Recursion

Instructor: Pei-Chiang Shao (邵培強)

Department of Mathematics, National Central University
Jhongli District, Taoyuan City 32001, Taiwan (R.O.C.)

E-mail: shaopj823@gmail.com

Recursion

We can use recursion to solve a problem if the original problem can be reduced to some smaller same problems. For example,

$$f(n) = n! = 1 \times 2 \times 3 \times \cdots \times n$$

can be defined recursively by

$$\begin{aligned} 0! &= 1 \\ f(n) &= n \times f(n-1), \text{ for } n = 1, 2, \ldots \end{aligned}$$

where $f(n-1)$ can be viewed as the same problem as $f(n)$ but with smaller size.

Recursion for $n!$

The following recursive matlab function compute $n!$ using recursion.

```
% matlab code
[Factorial(0); Factorial(5); Factorial(10)]
[Factorial(6.2); Factorial(-5)]

function f = Factorial(n)
    if mod(n,1)~=0 || n<0
        f = NaN;
        return
    end
    if n == 0
        f = 1;
    else
        f = n*Factorial(n-1);
    end
end
```

Inefficiency of recursion

However, the implementation of $n!$ by recursion seems easy but is not efficient since it can be programmed by using simple iteration, which is more computationally efficient.

```
% matlab code
[Factorial2(0); Factorial2(5); Factorial2(10)]
[Factorial2(6.2); Factorial2(-5)]

function f = Factorial2(n)
    if mod(n,1)~=0 || n<0
        f = NaN;
        return
    end
    f = 1;
    for i = 1 : n
        f = i*f;
    end
end
```

Determinant of square matrices

The determinant of a square matrix A can be computed recursively using the following expansion

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det(M_{ij}),$$

where M_{ij} is the $(n-1) \times (n-1)$ submatrix obtained by deleting row i and column j of the matrix A.

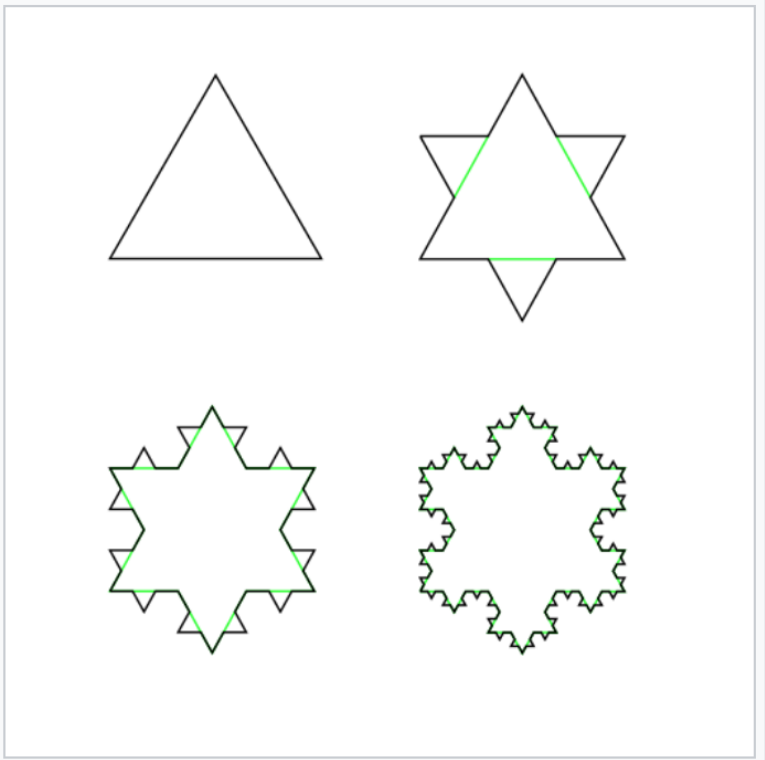
The matlab code is listed in the next slide.

Determinant of square matrices

```
% matlab code
A = [5 2 3; 9 1 6; 4 8 1];
Det(A)
function d = Det(A)
    n = length(A);
    if n == 1
        d = A(1,1);
    else
        d = 0;
        sgn = 1;
        for j = 1 : n
            M_1j = [A(2:n,1:j-1) A(2:n,j+1:n)];
            d = d + sgn*A(1,j)*Det(M_1j);
            sgn = -sgn;
        end
    end
end
```

Koch snowflake

Koch snowflake is a mathematical fractal curve. The progression for the area of the snowflake converges to $\frac{8}{5}$ times the area of the original triangle, while the progression for the perimeter diverges to infinity.



The first four iterations of the Koch snowflake

Construction of Koch snowflake

The Koch snowflake can be constructed by starting with an equilateral triangle, then recursively altering each line segment as follows:

Step 1 : divide the line segment into three segments of equal length.

Step 2 : draw an equilateral triangle that has the middle segment from Step 1 as its base and points outward.

Step 3 : remove the line segment that is the base of the triangle from Step 2.

Implementation for Koch snowflake

```
p = [0 0]; % starting position
q = [1 0]; % terminal position
n = 2;     % the number of progression
koch(p,q,n)
axis square; axis off;
function koch(p,q,n)
    if n == 0, plot([p(1),q(1)],[p(2),q(2)]); hold on;
    else
        d = q - p;
        a = p + d/3;
        b = q - d/3;
        c = (p+q)/2 + [-d(2) d(1)]/(2*sqrt(3));
        koch(p,a,n-1); koch(a,c,n-1);
        koch(c,b,n-1); koch(b,q,n-1);
    end
end
```

Homework

Use koch(p,q,n) to run 2 progressions to obtain the following results.