

Numerical Integration

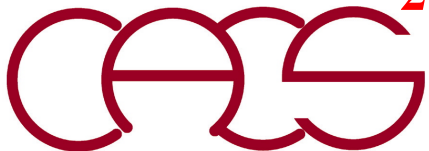
Aiichiro Nakano

*Collaboratory for Advanced Computing & Simulations
Department of Computer Science
Department of Physics & Astronomy
Department of Chemical Engineering & Materials Science
Department of Quantitative & Computational Biology
University of Southern California*

Email: anakano@usc.edu

New toolbox (use it! it's user friendly):

- 1. Gaussian quadratures (orthogonal functions)**
- 2. Newton's method**



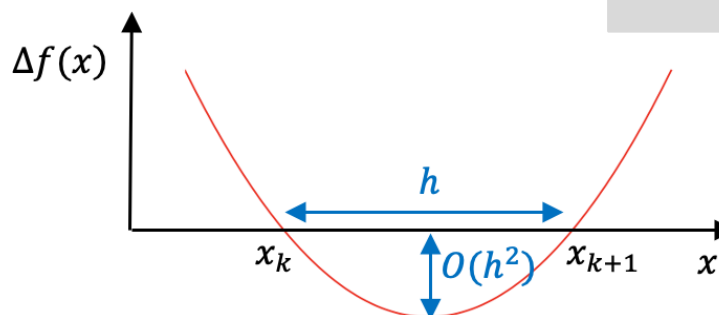
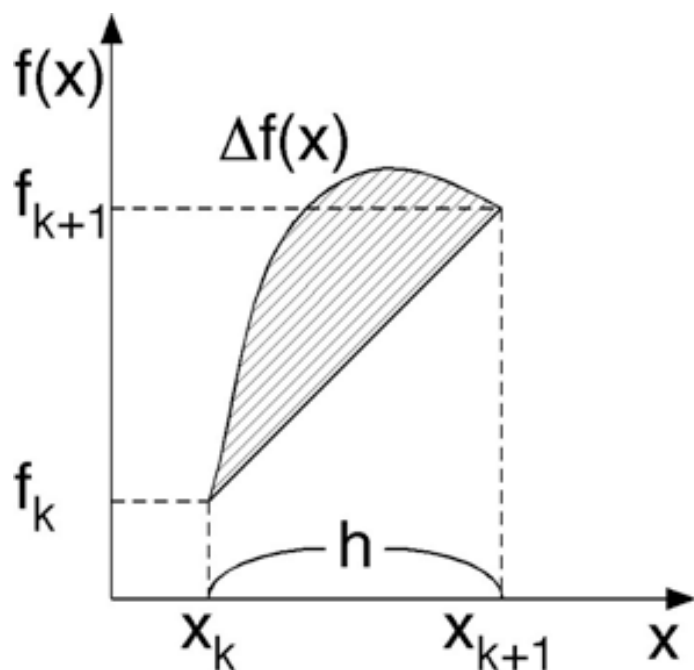
Numerical Integration

- Numerical integration = weighted sum of function values**

$$S = \int_a^b f(x)dx \cong \sum_{k=0}^{n-1} w_k f(x_k)$$

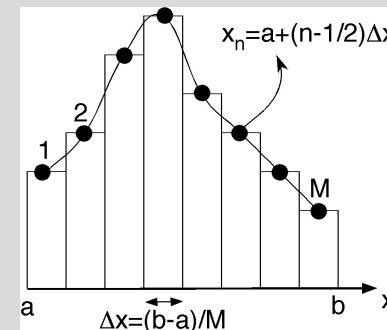
- Trapezoid quadrature: Piecewise linear approximation**

$$f(x) \cong f_k + (x - x_k)(f_{k+1} - f_k)/h \quad x \in [x_k, x_{k+1}]$$



$$\text{error} = \text{height } (O(h^2)) \times \text{base } (h) \times \# \text{ of bins } \left(\frac{b-a}{h} \right) = O(h^2)$$

Resulting area: $\int_a^b f(x)dx \cong \frac{h}{2} \sum_{k=0}^{n-1} (f_k + f_{k+1}) + O(h^2)$



Piecewise constant
 $O(h)$ approximation

$$\begin{cases} x_k = kh = (b-a)k/n \\ w_k = h \end{cases}$$

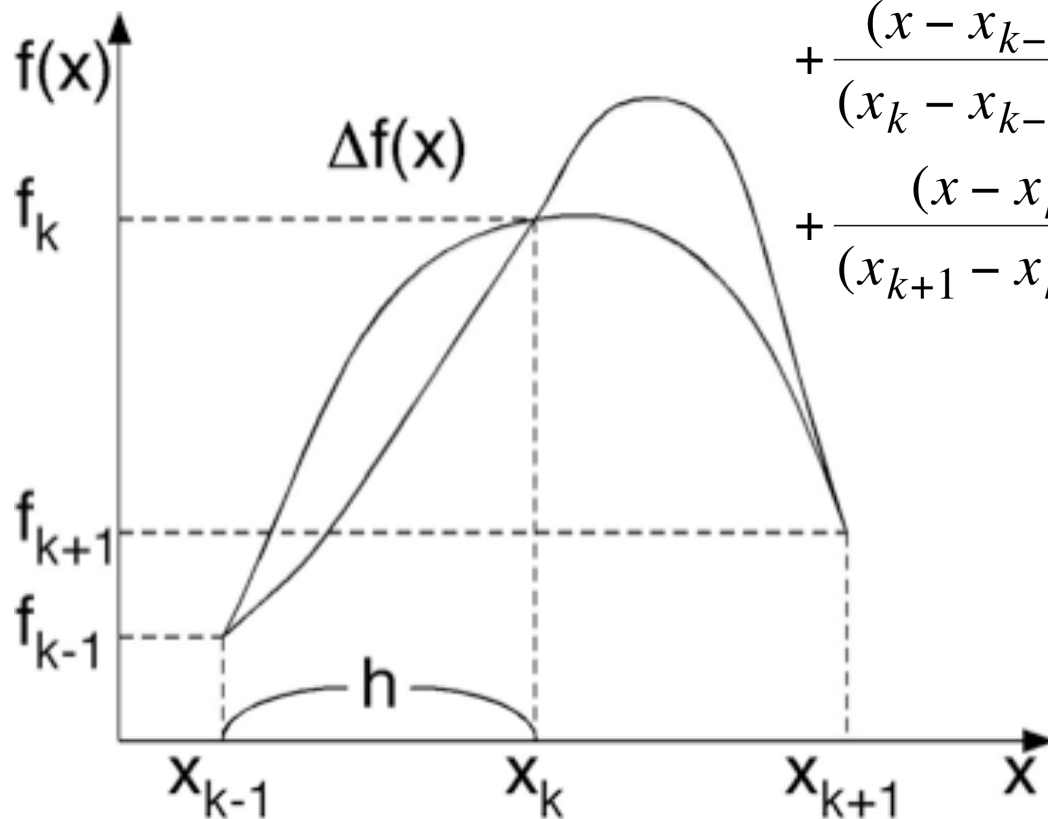
Simpson Rule

- **Simpson quadrature: Piecewise quadratic approximation**

- **Lagrange interpolation:** $f(x) \cong \frac{(x - x_k)(x - x_{k+1})}{(x_{k-1} - x_k)(x_{k-1} - x_{k+1})} f_{k-1}$

$$+ \frac{(x - x_{k-1})(x - x_{k+1})}{(x_k - x_{k-1})(x_k - x_{k+1})} f_k$$

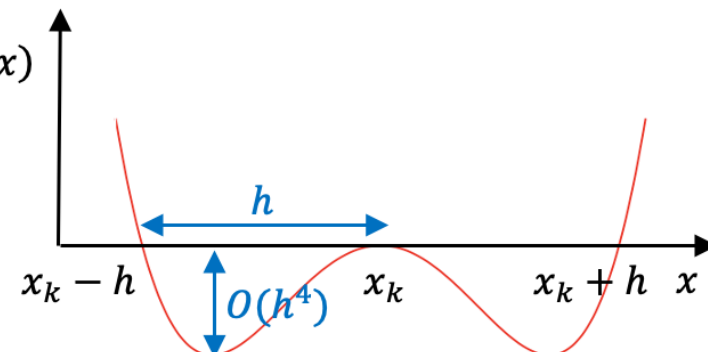
$$+ \frac{(x - x_{k-1})(x - x_k)}{(x_{k+1} - x_{k-1})(x_{k+1} - x_k)} f_{k+1}$$



$$\int_{x_{k-1}}^{x_{k+1}} \Delta f(x) dx = 0$$

The equation shows the integral of the error $\Delta f(x)$ over the interval $[x_{k-1}, x_{k+1}]$ is zero. The error is labeled $\Delta f(x) = c(x - h)x(x + h)$.

$$\int_a^b f(x) dx \cong \frac{h}{3} \sum_{l=0}^{n/2-1} (f_{2l} + 4f_{2l+1} + f_{2l+2}) + O(h^4)$$



Gaussian Quadratures

- **Idea of Gaussian quadrature:** Freedom to choose both weighting coefficients & the location of abscissas to evaluate the function
- **Gaussian quadrature:** Chooses the weight & abscissas to make the integral exact for a class of integrands “polynomials times some known function $W(x)$ ”.

> **Gauss-Legendre:** $W(x) = 1; -1 < x < 1$

> **Gauss-Chebyshev:** $W(x) = (1 - x^2)^{-1/2}; -1 < x < 1$

...

$$\int_a^b W(x) f(x) dx = \sum_{k=1}^n w_k f(x_k)$$

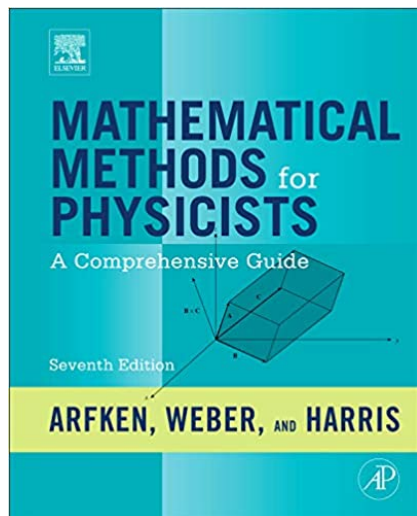
- **New toolbox:** (1) **orthogonal functions** (recursive generation via a generating function); (2) **Newton method** for root finding

See [gauleg-driver.c](#) & [gauleg.c](#)

W.H. Press, B.P. Flannery, S.A. Teukolsky, & W.T. Vetterling,
Numerical Recipes, 2nd Ed. (Cambridge U Press, '93), Sec. 4.5

Orthogonal Functions

- Gaussian quadratures are defined through orthogonal functions
- Orthogonal functions are often introduced as solutions to differential equations
- **Examples:** Legendre, Bessel, Laguerre, Hermite, Chebyshev, ...
- Operationally well-defined to compute the function values & derivatives
- Efficiently computable through **recursive relations** (more than elementary functions like $\sin(x)$, $\exp(x)$, ...)



13	Gamma Function	599
14	Bessel Functions	643
15	Legendre Functions	715

Orthogonal Functions

- **Scalar product:**

$$\langle f | g \rangle \equiv \int_a^b W(x) f(x) g(x) dx$$

- **Orthonormal set of functions:** Mutually orthogonal & normalized

$$\langle p_m | p_n \rangle = \delta_{m,n} = \begin{cases} 1, & m = n \\ 0, & m \neq n \end{cases}$$

- **Recurrence relation to construct an orthonormal set:**

$$p_{-1}(x) \equiv 0$$

$$p_0(x) \equiv 1$$

$$p_{j+1}(x) = (x - a_j)p_j(x) - b_j p_{j-1}(x) \quad j = 0, 1, 2, \dots$$

$$a_j = \frac{\langle x p_j | p_j \rangle}{\langle p_j | p_j \rangle} \quad j = 0, 1, \dots$$

$$b_j = \frac{\langle p_j | p_j \rangle}{\langle p_{j-1} | p_{j-1} \rangle} \quad j = 1, 2, \dots$$

(Theorem) $p_j(x)$ has exactly j distinct roots in (a,b) , & the roots interleave the $j-1$ roots of $p_{j-1}(x)$

Legendre Polynomial

$$W(x) = 1 \quad -1 < x < 1$$

- **Recursive function evaluation**

$$(j+1)P_{j+1} = (2j+1)xP_j - jP_{j-1} \quad P_0 = 1 \quad P_1 = x$$

- **Generating function:** The recurrence may be obtained through the Taylor expansion of the following function with respect to t

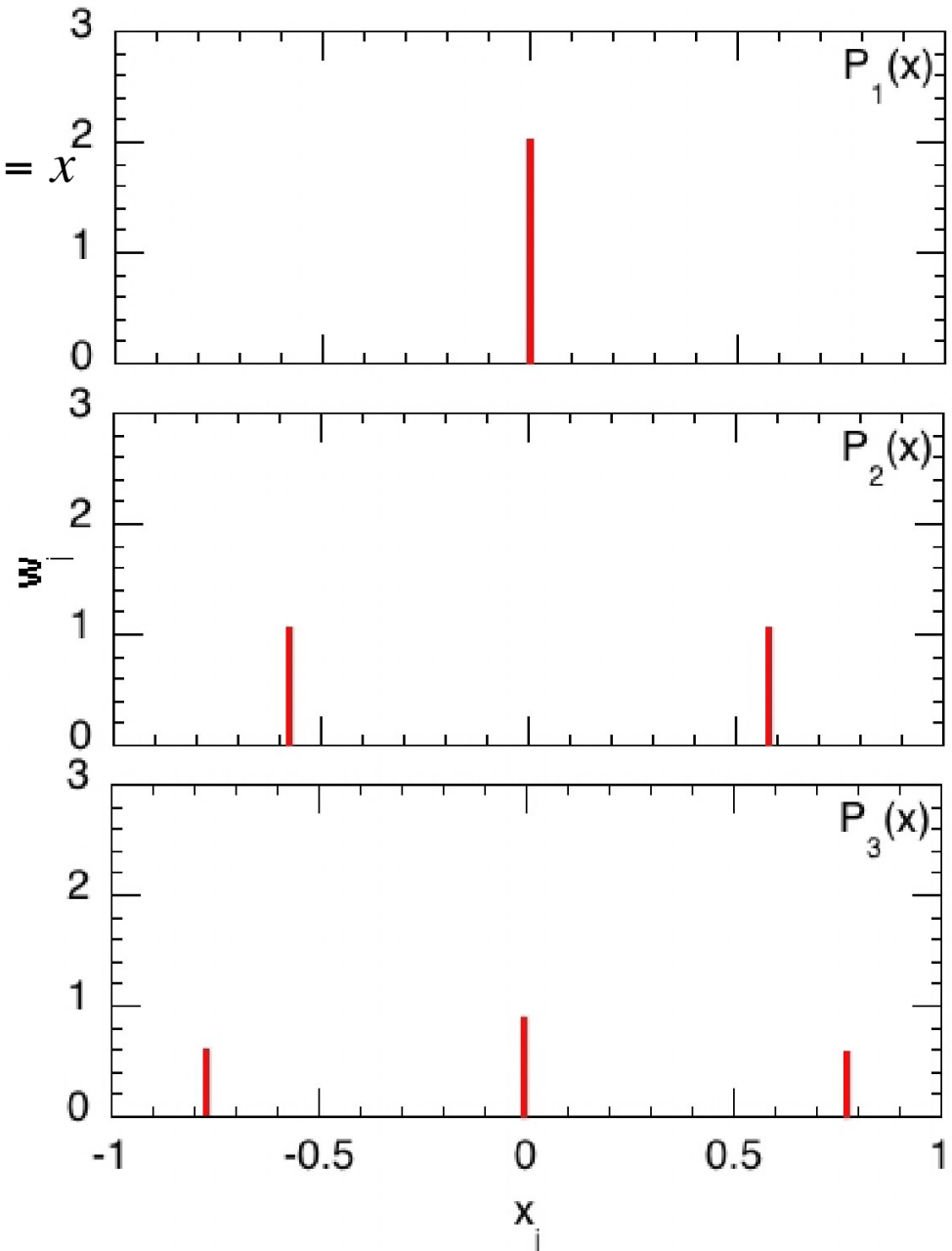
$$g(t, x) = \frac{1}{\sqrt{1 - 2xt + t^2}} \equiv \sum_{j=0}^{\infty} P_j(x) t^j$$

(Hint) Differentiate both sides by t & compare the coefficients of t^j

- **Function derivative:** A recurrence derived by differentiating g by x

$$(x^2 - 1)P'_j = jxP_j - jP_{j-1}$$

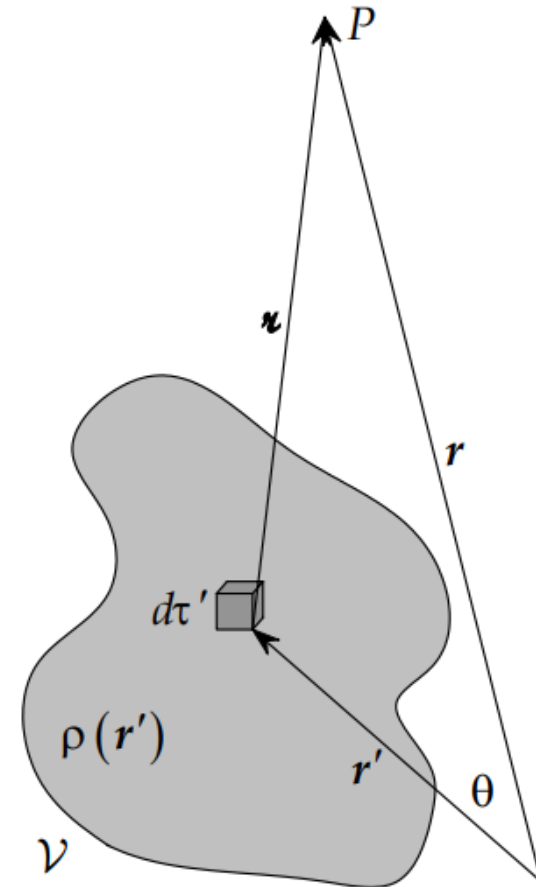
See lecture on [recursive formula for Legendre polynomials](#)



Origin of Legendre Polynomial

- Generating function of the Legendre polynomial is used for multipole expansion in electrostatics

$$\begin{aligned}\frac{1}{\sqrt{\vec{r} - \vec{r}'}} &= \frac{1}{\sqrt{r^2 - 2rr'\cos\theta + r'^2}} \\ &= \frac{1}{r \sqrt{1 - 2\frac{r'}{r}\cos\theta + \left(\frac{r'}{r}\right)^2}} \\ &= \frac{1}{r} \sum_{j=0}^{\infty} P_j(\cos\theta) \left(\frac{r'}{r}\right)^j\end{aligned}$$



See lecture note on [O\(N\) fast multipole method](#)

Open-source code: S. Ogata *et al.*, [Comput. Phys. Commun](#) **153**, 445 ('03)

Gauss-Legendre Quadrature

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

- **Abscissae from roots, x_k**

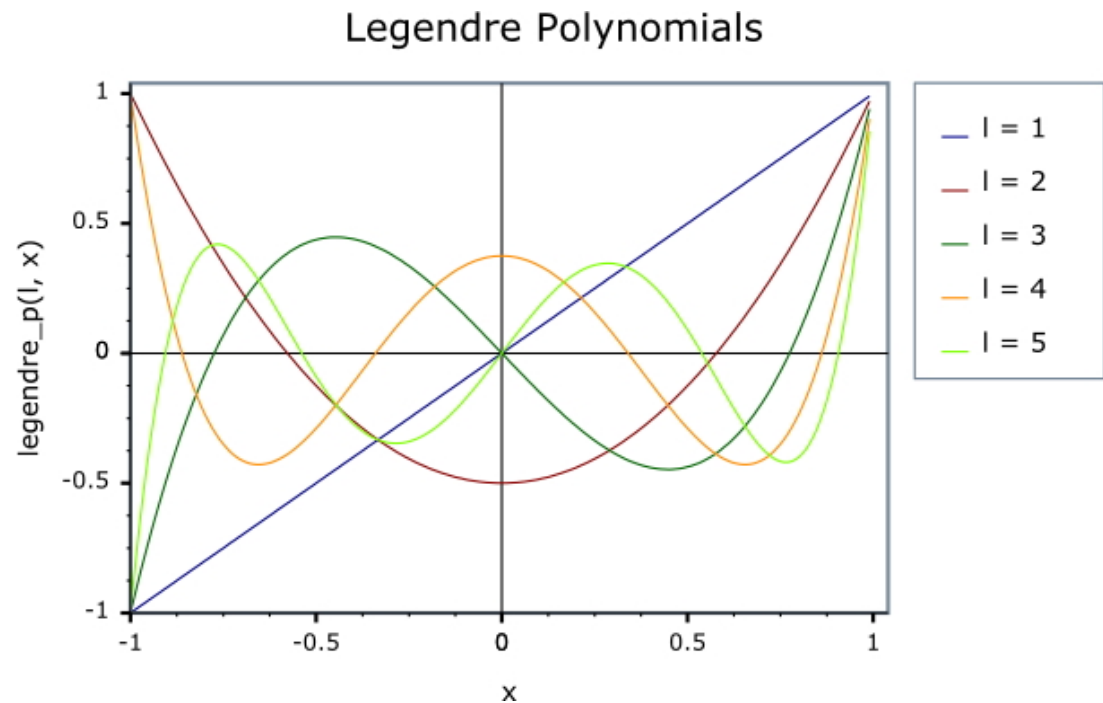
$$P_n(x_k) = 0 \quad k = 1, \dots, n$$

- **Weights, w_k : To reproduce some integrals exactly (linear equation)**

$$\int_{-1}^1 P_0(x) P_n(x) dx = \frac{2}{2n+1} \delta_{0,n} = \sum_{k=1}^n w_k P_n(x_k)$$

or

$$w_k = \frac{2}{n P_{n-1}(x_k) P'_n(x_k)} = \frac{2}{(1-x_k^2) [P'_n(x_k)]^2}$$



Newton's Method for Root Finding

- **Problem:** Find a root of a function

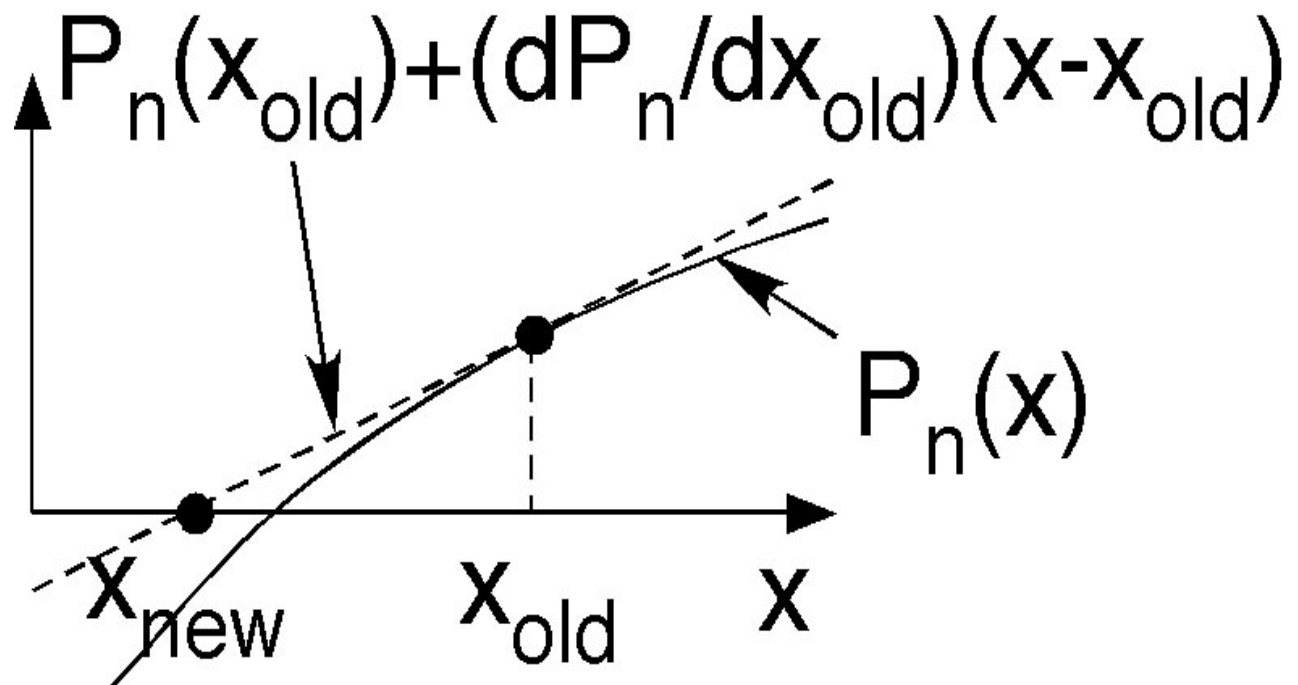
$$P_n(x) = 0$$

- **Newton iteration:** Successive linear approximation of the function

1. Start from an initial guess, x_0 , of the root
2. Given the k -th guess, x_k , obtain a refined guess, x_{k+1} , from the linear fit:

$$P_n(x) \cong P'_n(x_k)(x - x_k) + P_n(x_k) = 0$$

$$\rightarrow x_{k+1} = x_k - \frac{P_n(x_k)}{P'_n(x_k)}$$



Gauss-Legendre Program

- Given the lower & upper limits (x_1 & x_2) of integration & n , returns the abscissas & weights of the Gauss-Legendre n -point quadrature in $x[1:n]$ & $w[1:n]$.

```
void gauleg(float x1,float x2,float x[],float w[],int n) {
    int m,j,i;
    double z1,z,xm,xl,pp,p3,p2,p1;
    m=(n+1)/2; // Find only half the roots because of symmetry
    xm=0.5*(x2+x1);
    xl=0.5*(x2-x1);
    for (i=1;i<=m;i++) {
        z=cos(3.141592654*(i-0.25)/(n+0.5));
        do {
            p1=1.0; p2=0.0;
            for (j=1;j<=n;j++) { // Recurrence relation
                p3=p2; p2=p1;
                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
            }
            pp=n*(z*p1-p2)/(z*z-1.0); // Derivative
            z1=z;
            z=z1-p1/pp; // Newton's method
        } while (fabs(z-z1) > EPS); // EPS=3.0e-11
        x[i]=xm-xl*z;
        x[n+1-i]=xm+xl*z;
        w[i]=2.0*xl/((1.0-z*z)*pp*pp);
        w[n+1-i]=w[i];
    }
}
```

$$\begin{cases} P_{-1} = 0 \\ P_0 = 1 \\ jP_j = (2j-1)zP_{j-1} - (j-1)P_{j-2} \end{cases}$$

$$(z^2 - 1)P'_j = jzP_j - jP_{j-1}$$

$$z \leftarrow z - \frac{P_n(z)}{P'_n(z)}$$

$$w_i = \frac{2}{(1 - x_i^2)[P'_n(x_i)]^2}$$

How to Use the Gauss-Legendre Program

\$ cc -o gauleg-driver gauleg-driver.c gauleg.c -lm

```
//gauleg-driver.c
#include <stdio.h>
#include <math.h>

double *dvector(int, int);
void gauleg(double, double, double *, double *, int);

int main() {
    double *x,*w;
    double x1= -1.0,x2=1.0,sum;
    int N,i;
    printf("Input the number of quadrature points\n");
    scanf("%d",&N);
    x = dvector(1,N); // Allocate & use array elements x[1], ..., x[N]
    w = dvector(1,N); // It's Numerical Recipe's utility function (in gauleg.c)
    gauleg(x1,x2,x,w,N);
    sum=0.0;
    for (i=1; i<=N; i++)
        sum += w[i]*2.0/(1.0 + x[i]*x[i]);
    printf("Integration = %f\n", sum);
}
```

$$\pi = \int_{-1}^1 dx \frac{2}{x^2 + 1} \cong \sum_{k=1}^N w_k \frac{1}{x_k^2 + 1}$$

Recursive Function Evaluation

- Legendre function

```
p1=1.0; p2=0.0;
for (j=1;j<=n;j++) { // Recurrence relation
    p3=p2;
    p2=p1;
    p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
}
pp=n*(z*p1-p2)/(z*z-1.0); // Derivative
```

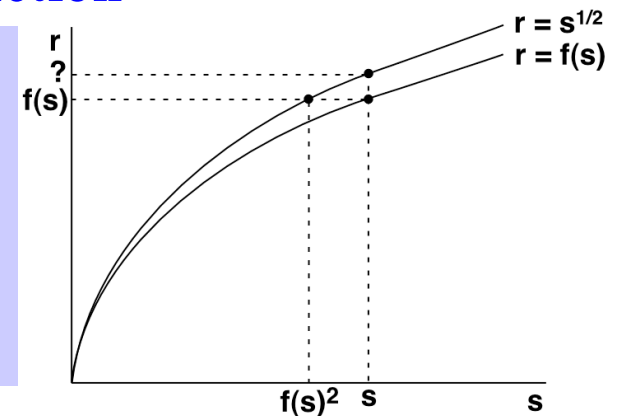
$$\begin{cases} P_{-1} = 0 \\ P_0 = 1 \\ jP_j = (2j-1)zP_{j-1} - (j-1)P_{j-2} \end{cases}$$

$$(z^2 - 1)P'_j = jzP_j - jP_{j-1}$$

- Compare it with a (low-accuracy) square-root function

```
#define C0 0.188030699
#define C1 1.48359853
#define C2 (-1.0979059)
#define C3 0.430357353

fs = C0+x*(C1+x*(C2+x*C3)); // Polynomial approximation
sr = fs+0.5*(x/fs-fs); // Newton correction
```



$$r - f(s) \approx \frac{dr}{ds} (s - f(s)^2)$$

$$\frac{dr}{ds} = \frac{d}{ds} s^{1/2} = \frac{1}{2} s^{-1/2} \approx \frac{1}{2f(s)}$$

$$\therefore r - f(s) = \frac{1}{2f(s)} (s - f(s)^2) = \frac{1}{2} \left(\frac{s}{f(s)} - f(s)^2 \right)$$

Where to Go from Here?

- **Gaussian quadrature for multiscale simulations?** *cf.* quasicontinuum method, where each function evaluation is an expensive quantum-mechanical calculation
cf. Knap & Ortiz, *J. Mech. Phys. Solids* **49**, 1899 (2001)
- **Adaptive Gaussian quadrature?** *cf.* power of Metropolis importance sampling: $2 \times 10^6 \ll 2^{400} \sim 10^{120}$ configurations
Lepage, *J. Comput. Phys.* **27**, 192 (1978)
Evila *et al.*, *IEEE T. Signal Process.* **69**, 474 (2021)
- **Related technique: Bayesian optimization (active learning, kriging), using Gaussian process regression with minimal number of function evaluations (trade-off between *exploration* & *exploitation*)**

Bassman *et al.*, *npj Comput. Mater.* **4**, 74 (2018)

Shields *et al.*, *Nature* **590**, 89 (2021)

