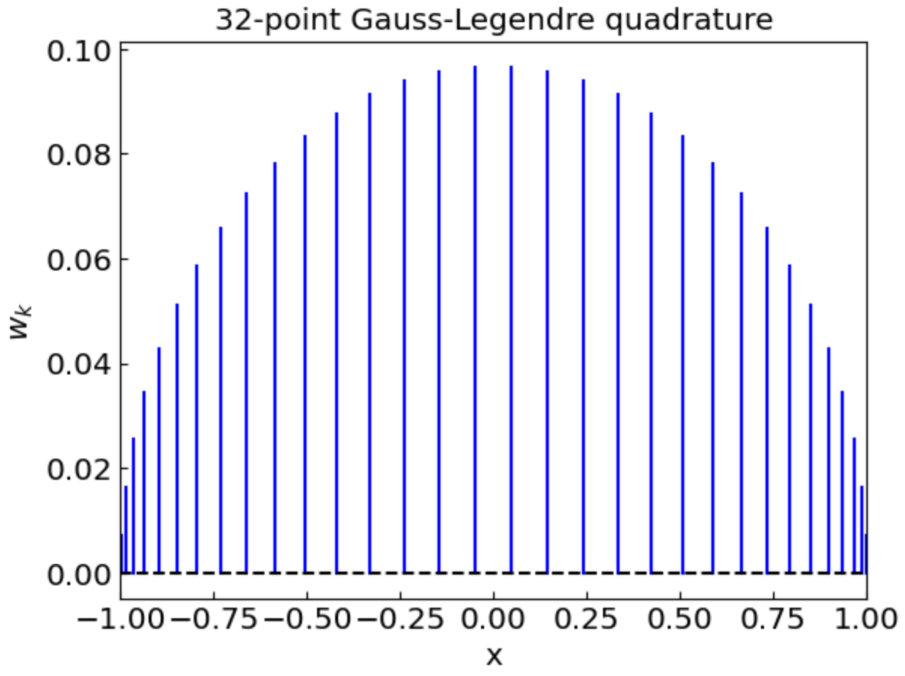
Gauss-Legendre quadrature

How to find the nodes x_k and weights w_k?

In general, we can use PolyRoots to find x_k and e.g. Romberg method for w_k

For the Gauss-Legendre quadrature a more efficient procedure exists
(see e.g. http://www-personal.umich.edu/~mejn/cp/programs/gaussxw.py)

```python
from numpy import ones,copy,cos,tan,pi,linspace

def gaussxw(N):

    # Initial approximation to roots of the Legendre polynomial
    a = linspace(3,4*N-1,N)/(4*N+2)
    x = cos(pi*a+1/(8*N*N*tan(a)))

    # Find roots using Newton's method
    epsilon = 1e-15
    delta = 1.0
    while delta>epsilon:
        p0 = ones(N,float)
        p1 = copy(x)
        for k in range(1,N):
            p0,p1 = p1,((2*k+1)*x*p1-k*p0)/(k+1)
        dp = (N+1)*(p0-x*p1)/(1-x*x)
        dx = p1/dp
        x -= dx
        delta = max(abs(dx))

    # Calculate the weights
    w = 2*(N+1)*(N+1)/(N*N*(1-x*x)*dp*dp)

    return x,w
```



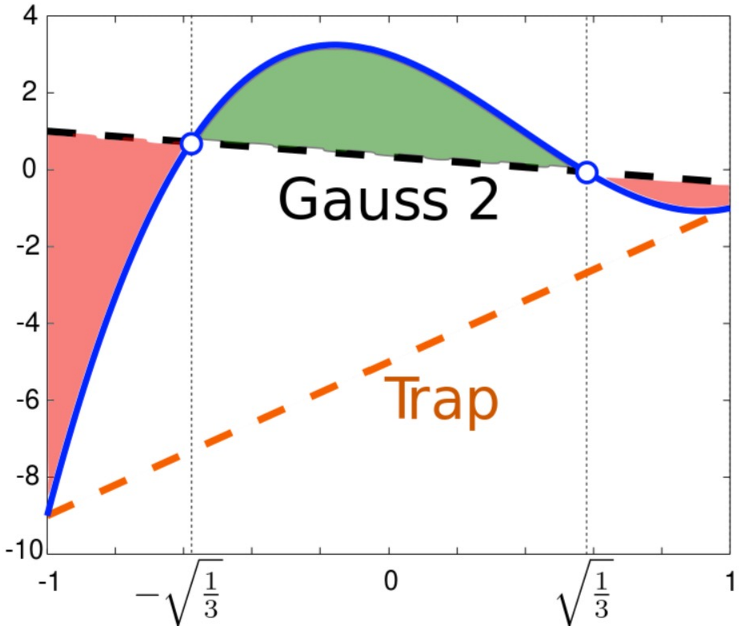
Gauss-Legendre quadrature: polynomials

$$I = \int_0^2 x^4 - 2x + 2 = 6.4$$

```
Computing the integral of x^4 - 2x + 2 over the interval ( 0.0 , 2.0 ) using Gauss-Legendre quadratures
          N                    I_N
          1    1.9999999999999969
          2    6.2222222222222303
          3    6.4000000000000066
          4    6.4000000000000208
          5    6.4000000000000190
          6    6.4000000000000021
          7    6.4000000000000083
```

$$I = \int_{-1}^1 (7x^3 - 8x^2 - 3x + 3) = \frac{2}{3}$$

```
Computing the integral of 7x^3-8x^2-3x+3 over the interval ( -1.0 , 1.0 )
    Trapezoidal: -10.0
Clenshaw-Curtis: -2.0
 Gauss-Legendre: 0.6666666666666641
```



Generalized Gaussian quadratures

The method of Gaussian quadratures can be generalized to integrals of the following type

$$\int_a^b \omega(x) f(x) dx \approx \sum_{k=1}^{n} w_k f(x_k) \qquad\qquad \omega(x) \text{ – weight function}$$

In this case it is possible to construct an *n*-point quadrature that provides the exact answer when *f(x)* is a polynomial of degree up to *2n - 1*. The weights w_k are given by

$$w_k = \int_a^b \omega(x)\, L_{n-1,k}(x)\, dx$$

and the nodes x_k are the roots of a polynomial $p_n(x)$ satisfying

$$\int_a^b \omega(x)\, x^k\, p_n(x)\, dx = 0, \qquad k = 0, \dots, n-1$$

For $a = -1$, $b = 1$, $\omega(x) = 1$ we have **Gauss-Legendre** quadrature

For $a = -1$, $b = 1$, $\omega(x) = (1-x)^\alpha (1+x)^\beta$ we have **Gauss-Jacobi** quadrature

Generalized Gaussian quadratures

The interval *(a,b)* does not have to be finite

- **Gauss-Laguerre quadrature**

$$\int_0^\infty e^{-x} f(x) dx \approx \sum_{k=1}^{n} w_k f(x_k) \, .$$

x_k are the roots of Laguerre polynomial $L_n(x)$

Example: Fermi-Dirac/Bose-Einstein integrals in relativistic systems

- **Gauss-Hermite quadrature**

x_k are the roots of Hermite polynomial $H_n(x)$

$$\int_{-\infty}^\infty e^{-x^2} f(x) dx \approx \sum_{k=1}^{n} w_k f(x_k) \, .$$

Example: Expectation value of a function of a normally distributed random variable

Another approach: map (semi-)infinite interval to (-1,1) and use the Gauss-Legendre quadrature

Summary: Choosing the integration method

- Rectangle/trapezoidal rule
  - Good for quick calculations not requiring great accuracy
  - Does not rely on the integrand being smooth; a good choice for noisy/singular integrands, equally spaced points

- Romberg method
  - Control over error
  - Good for relatively smooth functions evaluated at equidistant nodes

- Gaussian quadrature
  - Theoretically most accurate if the function is relatively smooth
  - Good for many repeated calculations of the same type of integral
  - Requires unequally spaced nodes
  - Error can be challenging to control, especially for non-smooth functions
    - Gauss-Kronrod quadrature gives control over error
  - Bad for discontinuous integrands

final project idea(?)