Lecture 14 Review

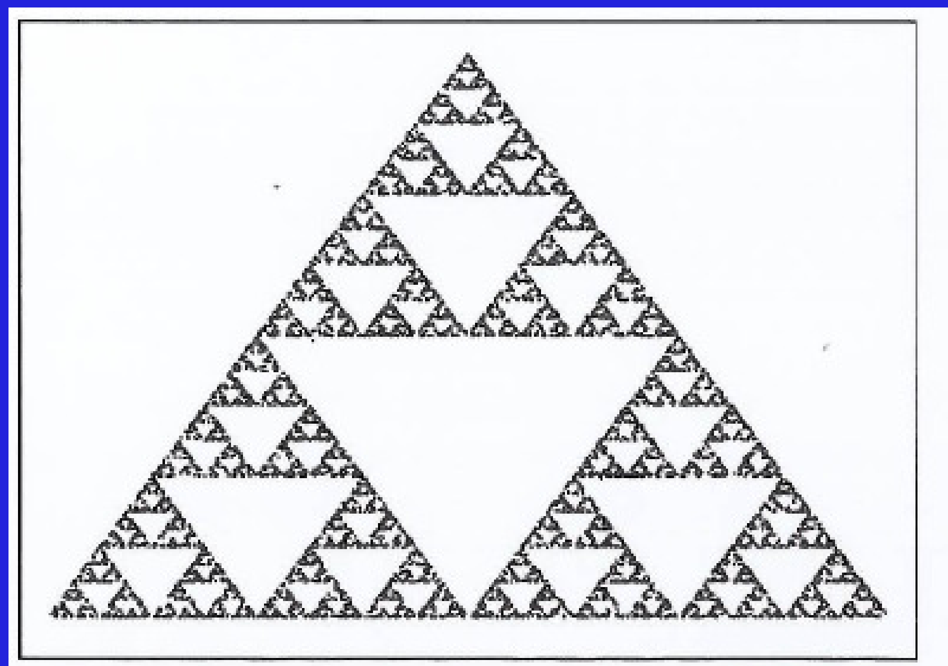
Chaos ID and Lyapunov exponent

Introduction to fractals (examples and simple properties)

Dimension of Sierpinksi Gasket

Q: What is dimension of Sierpinski's gasket?

$$d = \frac{\ln m}{\ln r}$$



r = ?

m = ?

d = ?

Affine Transformations

Self-similar fractals are generated from *affine transformations:*

A mapping of one set of points into another using a linear transformation + translation.

$$x' = Ax + b$$

Translation.

Scaling, shearing or rotation.

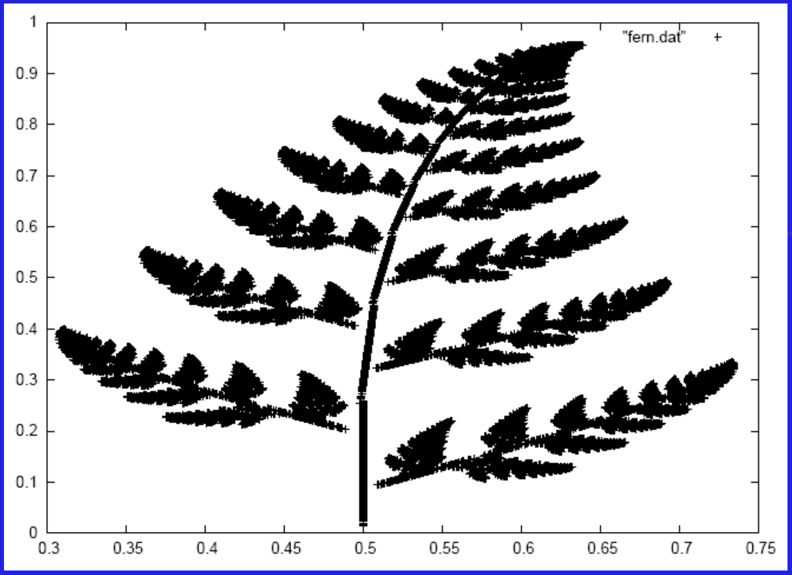
S'ski gasket: $$\left(x_{k+1}, y_{k+1}\right) = \frac{(x_k, y_k) + (Vx_n, Vy_n)}{2}$$ n = integer(1+3r_i)

Scaling

Translation

Another (Iterative) Affine Transformation

Barnsley's fern is a fractal. See `fern.cc`



random number

$$(x,y)_{n+1} = \begin{cases} (0.5, 0.27y_n) & r < 0.2 \\ (-0.139x_n + 0.263y_n + 0.57, \\ 0.246x_n + 0.224y_n - 0.036) & 0.02 \le r \le 0.17 \\ (0.17x_n - 0.215y_n + 0.408, \\ 0.222x_n + 0.176y_n + 0.0893) & 0.17 < r \le 0.3 \\ (0.781x_n + 0.034y_n + 0.1075, \\ -0.032x_n + 0.739y_n + 0.27) & 0.3 < r < 1.0 \end{cases}$$

Lab Exercise: Grow a Tree

Iteratively (but randomly) select the following affine transformations

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

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.

$\boxed{f(x^*) = 0}$ (x* is a true root.)

Taylor series expand f(x) around x*

$$\boxed{x^* = x_0 + \delta x}$$

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

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

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

$$\boxed{x' = x_0 - f(x_0)/f'(x_0)}$$

x_0 is a <u>guess</u> for the root

x' is an improved estimate.

Root Finding via Newton-Raphson Technique

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

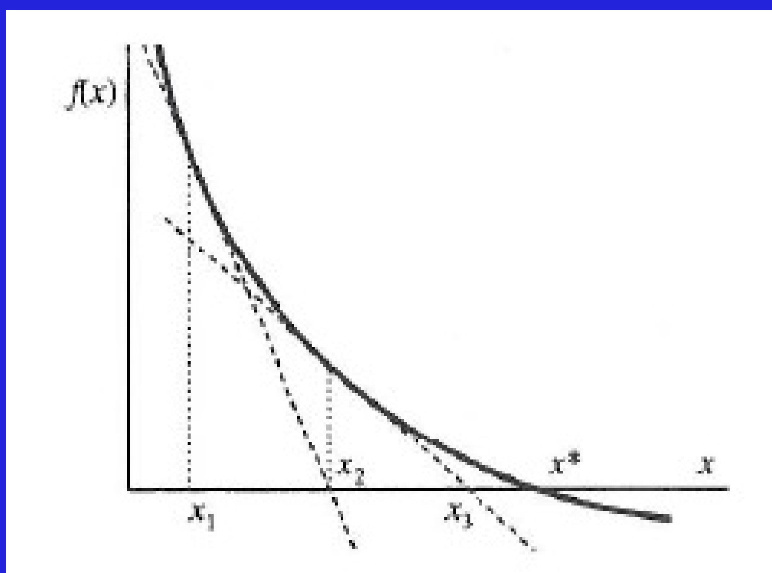
$$x' = x_0 - f(x_0)/f'(x_0)$$

x_0 is a <u>guess</u> 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? $|f(x_k)| \leq \epsilon$

<u>You</u> pick ε, i.e., ε = 0.001

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

See `newton.cc`

newton.cc

```
/*
newton-raphson method 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.

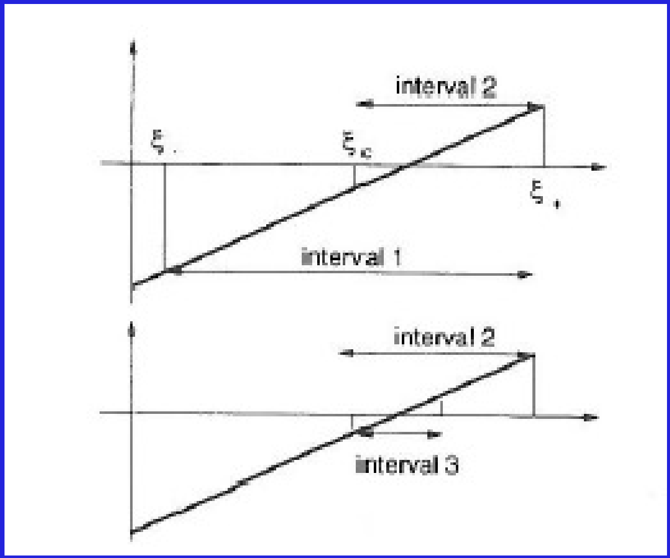
Find root to f(x) = 0, x $\in$ [a,b]. Start w/ "bracket values x_L and x_R.

$f(x_L)f(x_R) < 0$ (we'll code this slightly differently below.)

$x_C = \frac{1}{2}(x_L + x_R)$ 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: $|x_L - x_R| \leq \epsilon$

Number of steps k = $\log_2(\frac{b-a}{\epsilon})$

Bisection technique is sure but slow.

See `bisection.cc`

bisection.cc

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <gsl/gsl_math.h>

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 (GSL_SIGN(func_f(a)) * GSL_SIGN(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 ( GSL_SIGN(func_f(x_mid)) * GSL_SIGN(func_f(x_high)) < 0){
       x_low = x_mid;
     }
     else {
       x_high = x_mid;
     }
     ++N;
   }

   if(N > N_limit)
     cout << "iteration limit exceeded" << endl;
   else
     cout << "estimated root = " << x_mid << " with " << N << "
iterations." << endl;

   return 0;
}
```

bisection.cc example

Lab example: $f(x) = x^2 - 4\sin(x) = 0$

a= 1, b =3

$\varepsilon = 10^{-6}$

Print: a f(a) b f(b) for each step .

Q: How many steps k are required?

Compare w/ prediction on previous slides. $k = \log_2(\frac{b-a}{\epsilon})$

Summary

Dimension of Sierpinski's gasket fractal.

Affine transformations and fractal examples.

Root finding (N-R & bisection techniques).

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