Lecture 16 Review

Typical fractal properties.

Meaning of “fractional” dimension, with examples.

Affine transformations and fractal examples.

Lab Exercise: Grow a Tree

Iteratively (but randomly) select the following affine transformations

$$(x_{n+1}, y_{n+1}) = \begin{cases} (0.05x_n, 0.6y_n) & 10\% \text{ probability} \\ (0.05x_n, -0.5y_n + 1.0) & 10\% \text{ probability} \\ (0.46x_n - 0.15y_n, 0.39x_n + 0.38y_n + 0.6) & 20\% \text{ probability} \\ (0.47x_n - 0.15y_n, 0.17x_n + 0.42y_n + 1.1) & 20\% \text{ probability} \\ (0.43x_n + 0.28y_n, -0.25x_n + 0.45y_n + 1.0) & 20\% \text{ probability} \\ (0.42x_n + 0.26y_n, -0.35x_n + 0.31y_n + 0.7) & 20\% \text{ probability} \end{cases}$$

Use `fern.cc` as a guide.

You can "plant" the tree at $(x_0, y_0) = (0.5., 0.0)$

Root finding

Often need to find roots (zeroes) to equations, i.e., solve f(x) =0
Consider case of only 1 independent variable.

Two general techniques: Newton-Raphson and Bisection.

$f(x^*) = 0$ (x* is a true root.)

Taylor series expand f(x) around x*

$$x^* = x_0 + \delta x$$

$$f(x^*) = f(x_0) + \delta x f'(x_0) + \tfrac{1}{2}(\delta x)^2 f''(x_0) + \cdots$$

$$0 \simeq f(x_0) + \delta x f'(x_0)$$

$$\Rightarrow \delta x \simeq -f(x_0)/f'(x_0)$$

$$x' = x_0 - f(x_0)/f'(x_0)$$

x_0 is a guess for the root

x' is an improved estimate.

Root Finding via Newton-Raphson Technique

$$\Rightarrow \delta x \simeq -f(x_0)/f'(x_0)$$

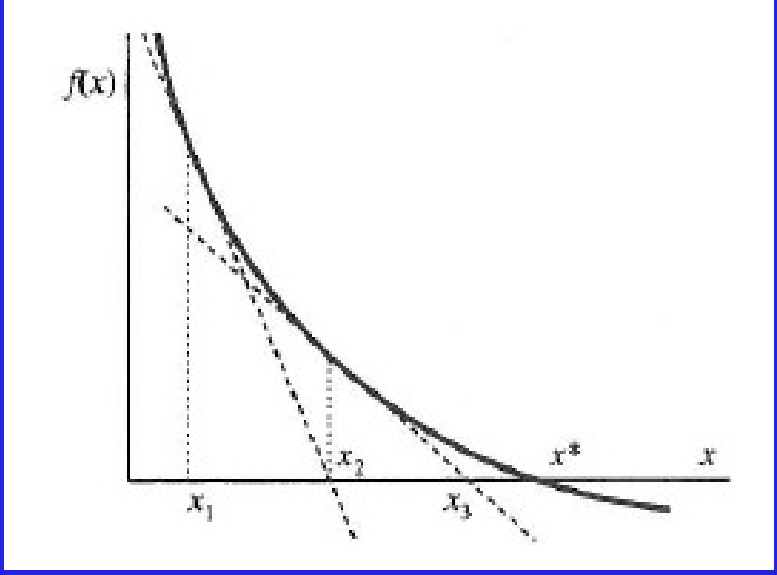
$$x' = x_0 - f(x_0)/f'(x_0)$$

x_0 is a guess for the root

x' is an improved estimate.

Process can be repeated to yield yet a better estimate for x*:

$$x_{N+1} = x_N - f(x_N)/f'(x_N)$$



When do we quit? $|f(x_k)| \leq \epsilon$

You pick ε, i.e., ε = 0.001

Needed to start: f(x), f'(x), x_0.

See `newton.cc`

newton.cc

```
/*
newton-raphson mthod for root finding.

*/

#include <iostream>
#include <cstdlib>
#include <cmath>

using std::endl;
using std::cout;

// define function:

double func_y(double x){
  double y = pow(x,3.0) + 1.0;
  return y;
}


//define first derivative of function:

double func_dydx(double s){
  return 3.0*pow(s,2.0);
}
```

```
int main()
{
   int N= 0;
   int N_limit = 1000;          // max number of trys
   double s = -1.1;             // initial guess for root
   double tol = 1.0e-5;         // tolerance for finding zero

   while ((fabs(func_y(s)) >= tol) && (N < N_limit)){
     s = s - func_y(s)/func_dydx(s);
     ++N;
        }

   if(N > N_limit){
     cout << "iteration limit exceeded" << endl;
     return 0;
   }

   if (N <= N_limit){
     cout << "estimated root = " << s << " with " << N << "
iterations." << endl;
   }

   return 0;
}
```

Root Finding via Bisection Algorithm

N-R technique sometimes fails (e.g., when f '(x_0) = 0)

The "bisection" algorithm is a robust alternative for root finding.

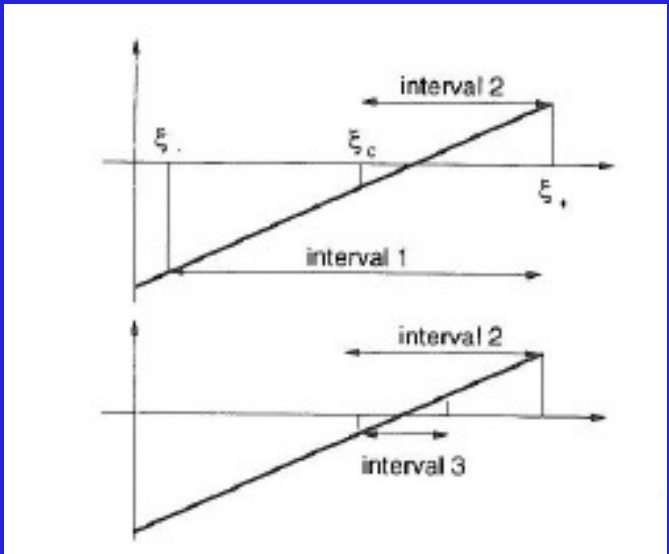
Find root to f(x), x ∈ [a,b].

$$f(x_L)f(x_R) < 0$$

$$x_C = \tfrac{1}{2}(x_L + x_R)$$ guess for root

If $f(x_C)f(x_R) < 0$ then $x_L = x_C$

otherwise $x_R = x_C$



Needed to start: f(x), [a,b]

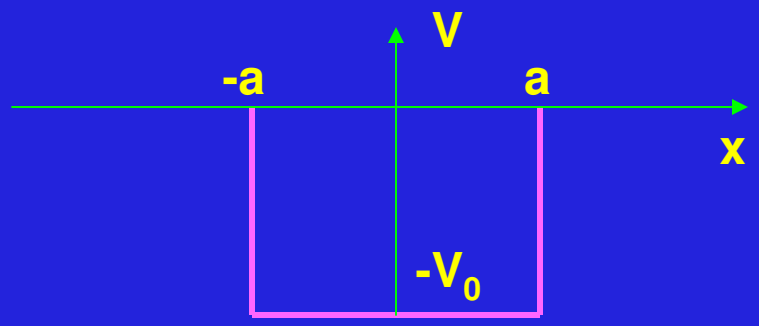
Quit: $|x_L - x_R| \leq \epsilon$

See `bisection.cc`

bisection.cc

```
#include <iostream>
#include <cstdlib>
#include <cmath>

using std::endl; using std::cout;

double func_f(double x){  //define function
  return pow(x,3.0) + 1.0;
}

int main()
{
   int N= 0;
   int N_limit = 1000;          // max number of trys
   double a = -100.0;   // lower bound for [a,b]
   double b = 3.0;      // upper bound for [a,b]
   double tol = 1.0e-5;    // tolerance for finding zero

if (func_f(a) == 0.0) {
     cout << a << " is a root." << endl;
     return 0;
   }
if (func_f(b) == 0.0) {
     cout << b << " is a root." << endl;
     return 0;
   }
 if (func_f(a)*func_f(b) > 0.0){
    cout << "[" << a << "," << b << "] do not bracket a root." << endl;
    return 0;
  }

   double x_low = a; double x_high = b; double x_mid;

   while ((fabs(x_high - x_low) >= tol) && (N < N_limit)){
     x_mid = 0.5*(x_low + x_high);
     if ( func_f(x_mid)*func_f(x_high) < 0){
       x_low = x_mid;
     }
     else {
       x_high = x_mid;
     }
     ++N;
   }

if (N <= N_limit){
     cout << "estimated root = " << x_mid << " with " << N << "
iterations." << endl;
   }

   return 0;
}
```

Particle in a Quantum Box



$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + V(x)\psi(x) = E\psi(x)$$

$$\mathcal{P} = |\psi(x)|^2 dx \qquad \int_{-\infty}^{\infty} dx\,|\psi(x)|^2 = 1$$

$$V(x) = \begin{cases} -V_0 = -83\,\text{MeV}, & \text{for } |x| \le a = 2\,\text{fm} \\ 0, & \text{for } |x| > a = 2\,\text{fm} \end{cases}$$

$$\frac{d^2\psi(x)}{dx^2} + \frac{2m}{\hbar^2}(E + V_0)\psi(x) = 0 \qquad \text{for } |x| \le a$$

$$\frac{d^2\psi(x)}{dx^2} + \frac{2m}{\hbar^2}E\psi(x) = 0 \qquad \text{for } |x| > a$$

$$\frac{2m}{\hbar^2} = \frac{2mc^2}{(\hbar c)^2} = \frac{2\times 940\,\text{MeV}}{(197\,\text{Mev}-\text{fm})^2} = 0.483\,\text{MeV}^{-1}\text{fm}^{-2}$$

Particle in a Quantum Box

$$\psi(x) = \begin{cases} Ce^{\beta x}, & \text{for } -\infty < x < -a \\ B\cos\alpha x, & \text{for } -a < x < a \\ Ce^{-\beta x}, & \text{for } a < x < +\infty \end{cases}$$

$$\alpha = \sqrt{\frac{2m(E+V_0)}{\hbar^2}} = \sqrt{0.483(E+83)}$$

$$\beta = \sqrt{\frac{-2mE}{\hbar^2}} = \sqrt{-0.483E}$$

$B\cos\alpha a = Ce^{-\beta a}$ ψ continuity

$-\alpha B\sin\alpha a = -\beta Ce^{-\beta a}$ ψ' continuity

$\Rightarrow$ $\alpha a\tan\alpha a - \beta a = 0$

$$\sqrt{2m(E+V_0)}\tan\sqrt{\frac{2m(E+V_0)}{\hbar^2}}\,a - \sqrt{-2mE} = 0$$

$\xi\tan\xi - \eta = 0$ with $\xi = \alpha a$ $\eta = \beta a$

$$\xi^2 + \eta^2 = \frac{2mV_0a^2}{\hbar^2} = 16.08$$

$f(\xi) = \xi\tan\xi - \eta = 0$ ←

Summary

Tree hugging (fractally speaking).

Root finding: Newton-Raphson & bisection algorithms.

Application of root finding to particle in a quantum box

Don’t suffer in silence. Scream for help!!!