Sageable Mathematics

June 4, 2019

Sageable Mathematics is a project that introduces SageMath (or Sage) to students and encourages them to program in Sage. Each student (or each small group of students) will pick a mathematics concept and implement the computation in Sage. The goal is to experience programming through Sage and also gain a deeper insight into the concept.

This document is a gallery of students' work in Math316: Practice of Applied Mathematics at National Sun Yat-sen University, in 2019 Spring. The works, by the authors in each section, in this document are licensed under a Creative Commons Attribution 4.0 International License.

Contents

Determinant

Overview

Give an $n \times n$ matrix,
we want to calculate the **determinant** of the matrix.

Algorithm

1. Given an $n \times n$ matrix $\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}$
2. Do row operations to change the matrix into an upper triangular matrix.
3. And times all diagonal element then we get the determinant of the matrix.

Explanation

The goal is to calculate the determinant of the $n \times n$ matrix.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}$$

First we add $-\frac{a_{i,1}}{a_{1,1}} \times \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \end{bmatrix}$ to the i row for $i = 2, 3, \cdots, n-1, n$. Now we have

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & a_{2,2} - \frac{a_{2,1}}{a_{1,1}} \times a_{1,2} & \cdots & a_{2,n} - \frac{a_{2,1}}{a_{1,1}} \times a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n,2} - \frac{a_{n,1}}{a_{1,1}} \times a_{1,2} & \cdots & a_{n,n} - \frac{a_{n,1}}{a_{1,1}} \times a_{1,n} \end{bmatrix}$$

將該矩陣去除第一行及第一列後存成一個新方陣，使用迴圈做相同運算。

Example for 3*3 matrix

In [1]:

```
A = matrix(QQ, [
        [1,7,5],
        [5,6,7],
        [5,0,6]
    ])
A[1,:] +=  -A[1,0]/A[0,0] * A[0,:]
A[2,:] +=  -A[2,0]/A[0,0] * A[0,:]
b = A[0,0]
B = A[1:3,1:3]
B[1,:] +=  -B[1,0]/B[0,0] * B[0,:]
b = b*B[0,0]
C = B[1:2,1:2]
b = b*C[0,0]
print("the determinant of M is")
show(b)
```

```
the determinant of M is
```

$$-79$$

Here you can try my determinant calculator

In [2]:

```
##Enter a n*n matrix M
M = matrix(QQ, [
        [1,7,5],
        [5,6,7],
        [5,0,6]
    ])

#Below are my codes.
m,n = M.dimensions()
a = 1
c = 0
for i in range(1,m):
    if M[0,0]==0:
        c = c+1
        K=M[0,:]
        M[0,:]=M[i,:]
        M[i,:]=K;
    if M[0,0]!=0:
        break;
if M[0,0]==0:
    a=0;
if M[0,0]!=0:
    for i in range(m):
        if m-i>1:
            a = a*M[0,0];
            for j in range(m-i-1):
                if j==m-i-2:
                    M[j+1,:] +=  -M[j+1,0]/M[0,0] * M[0,:]
                    M = M[1:m-i,1:m-i];
                if j!=m-i-2:
                    M[j+1,:] +=  -M[j+1,0]/M[0,0] * M[0,:];
        if m-i<=1:
            a = a*M[0,0];
print("the determinant of M is")
if a==0:
    show(a)
if a!=0:
    if c%2!=0:
        show(-a)
    if c%2==0:
        show(a)
```

```
the determinant of M is
```

$$-79$$

In [3]:

```
### Jephian:  Let's make it a function for convenience.

def matrix_determinant(M):
    #Below is your code.
    m,n = M.dimensions()
    a = 1
    c = 0
    for i in range(1,m):
        if M[0,0]==0:
            c = c+1
            K=M[0,:]
            M[0,:]=M[i,:]
            M[i,:]=K;
        if M[0,0]!=0:
            break;
    if M[0,0]==0:
        a=0;
    if M[0,0]!=0:
        for i in range(m):
            if m-i>1:
                a = a*M[0,0];
                for j in range(m-i-1):
                    if j==m-i-2:
                        M[j+1,:] +=  -M[j+1,0]/M[0,0] * M[0,:]
                        M = M[1:m-i,1:m-i];
                    if j!=m-i-2:
                        M[j+1,:] +=  -M[j+1,0]/M[0,0] * M[0,:];
            if m-i<=1:
                a = a*M[0,0];
    print("the determinant of M is")
    if a==0:
        show(a)
    if a!=0:
        if c%2!=0:
            show(-a)
        if c%2==0:
            show(a)
```

In [4]:

```
M = matrix(QQ, [
    [0,2,3],
    [3,5,6],
    [0,8,9]
])

matrix_determinant(M)

print 'sage answer:', M.determinant()
```

```
the determinant of M is
```

18

```
sage answer: -18
```

In [0]:

```
```

Good Gaussian elimination

Overview

Given a matrix with finite rows and finite columns, Good Gaussion Elimination aims to reduce a matrix into its upper triangular form within as less time as it could.

Algorithm

1. Create an $N \times M$ matrix that you what to reduce.

$$A = \begin{bmatrix} X_{1,1} & \cdots & X_{1,M} \\ \vdots & \vdots & \vdots \\ X_{N,1} & \cdots & X_{N,M} \end{bmatrix}$$

2. Then compute a upper triangular matrix

$$A' = \begin{bmatrix} Y_{1,1} & \cdots & Y_{1,M} \\ \vdots & \vdots & \vdots \\ 0 & \cdots & Y_{N,M} \end{bmatrix}$$

Implementation

In [1]:

```
##########
## MAIN ##
##########

##algorithm part
def gauss_eli(A):
    n,m=A.dimensions()
    c=0
    d=[]
    for k in range(0,m):
        t=0
        for i in range(0,n):
            if((A[i,k] != 0) and (t != 1) and (i not in d)):
                t=1
                d=d+[i]
                for j in range(0,n):
                    if(j != i):
                        A[j,0:] = A[j,0:]*A[i,k]-A[i,0:]*A[j,k]
    #answer
    #return A

    #rearrange
    AR = copy(A)
    for p in range(0,m):
        if(p not in d):
            d=d+[p]
    for l in range(0,n):
            AR[l,0:]=A[d[l],0:]
    return AR




##Trials
import numpy as np
a=matrix(np.random.rand(3,4)*10)
a=matrix([[3,3,3],[2,2,2],[4,4,4]])
a=matrix([[0,0,-2,1],[2,2,2,1],[-2,0,0,1]])
print "original matrix"
print a
print '---'
b=copy(a)
print "my answer"
print gauss_eli(a)
print "---"
print "sage default answer"
print b.echelon_form()
```

```
original matrix
[ 0  0 -2  1]
[ 2  2  2  1]
[-2  0  0  1]
---
my answer
[-128    0    0   64]
[   0  -64    0  -96]
[   0    0  -16    8]
---
```

```
sage default answer
[ 2  0  0 -1]
[ 0  2  0  3]
[ 0  0  2 -1]
```

In [2]:

```
A = matrix(QQ, [
    [0,0,1],
    [0,1,1],
    [1,1,0]
])
```

In [3]:

```
gauss_eli(A)
```

Out[3]:

```
[1 0 0]
[0 1 0]
[0 0 1]
```

In [4]:

```
A.echelon_form()
```

Out[4]:

```
[1 0 0]
[0 1 0]
[0 0 1]
```

In [ ]:

```

```

Gradient Descent

Overview

Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a function. Our main goal is to find a local minimum of a function f.

We use the method of **gradient descent** to find a local minimum of f.

Define a function f from $\mathbb{R}^2$ to $\mathbb{R}$.

For example, when $f(x, y) = x^2 + y^2$, the algorithm will return an approximation of the location of the minimum $(0, 0)$.

Algorithm

1. Let f be a function from $\mathbb{R}^2$ to $\mathbb{R}$ and choose $\mathbf{x}_0 = (x_0, y_0) = (0, 0)$ as an initial point.
2. Then compute $\nabla f(x_i, y_i)$, the gradient of f at $\mathbf{x}_i$, and

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \cdot \nabla f(x_i, y_i)$$

where α is a small enough number such that the sequence$(\mathbf{x}_n)$ converges to a local minimum.

1. Repeat Step 2 for several steps.

Explanation

The goal is to find a local minimum of a given function.

Given functions are restricted to be two variables and the gradient of function is needed to be calculated by hands.

We set $\alpha = 0.05$ and the step of x denoted by $\epsilon < 10^{-5}$.

The sequence might converge slowly or not converge, so we need to set the maximum iteration times.

Implimentation

In [1]:

```
import numpy as np

def gradient_descent(f, gradf, steps=100, alpha=0.5):
    """
    Input:
        f: a function of x,y that we want to evaluate the minimum
        gradf:gradient of the function f
        steps:the steps of iterations
        alpha:step size multiplier
    Output:
        min:the minimum of f
    """
    v1 = vector(np.random.randn(2)); ### next vector
    v2 = vector(np.random.randn(2)); ### current vector
    for i in range(steps):
        v2 = v1;
        v1 = v2 - alpha * vector(gradf(v2));
        h = v1[0] - v2[0];
        if abs(h) <= epsilon :
            break;
    print("Minimum at",v1," and minimum is",f(v1));
```

Example 1

$f(x, y) = x^2 + y^2$

In [2]:

```
epsilon=0.00001;

f = lambda v1: v1[0]^2 + v1[1]^2

gradf = lambda v2: (2*v2[0], 2*v2[1])

gradient_descent(f,gradf, steps=100, alpha=0.5)
```

```
('Minimum at', (0.0, 0.0), ' and minimum is', 0.0)
```

In [3]:

```
### If we use a large alpha,then the result may be different.

gradient_descent(f,gradf, steps=100, alpha=1)
```

```
('Minimum at', (1.236371488782072, -0.781210485847116), ' and minimum
is', 2.1389042814706842)
```

Example 2

$f(x, y) = x^2 - y^2 + xy - 4$

In [4]:

```
epsilon=0.00001;

f = lambda v1: v1[0]^2 - v1[1]^2 + v1[0]*v1[1] - 4

gradf = lambda v2: (2*v2[0] + v2[1], -2*v2[1]+v2[0])

gradient_descent(f,gradf, steps=100)
```

```
('Minimum at', (2.792893696625991e+31, -1.1830887552838372e+32), ' and
minimum is', -1.652120563590659e+64)
```

In [5]:

```
### example of vectors and functions in SageMath

### tuples are nice but their addition is not what we want
print "(0,0) + (1,0) =", (0,0) + (1,0)

### Use vector instead
v = vector([0,0])
u = vector([1,0])
print "u+v =", u+v

### get vector entries
print "u[0],u[1] =", u[0], u[1]

### a function that takes a vector as input
### syntax:
### lambda input: output
### this is called the lambda method to define a function
f = lambda v: v[0]^2 + v[1]^2
print "f(v) =", f(v)
print "f(u) =", f(u)

### alternatively, you have to do the classical function define
def g(v):
    return v[0]^2 + v[1]^2
### two methods are almost the same except that the lambda method you don't have to
print "g(v) =", g(v)
print "g(u) =", g(u)

### Therefore, the gradf can be
gradf = lambda v: 2*v[0] + 2*v[1]
```

```
(0,0) + (1,0) = (0, 0, 1, 0)
u+v = (1, 0)
u[0],u[1] = 1 0
f(v) = 0
f(u) = 1
g(v) = 0
g(u) = 1
```

In [ ]:

```

```

Graph Coloring

Overview

Given a simple graph G with a vertex set $V = \{v_1, v_2, \dots, v_n\}$ and a color set $[m] := \{1, \dots, m\}$,
an m**-coloring** of G is an assignment $c : V \to [m]$.

A **proper** m**-coloring** on a graph G is an m-coloring c such that
v_i and v_j have different colors if v_i and v_j are connected by an edge $\forall i, j \in [n]$.

The goal is to find the smallest m such that G has a proper m-coloring.
This minimum m is called the **chromatic number** of G, denoted as $\chi(G)$.

Algorithm (Greedy coloring algorithm)

1. Given a graph G and a vertex ordering $\{v_1, v_2, \dots, v_n\}$
2. For each i, color v_i with the smallest positive integer j such that
   j is not used in any of neighbors of v_i.
3. Output the number of colors used (j).

Explanation

Given a graph G with a vertex set V and an edge set E
Label the vertices as $V = \{0, 1, \dots, n-1\}$.
By the **Geedy Algorithm**, color the vertex 0 with color `c[1]` .
If vertex 1 is a neighbor of vertex 0, then color vertex 1 with `c[2]` .
If not, then color vertex 1 with `c[1]` .
But notice that the output of Greedy Algorithm might not be the minimum color set.

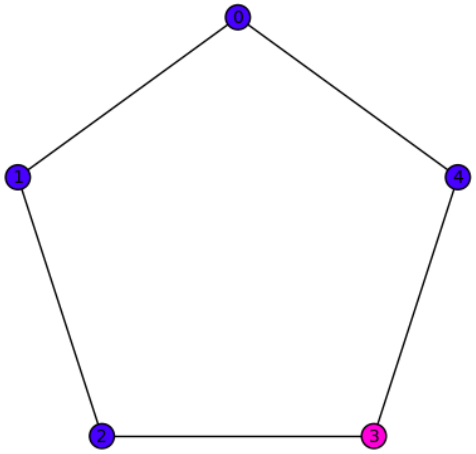
Implementation

In [1]:

```
def greedy_coloring(g, vertex_order=None, show_color=False):
    """
    Input:
        g: a simple graph
        vertex_order: a list of vertices of g;
                      if None, g.vertices() is taken.
        show_color: show the graph with the output color if True
    Output:
        a coloring
    """
    if vertex_order==None:
        vertex_order = g.vertices()
    n = g.order()

    coloring={}
    for v in vertex_order:
        color_in_nbr = []
        colored = coloring.keys()
        for u in g.neighbors(v):
            if u in colored:
                color_in_nbr.append(coloring[u])
        for color in range(n):
            coloring[v] = n
            if u in g.neighbors(v):
                coloring[u] = n + 1

    if show_color:
        num_colors = max(coloring.values()) + 1
        colors = rainbow(num_colors) ### colors is a list of num_colors colors
        ### the coloring few lines create a dictionary {color: [vertices]}
        c = {colors[i]: [] for i in range(num_colors)}
        for v in vertex_order:
            i = coloring[v]
            c[colors[i]].append(v)
        g.show(vertex_color=c)
    return coloring
```

In [4]:

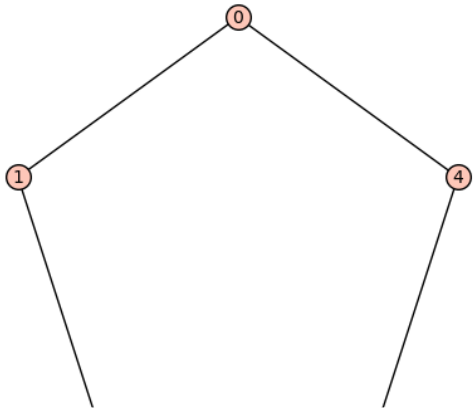
```
g = graphs.CycleGraph(5)
print greedy_coloring(g,show_color=True)
```




```
{0: 5, 1: 5, 2: 5, 3: 6, 4: 5}
```

In [5]:

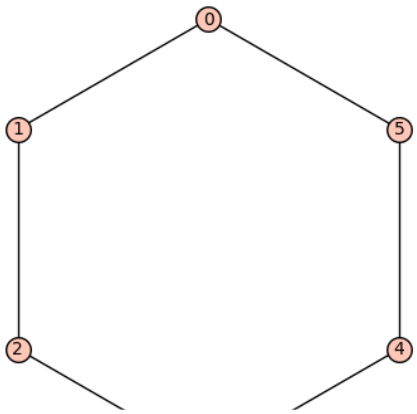
```
### some functions of graph

### graphs contains many built-in graphs
g = graphs.CycleGraph(5)

g.show()

print "vertices", g.vertices()
print "edges", g.edges()
print "neighbors of 0", g.neighbors(0)

### an example of coloring
### c = {vertex: color}
c = {0:0, 1:1, 2:0, 3:1, 4:2}

### call a color
print "the color for vertex 3 is", c[3]
```



Examples

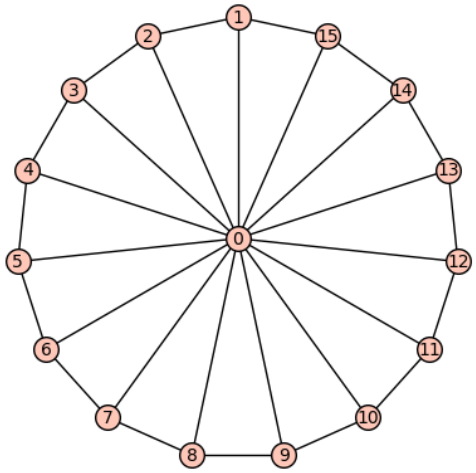
In [6]:

```
### Cycle graph
g = graphs.CycleGraph(6)
g.show()

print "vertices", g.vertices()
print "edges", g.edges()
print "neighbors of 0", g.neighbors(0)


def greedy_coloring(g, vertex_order=None, show_color=False):
    """
    Input:
        g: a simple graph
        vertex_order: a list of vertices of g;
                      if None, g.vertices() is taken.
        show_color: show the graph with the output color if True
    Output:
        a coloring
    """
    if vertex_order==None:
        vertex_order = g.vertices()
    coloring={}
    for v in vertex_order:
        coloring[0] = 0
        if v + 1 == g.neighbors(v):
            coloring[v + 1] = coloring[v] + 1    ### this part your need to find a proper j and assign coloring[v] = j
        if v + 1 != g.neighbors(v):
            coloring[v + 1] = coloring[v]

    if show_color:
        num_colors = max(coloring.values()) + 1
        colors = rainbow(num_colors) ### colors is a list of num_colors colors
        ### the coloring few lines create a dictionary {color: [vertices]}
        c = {colors[i]: [] for i in range(num_colors)}
        for v in vertex_order:
            i = coloring[v]
            c[colors[i]].append(v)
        g.show(vertex_color=c)
    return coloring

print greedy_coloring(g,show_color=True)
```

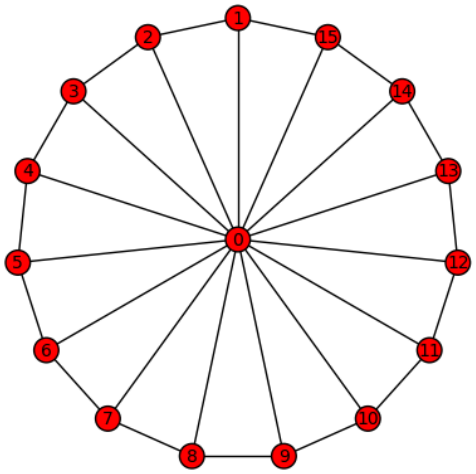


In [7]:

```
### Wheel graph
g = graphs.WheelGraph(16)
g.show()

print "vertices", g.vertices()
print "edges", g.edges()
print "neighbors of 0", g.neighbors(0)


def greedy_coloring(g, vertex_order=None, show_color=False):
    """
    Input:
        g: a simple graph
        vertex_order: a list of vertices of g;
                      if None, g.vertices() is taken.
        show_color: show the graph with the output color if True
    Output:
        a coloring
    """
    if vertex_order==None:
        vertex_order = g.vertices()
    coloring={}
    for v in vertex_order:
        coloring[0] = 0
        if v + 1 == g.neighbors(v):
            coloring[v + 1] = coloring[v] + 1    ### this part your need to find a proper j and assign coloring[v] = j
        if v + 1 != g.neighbors(v):
            coloring[v + 1] = coloring[v]

    if show_color:
        num_colors = max(coloring.values()) + 1
        colors = rainbow(num_colors) ### colors is a list of num_colors colors
        ### the coloring few lines create a dictionary {color: [vertices]}
        c = {colors[i]: [] for i in range(num_colors)}
        for v in vertex_order:
            i = coloring[v]
            c[colors[i]].append(v)
        g.show(vertex_color=c)
    return coloring

print greedy_coloring(g,show_color=True)
```



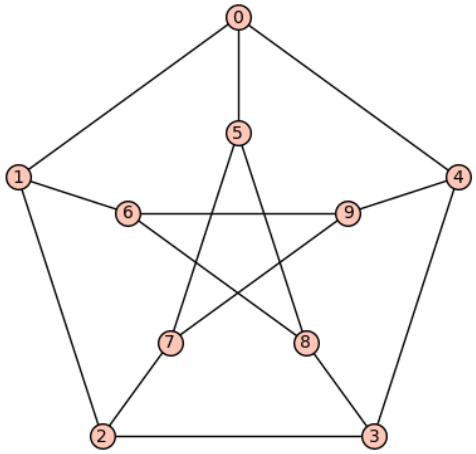

```
vertices [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
edges [(0, 1, None), (0, 2, None), (0, 3, None), (0, 4, None), (0, 5, None), (0, 6, None), (0, 7, None), (0, 8,
None), (0, 9, None), (0, 10, None), (0, 11, None), (0, 12, None), (0, 13, None), (0, 14, None), (0, 15, None),
(1, 2, None), (1, 15, None), (2, 3, None), (3, 4, None), (4, 5, None), (5, 6, None), (6, 7, None), (7, 8, None),
(8, 9, None), (9, 10, None), (10, 11, None), (11, 12, None), (12, 13, None), (13, 14, None), (14, 15, None)]
neighbors of 0 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```



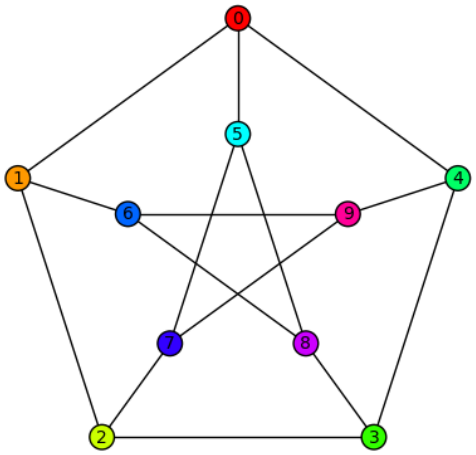

```
{0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0}
```

In [8]:

```
### Petersen graph
g = graphs.PetersenGraph()
g.show()

print "vertices", g.vertices()
print "edges", g.edges()
print "neighbors of 0", g.neighbors(0)


def greedy_coloring(g, vertex_order=None, show_color=False):
    """
    Input:
        g: a simple graph
        vertex_order: a list of vertices of g;
                      if None, g.vertices() is taken.
        show_color: show the graph with the output color if True
    Output:
        a coloring
    """
    if vertex_order==None:
        vertex_order = g.vertices()
    coloring={}
    for v in vertex_order:
        coloring[v] = v ### this part your need to find a proper j and assign coloring[v] = j

    if show_color:
        num_colors = max(coloring.values()) + 1
        colors = rainbow(num_colors) ### colors is a list of num_colors colors
        ### the coloring few lines create a dictionary {color: [vertices]}
        c = {colors[i]: [] for i in range(num_colors)}
        for v in vertex_order:
            i = coloring[v]
            c[colors[i]].append(v)
        g.show(vertex_color=c)
    return coloring

print greedy_coloring(g,show_color=True)
```




```
vertices [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
edges [(0, 1, None), (0, 4, None), (0, 5, None), (1, 2, None), (1, 6, None), (2, 3, None), (2, 7, None), (3, 4,
None), (3, 8, None), (4, 9, None), (5, 7, None), (5, 8, None), (6, 8, None), (6, 9, None), (7, 9, None)]
neighbors of 0 [1, 4, 5]
```



```
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9}
```

In [0]:

```

```

This work is done by 藍博瀚.

In [1]:

```
function [Np Nn Nz] = sylv_inertia(A)
% compute the Sylvester's inertia of a symmetric matrix
% Np: number of positive eigenvalues of the input matrix
% Nn: number of negative eigenvalues of the input matrix
% Nz: number of zero eigenvalues of the input matrix

[V D] = eig(A) % D is a diagonal matrix composed of the eigenvalues of A
                % V is a matrix composed of the eigenvectors of A
d = diag(D); % d is a vector composed of all eigenvalues of A
Np = sum(d > 0); % Np is the number of positive eigenvalues of A
Nn = sum(d < 0); % Nn is the number of negative eigenvalues of A
Nz = sum(d == 0); % Nz is the number of zero eigenvalues of A
end

% the main program for calling the sylv_inertia function

% clear all; % clear all existing variables
% close all; % close all graph windows

% A is the input symmetric matrix
A = [2 5 8 -7; 5 6 9 12; 8 9 -4 18; -7 12 18 -30]; % example 1
%A = [3 6 8 -7; 6 10 9 12; 8 9 -4 18; -7 12 18 45]; % example 2
%A = [14 -2 28 -7; -2 10 9 12; 28 9 -9 18; -7 12 18 2]; % example 3
disp('The input matrix is ');
disp(A);
[Np Nn Nz] = sylv_inertia(A); % call the sylv_inertia function
disp('sylv_inertia(A)');
disp(['     Np    Nn    Nz']);
disp([Np, Nn, Nz]);
```

```
The input matrix is
    2    5    8   -7
    5    6    9   12
    8    9   -4   18
   -7   12   18  -30
V =

   0.227952  -0.076146  -0.922865   0.300932
  -0.160742  -0.661920   0.240412   0.691539
  -0.413408   0.707753   0.026099   0.572274
   0.866772   0.234837   0.299736   0.322050

D =

Diagonal Matrix

  -42.6514         0         0         0
         0   -7.3054         0         0
         0         0    2.7447         0
         0         0         0   21.2121

sylv_inertia(A)
     Np    Nn    Nz
   2   2   0
```

Limit

Introduction

This program aims to show what is the "Limit".
In mathematics, a limit is the value that a function (or sequence) "approaches" as the input (or index) "approaches" some value.

1. Give a simple polynomial,ex:

$$\frac{x+1}{x^2+2x-3}$$

   We want to know what will happens when $x \to 3$.
2. It's easy to see that

$$\frac{x+1}{x^2+2x-3} \to \frac{1}{3}$$

   as $x \to 3$.
3. We can denote as

$$\lim_{x \to 3} \frac{x+1}{x^2+2x-3} = \frac{1}{3}$$

   which means the value of $\frac{x+1}{x^2+2x-3}$ will aprroach to $\frac{1}{3}$, as $x \to 3$.

Algorithm

To solve the problem by the following process:
We can start with giving a value of x.
For example: $x = 2$,which lead $\frac{x+1}{x^2+2x-3} = \frac{2}{5}$ Then start to approaching to $x = 3$ from $x = 2$ like below

$x = 2$
$\frac{x+1}{x^2+2x-3} = \frac{3}{5} = 0.6$
$x = 2.1$
$\frac{x+1}{x^2+2x-3} = \frac{3.1}{5.61} = 0.553$
.
.
.
$x = 2.8$
$\frac{x+1}{x^2+2x-3} = \frac{3.8}{10.44} = 0.364$
$x = 2.9$
$\frac{x+1}{x^2+2x-3} = \frac{3.9}{11.21} = 0.348$
$x = 3$
$\frac{x+1}{x^2+2x-3} = \frac{1}{3} = 0.333$

We can see that the value of $\frac{x+1}{x^2+2x-3}$ is gradually approach to $\frac{1}{3}$

Implimentation

First,need to check if the limit exists or not
Except some unique function like $f(x) = -1^x$

> If the function is fraction form like $\frac{x+5}{x-3}$,and input approach to some value a

We can check the function does or doesn't have limit when input approach to a by this way

> if (x-3)%(x-a)=0,limit doesn't exist
> if (x-3)%(x-a)≠0,limit exists

For example,we want to know,

> $f(x) = \frac{x^2-1}{x} \to ?$ if $x \to 2$

First,check limit does or doesn't exist

In [1]:

```
if (x)%(x-2)==0:
    print "limit doesn't exist"
if (x)%(x-2)!=0:
    print "limit exist"
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
 last)
<ipython-input-1-a814ec059bdb> in <module>()
----> 1 if (x)%(x-Integer(2))==Integer(0):
      2     print "limit doesn't exist"
      3 if (x)%(x-Integer(2))!=Integer(0):
      4     print "limit exist"

sage/structure/element.pyx in sage.structure.element.Element.__mod__
 (build/cythonized/sage/structure/element.c:13749)()

sage/structure/element.pyx in sage.structure.element.Element._mod_ (bu
ild/cythonized/sage/structure/element.c:14075)()

TypeError: unsupported operand parent(s) for %: 'Symbolic Ring' and 'S
ymbolic Ring'
```

So we know $lim_{x\to 2}\frac{x^2-1}{x}$ exists Then we can solve $lim_{x\to 2}\frac{x^2-1}{x}$

In [5]:

```
for i in range(11):
    a=1+0.1*i
    h=(a^2-1)/a
    print h
```

```
0.000000000000000
0.190909090909091
0.366666666666667
0.530769230769231
0.685714285714286
0.833333333333333
0.975000000000000
1.11176470588235
1.24444444444444
1.37368421052632
1.50000000000000
```

We can also use "limit_app" to solve $lim_{x\to 2}\frac{x^2-1}{x} =?$ since we know limit exists

In [6]:

```
def limit_app(f,a):
    """
    Input:
        f: a function (symbolic exporession in x)
        a: a real number
    Output:
        print the approximation of f(x) as a approaches to a
    """
    for i in range(1,11):
        t=(a-1)+0.1*i
        g=f.subs(x=t)
        print g
    # return g
```

In [7]:

```
limit_app((x^2-1)/(x),2)
```

```
0.190909090909091
0.366666666666667
0.530769230769231
0.685714285714285
0.833333333333333
0.975000000000000
1.11176470588235
1.24444444444444
1.37368421052632
1.50000000000000
```

Explanation

The goal is to solve what value of function will be approached when the value of x approach to 3

$$x \to 3$$

,

$$\frac{x+1}{x^2+2x-3} \to ?$$

We can easy to calculate that when value of x is closer and closer to 3
Value of function

$$\frac{x+1}{x^2+2x-3}$$

is approaching to $0.333 \fallingdotseq \frac{1}{3}$

Notice that not any funtion can approach to a unique value when input approach to some value
For example,

$f(x) = -1^x$
when $x = 10 \to f(x) = 1$
But it doesn't mean that $\lim_{x \to 10} f(x) = 1$

Let's see why,
We all know

> $f(10) = 1$

But

> $f(9) = f(11) = -1$
> $f(8) = f(12) = 1$
> .
> .
> .
> $f(2) = f(18) = 1$
> $f(1) = f(19) = -1$

The value of $f(x)$ isn't just approaching to 1 while the value of x approaching to 10
The value of $f(x)$ just switch between 1 and -1

Thus $\lim_{x\to 10} f(x)$ doesn't exist

In the end,we know that $lim_{x\to 2}\frac{x^2-1}{x}$ exists and $lim_{x\to 2}\frac{x^2-1}{x} = 1.5$

Examples

1. $lim_{x\to 5}\frac{x^2}{x+5} = ?$

In [8]:

```
###check
if (x+5)%(x-5)==0:
    print "limt doesnt exist"
if (x+5)%(x-5)!=0:
    print "limit exist"
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
 last)
<ipython-input-8-8930f8a2484e> in <module>()
      1 ###check
----> 2 if (x+Integer(5))%(x-Integer(5))==Integer(0):
      3     print "limt doesnt exist"
      4 if (x+Integer(5))%(x-Integer(5))!=Integer(0):
      5     print "limit exist"

sage/structure/element.pyx in sage.structure.element.Element.__mod__
 (build/cythonized/sage/structure/element.c:13749)()

sage/structure/element.pyx in sage.structure.element.Element._mod_ (bu
ild/cythonized/sage/structure/element.c:14075)()

TypeError: unsupported operand parent(s) for %: 'Symbolic Ring' and 'S
ymbolic Ring'
```

In [9]:

```
for i in range(11):
    x=3+0.2*i
    g=(x^2)/(x+5)
    print g
```

```
1.12500000000000
1.24878048780488
1.37619047619048
1.50697674418605
1.64090909090909
1.77777777777778
1.91739130434783
2.05957446808511
2.20416666666667
2.35102040816327
2.50000000000000
```

$lim_{x \to 5} \frac{x^2}{x+5} = 2.5$

In [10]:

```
limit_app((x^2)/(x+5),5)
```

```
2.50000000000000
2.50000000000000
2.50000000000000
2.50000000000000
2.50000000000000
2.50000000000000
2.50000000000000
2.50000000000000
2.50000000000000
2.50000000000000
```

2. $lim_{x\to 7}\frac{x^3-5x}{x-7} = ?$

In [11]:

```
###check
if (x-7)%(x-7)==0:
    print "limt doesnt exist"
if (x-7)%(x-7)!=0:
    print "limit exist"
```

```
limt doesnt exist
```

$lim_{x\to 7}\frac{x^3-5x}{x-7}$ doesn't exists

In [12]:

```
for i in range(11):
    x=5+0.2*i
    g=(x^3-5*x)/(x-7)
    print g
```

```
-50.0000000000000
-63.6711111111111
-81.5400000000000
-105.440000000000
-138.426666666667
-186.000000000000
-259.160000000000
-383.573333333334
-636.239999999999
-1402.16000000000
+infinity
```

In [13]:

```
limit_app((x^3-5*x)/(x-7),7)
```

```
+infinity
+infinity
+infinity
+infinity
+infinity
+infinity
+infinity
+infinity
+infinity
+infinity
```

$lim_{x\to 7}\frac{x^3-5x}{x-7}$ doesn't exists

3. $lim_{x\to 10} 3x^2 - 6x - 1 = ?$

In [14]:

```
limit_app(3*x^2-6*x-1,10)
```

```
104.000000000000
104.000000000000
104.000000000000
104.000000000000
104.000000000000
104.000000000000
104.000000000000
104.000000000000
104.000000000000
104.000000000000
```

In [15]:

```
def existence(h,k):
    """
    Input:
        h:a function
        k:a function
    Output:
        determine limit exist or not
    """
    H=h.polynomial(QQ)
    K=k.polynomial(QQ)
    if H.quo_rem(K)==(t.polinomial(QQ),0)
    print "limit doesn't exist"
    else
    print "limit exists"
```

```
  File "<ipython-input-15-1a100a382493>", line 11
    if H.quo_rem(K)==(t.polinomial(QQ),Integer(0))
                                                  ^
SyntaxError: invalid syntax
```

In [16]:

```
### now you can do division algorithm

### create factors in polynomial
factor1 = (x-1).polynomial(QQ)
factor2 = (x+1).polynomial(QQ)

g.quo_rem(factor1) ### you may try factor2
```

```
---------------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
 last)
<ipython-input-16-c3c66227d409> in <module>()
      2
      3 ### create factors in polynomial
----> 4 factor1 = (x-Integer(1)).polynomial(QQ)
      5 factor2 = (x+Integer(1)).polynomial(QQ)
      6

sage/structure/element.pyx in sage.structure.element.Element.__getattr
__ (build/cythonized/sage/structure/element.c:4303)()

sage/structure/element.pyx in sage.structure.element.Element.getattr_f
rom_category (build/cythonized/sage/structure/element.c:4412)()

sage/cpython/getattr.pyx in sage.cpython.getattr.getattr_from_other_cl
ass (build/cythonized/sage/cpython/getattr.c:1834)()

AttributeError: 'sage.rings.real_mpfr.RealNumber' object has no attrib
ute 'polynomial'
```

In [17]:

```
### functions related to polynomials

f = x^2 + 2*x - 3

### originally f is a symbolic expression
print "type(f)", type(f)

g = f.polynomial(QQ)

### put f as a polynomial with QQ (rational numbers) as coefficients
print "type(g)", type(g)
```

```
type(f) <type 'sage.rings.real_mpfr.RealNumber'>

---------------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
 last)
<ipython-input-17-7220011bfae7> in <module>()
      6 print "type(f)", type(f)
      7
----> 8 g = f.polynomial(QQ)
      9
     10 ### put f as a polynomial with QQ (rational numbers) as coeffi
cients

sage/structure/element.pyx in sage.structure.element.Element.__getattr
__ (build/cythonized/sage/structure/element.c:4303)()

sage/structure/element.pyx in sage.structure.element.Element.getattr_f
rom_category (build/cythonized/sage/structure/element.c:4412)()

sage/cpython/getattr.pyx in sage.cpython.getattr.getattr_from_other_cl
ass (build/cythonized/sage/cpython/getattr.c:1834)()

AttributeError: 'sage.rings.real_mpfr.RealNumber' object has no attrib
ute 'polynomial'
```

In [18]:

```
f = (x^5).polynomial(QQ)

print_remainder(f,2)
```

```
---------------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
 last)
<ipython-input-18-efbd6e69877f> in <module>()
----> 1 f = (x**Integer(5)).polynomial(QQ)
      2
      3 print_remainder(f,Integer(2))

sage/structure/element.pyx in sage.structure.element.Element.__getattr
__ (build/cythonized/sage/structure/element.c:4303)()

sage/structure/element.pyx in sage.structure.element.Element.getattr_f
rom_category (build/cythonized/sage/structure/element.c:4412)()

sage/cpython/getattr.pyx in sage.cpython.getattr.getattr_from_other_cl
ass (build/cythonized/sage/cpython/getattr.c:1834)()

AttributeError: 'sage.rings.real_mpfr.RealNumber' object has no attrib
ute 'polynomial'
```

In [19]:

```
### sample code for recursive programming

### suppose f(x) = a_0 + a_1(x-3) + a_2(x-3)^2 + ... + a_10(x-3)^10
### let's find a_0, ..., a_10

def print_remainder(f, a):
    """
    Input:
        f: a polynomial
        a: a value
    Output:
        print a_0, ..., a_n recursively.
    """
    if f == (0*x).polynomial(QQ):
        return None
    else:
        quo, rem = f.quo_rem((x-a).polynomial(QQ))
        print rem
        print_remainder(quo, a)
```

In [20]:

```
###請問要如何讓迴圈中的i可以以小數點來計算?

print 'first kind'
for i in range(10):
    k = 3 - 10^(-i)
    print N(k)

print '---'
print 'second kind'
for i in range(10):
    k = 3 + 0.1 * i
    print k

#for i in range(2,4):
# g=(i+1)/(i^2+2*i-3)
# print g;


### Jephian: I don't quite understand what you plan to do?
```

```
first kind
2.00000000000000
2.90000000000000
2.99000000000000
2.99900000000000
2.99990000000000
2.99999000000000
2.99999900000000
2.99999990000000
2.99999999000000
2.99999999900000
---
second kind
3.00000000000000
3.10000000000000
3.20000000000000
3.30000000000000
3.40000000000000
3.50000000000000
3.60000000000000
3.70000000000000
3.80000000000000
3.90000000000000
```

In [21]:

```
f = (5*(x-3)^2 + 3*(x-3)).polynomial(QQ)

print_remainder(f,3)
```

```
---------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
 last)
<ipython-input-21-d00c6443d239> in <module>()
----> 1 f = (Integer(5)*(x-Integer(3))**Integer(2) + Integer(3)*(x-In
teger(3))).polynomial(QQ)
      2
      3 print_remainder(f,Integer(3))
      4

sage/structure/element.pyx in sage.structure.element.Element.__getattr
__ (build/cythonized/sage/structure/element.c:4303)()

sage/structure/element.pyx in sage.structure.element.Element.getattr_f
rom_category (build/cythonized/sage/structure/element.c:4412)()

sage/cpython/getattr.pyx in sage.cpython.getattr.getattr_from_other_cl
ass (build/cythonized/sage/cpython/getattr.c:1834)()

AttributeError: 'sage.rings.real_mpfr.RealNumber' object has no attrib
ute 'polynomial'
```

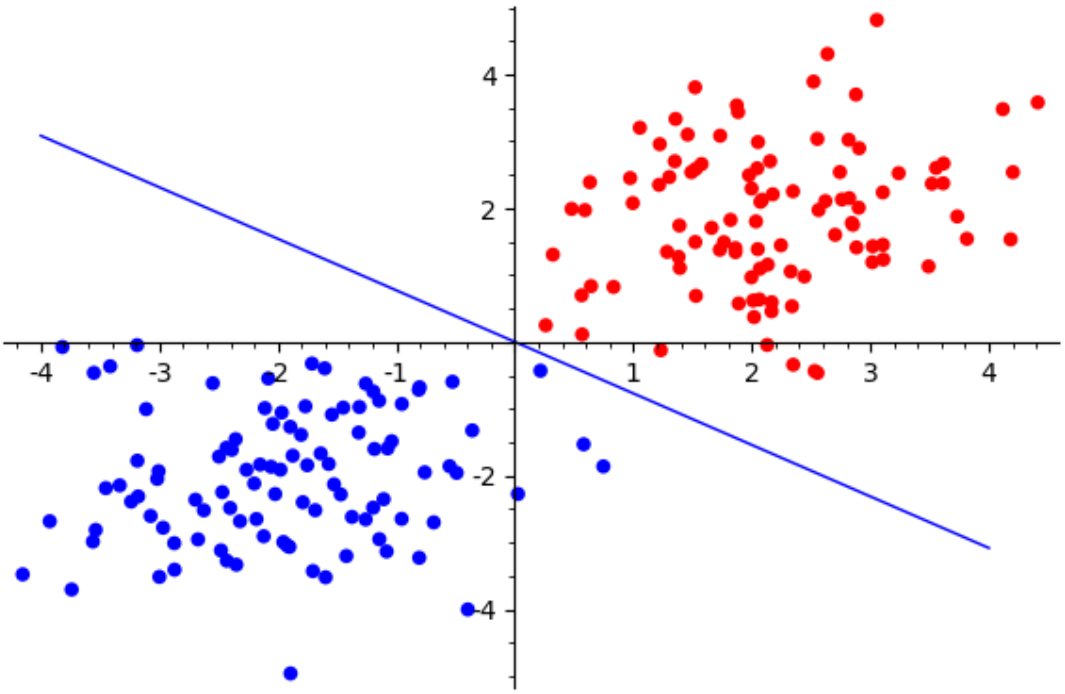
Linear Classifier

Overview

描述 Linear Classifier 的目的。

給定兩群正負對比且向量化的資料 A 與 B
其中 A 裡面的每筆資料都為以下形式
$(x_1, x_2, \ldots, x_m, 1)$
而 B 裡面的每筆資料都為以下形式
$(x_1, x_2, \ldots, x_m, -1)$。

目的是找到一個通過原點的超平面 $h : z_1x_1 + z_2x_2 + \cdots + z_mx_m = 0$ 且其法向量為 $\mathbf{v} = (z_1, z_2, \ldots, z_m)$，使得 A, B 兩群資料被一分為二。



Algorithm

假設有兩群正負對比的資料，每筆資料有 m 個不同屬性的類型和 1 個判斷正負的類型。
輸入 n 筆資料 $[(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)], \mathbf{x}_i \in \mathbb{R}^m$ and $y_i \in \{-1, 1\}$ for $i = 1, 2, \ldots, n$。
如果資料為正，則 $y_i = 1$。如果資料為負，則 $y_i = -1$。
輸出 $\mathbf{v}$ 使得 $\begin{cases} \mathbf{v} \cdot \mathbf{x}_i > 0 & , \text{if } y_i = 1 \\ \mathbf{v} \cdot \mathbf{x}_i < 0 & , \text{if } y_i = -1 \end{cases}$ for $i = 1, 2, \ldots, n$。

1. 初始值 $\mathbf{v} = \mathbf{0}$
2. 假設 $\begin{cases} \mathbf{v} \cdot \mathbf{x}_i > 0 & \text{and } y_i = -1 \\ \mathbf{v} \cdot \mathbf{x}_i < 0 & \text{and } y_i = 1 \end{cases}$, for some $i = \{1, 2, \dots, n\}$ ----------(1)
   令 $\mathbf{v} = \mathbf{v} + y_i \mathbf{x}_i$。
3. 重複第 2 步驟直到沒有任何點滿足 (1)。

Explanation

假設有一個超平面區分開兩筆資料，且超平面為 $y = \mathbf{v}$。
且 $\mathbf{v}$ 經過上述演算法後，會在正號資料的那一區，
使得正號資料跟 $\mathbf{v}$ 內積為正，負號資料跟 $\mathbf{v}$ 內積為負。

之後如有資料 $\mathbf{x} \in \mathbb{R}^m$ 輸入，
則 $\begin{cases} \mathbf{v} \cdot \mathbf{x} > 0 & \text{, then } y = 1, \text{歸類在正號區。} \\ \mathbf{v} \cdot \mathbf{x} < 0 & \text{, then } y = -1, \text{歸類在負號區。} \end{cases}$

Convergence of Linear Classifier

假設存在一個法向量 $\mathbf{v}$ 能夠將資料分成兩群，那必定會在有限的次數內找到 $\mathbf{v}$ 。

Proof :

Suppose $|\mathbf{v}| = 1, R = \max |\mathbf{x}_i|, \alpha = \min |\mathbf{v} \cdot \mathbf{x}_t|$ and $\mathbf{v}_1 = \mathbf{0}$.

Let $\mathbf{v}_k = \mathbf{v}_{k-1} + y_t \mathbf{x}_t$.

We note that $\mathbf{x}_t$ is chosen because $\mathbf{v}_{k-1}$ failed to classify $\mathbf{x}_t$.
That is, $\mathbf{v}_{k-1} \cdot \mathbf{x}_t$ and y_t has opposite signs.
Equivalently, $\mathbf{v}_{k-1} \cdot y_t \mathbf{x}_t < 0$.

(1)

$$\mathbf{v}_k \cdot \mathbf{v} = (\mathbf{v}_{k-1} + y_t \mathbf{x}_t)\mathbf{v} \geq \mathbf{v}_{k-1} \cdot \mathbf{v} + \alpha = (\mathbf{v}_{k-2} + y_t \mathbf{x}_t)\mathbf{v} + \alpha \geq \mathbf{v}_{k-1} \cdot \mathbf{v} + 2\alpha \geq \cdots \geq k\alpha$$

$$\implies \mathbf{v_k} \cdot \mathbf{v} \geq k\alpha$$

(2)

$$\begin{aligned} \|\mathbf{v}_k\|^2 &= \|\mathbf{v}_{k-1} + y_t \mathbf{x}_t\|^2 \\ &= \|\mathbf{v}_{k-1}\|^2 + 2(\mathbf{v}_{k-1} \cdot y_t \mathbf{x}_t) + \|\mathbf{x}_t\|^2 \\ &\leq \|\mathbf{v}_{k-1}\|^2 + \|\mathbf{x}_t\|^2 \\ &\leq \|\mathbf{v}_{k-1}\|^2 + R^2 \leq \cdots \leq kR^2 \end{aligned}$$

$$\implies \|\mathbf{v}_k\|^2 \leq kR^2$$

From (1)(2), by Cauchy–Schwarz inequality

$$k\alpha \leq \mathbf{v}_k \cdot \mathbf{v} \leq \|\mathbf{v}_k\| \|\mathbf{v}\| \leq \sqrt{kR^2}$$

$$\implies k\alpha \leq \sqrt{k}R$$

$$\implies k \leq \frac{R^2}{\alpha^2}$$

Implementation

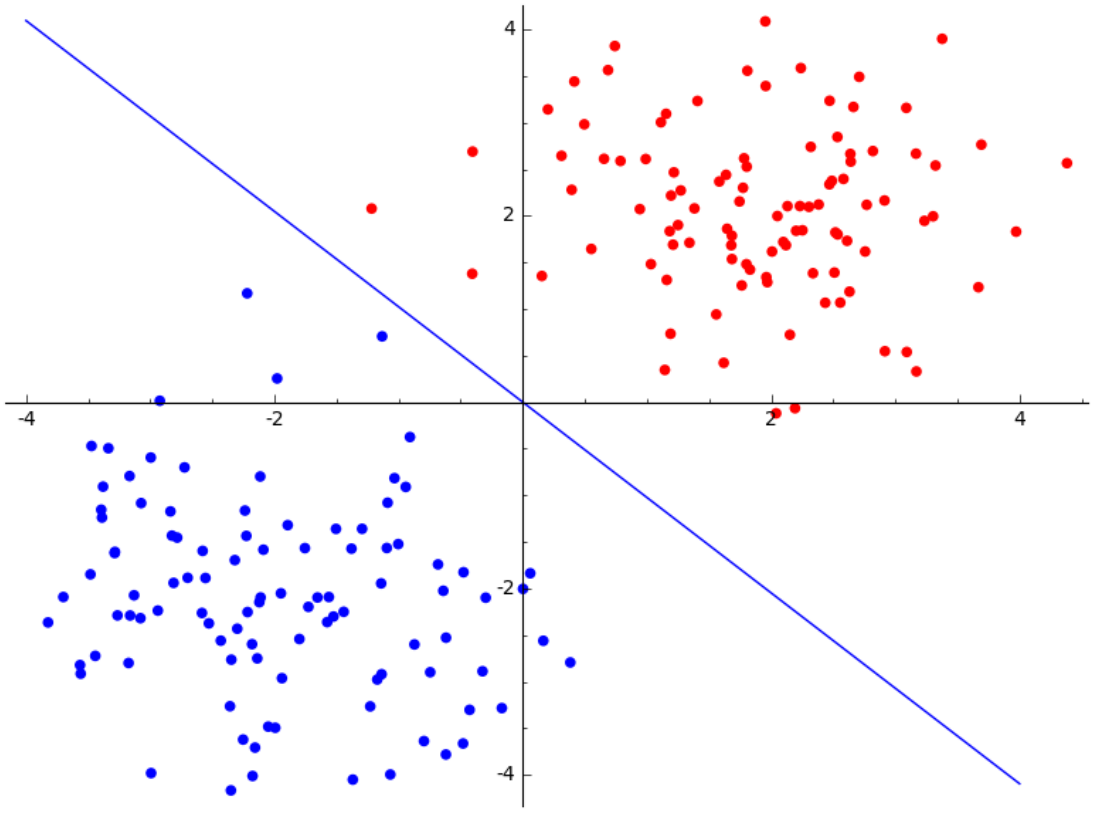
In [2]:

```
def discriminate(data, v):
    for i in data:
        tmp = i[0]*v[0] + i[1]*v[1]
        if (tmp*i[2])<=0:
            return [v[0] + i[2]*i[0], v[1] + i[2]*i[1]]
    else:
        return v

def linear_classifier(data):
    """
    Input:
        data: a list of pairs [(x1,y1,k), ..., (xN,yN,k)], k={-1,+1}
    Output:
        Output v so that the line v = a (xi,yi)
    """
    v = [0,0]

    while(v != discriminate(data, v)):
        v = discriminate(data, v)
    return v
```

Examples 1.

In [8]:

```
import numpy as np

n = 100
x1 = np.random.randn(n) + 2
y1 = np.random.randn(n) + 2
data1 = zip(x1,y1)
p = point(data1, rgbcolor='red', size=30)

x2 = np.random.randn(n) - 2
y2 = np.random.randn(n) - 2
data2 = zip(x2,y2)
q = point(data2, rgbcolor='blue', size=30)

data = zip(x1, y1, [1]*n) + zip(x2, y2, [-1]*n)
# print data
v = linear_classifier(data)
x = var('x')
g = -(v[0]/v[1])*x
pic = g.plot(xmin=-4,xmax=4)

f=p+q+pic
f.show()
```



In [ ]:

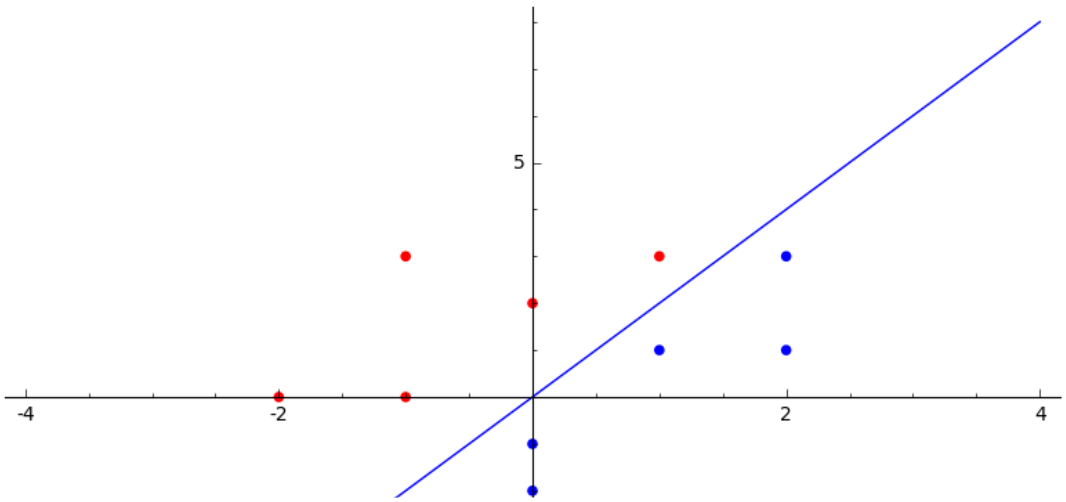
```
data
```


Examples 2.

In [9]:

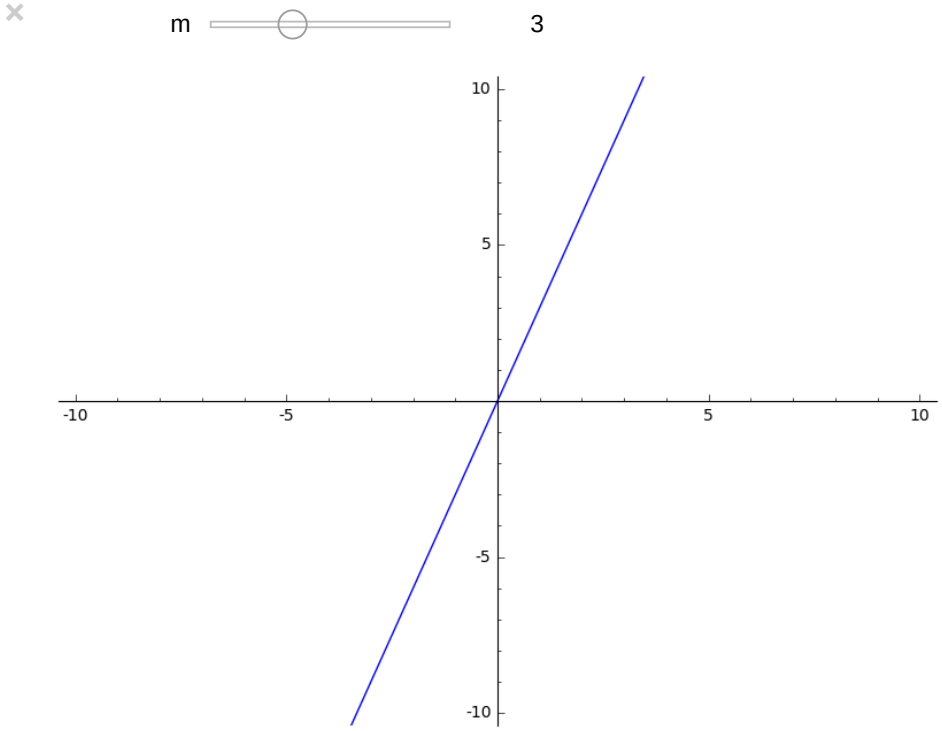
```
import numpy as np

n=5
x1 = [1,0,-1,-2,-1]
y1 = [3,2,3,0,0]
data1 = zip(x1,y1)
p = point(data1, rgbcolor='red', size=30)

x2 = [1,0,0,2,2]
y2 = [1,-1,-2,3,1]
data2 = zip(x2,y2)
q = point(data2, rgbcolor='blue', size=30)

data = zip(x1, y1, [1]*n) + zip(x2, y2, [-1]*n)
print data
v = linear_classifier(data)
print v
x = var('x')
g = -(v[0]/v[1])*x
pic = g.plot(xmin=-4,xmax=4)

f=p+q+pic
f.show()
```

```
[(1, 3, 1), (0, 2, 1), (-1, 3, 1), (-2, 0, 1), (-1, 0, 1), (1, 1, -
1), (0, -1, -1), (0, -2, -1), (2, 3, -1), (2, 1, -1)]
[-4, 2]
```



In [6]:

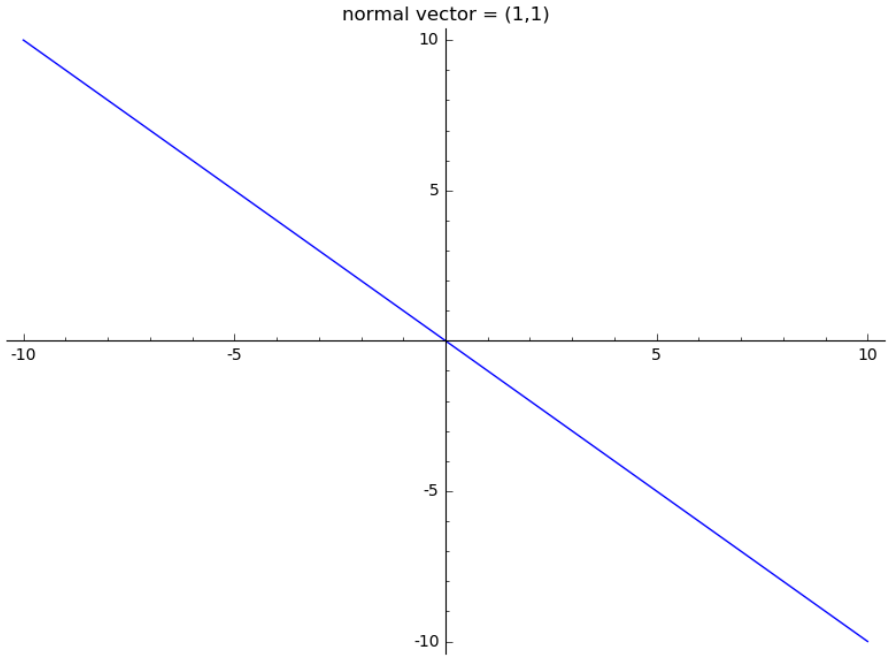
```
@interact

def _(m=slider(list(range(10)))): ### the name of the function does not matter, so
    f = m*x
    pic = f.plot(xmin=-10,xmax=10,ymin=-10,ymax=10)
    pic.show()

## See some examples here:
## http://doc.sagemath.org/html/en/prep/Quickstarts/Interact.html
## However, this only works in SageMath but not for Python in general.
```



In [7]:

```
n_vecs = [(1,1),(1,0),(1,-1),(2,-1),(3,-2),(4,-3)]
num_vects = len(n_vecs)

@interact
def _(step=slider(list(range(num_vects)))): ### the name of the function does not m
    vec = n_vecs[step]
    if vec[1] == 0:
        pic = plot(line([(0,-10),(0,10)]))
    else:
        m = -vec[0]/vec[1]
        f = m*x
        pic = f.plot(xmin=-10,xmax=10,ymin=-10,ymax=10)
    pic.show(xmin=-10,xmax=10,ymin=-10,ymax=10,title='normal vector = (%s,%s)'%vec)

## See some examples here:
## http://doc.sagemath.org/html/en/prep/Quickstarts/Interact.html
## However, this only works in SageMath but not for Python in general.
```

step 0



In [12]:

In [0]:

Linear dependent

Overview

Given a subset S of a vector space A=($\vec{a}_1\ \vec{a}_2 \ldots\ldots, \vec{a}_n$) and A ∈ $\mathbb{R}^n$

linear dependent aims to check wether the subset of vector space is linear dependent

$$c_1\vec{a}_1 + c_2\vec{a}_2 + \ldots + c_n\vec{a}_n = \vec{0}$$

If $c_i \in R$,$c_i \neq 0$, for i=1,2,...,n , then S is linear dependent .

Algorithm

1. Given a subset S of a vector space A=($\vec{a}_1, \vec{a}_2, \ldots\ldots, \vec{a}_n$) and A ∈ $\mathbb{R}^n$
2. Use the #gauss method to check whether the vector is linear dependent

Guass thm

A vector after doing the Guass eliminatin thm ,

at least a row or a column which the number of the elements are all zero.

The vector is linear dependent.

In [1]:

```
import numpy as np

A = np.array(
    [
        [1, 1 ,2 ,3],
        [2, 1, 1, 2],
        [3, 0, 1, 5],
        [6, 2, 4, 10]
    ])
```

In [2]:

```
#Let a set of matrix from your vector(a1,...an)
```

In [3]:

```
#gauss method
def Gauss(A):
    si=A.shape
    Y=A
    for j in range(1,1+si[1]):
        for i in range(1,1+si[0]-j):
            p1=Y[i+j-1,j-1]/Y[j-1,j-1]
            if p1>1e+16 or p1<-1e+16 or ( Y[i+j-1,j-1]==0 and Y[i-1,j-1]==0 ):
                tem=Y[i+j-1,:]
                Y[i+j-1,:]=Y[j-1,:]
                Y[j-1,:]=tem
            else:
                Y[i+j-1,:]=Y[i+j-1,:]-p1*Y[j-1,:]
    return Y
```

In [4]:

```
nA=Gauss(A)
nA
# now you can check matrix nA has zero vector?
```

Out[4]:

```
array([[ 1,  1,  2,  3],
       [ 0, -1, -3, -4],
       [ 0,  0,  4,  8],
       [ 0,  0,  0,  0]])
```

In [0]:

Matrix exponential

Overview

我們的目標是寫出一個函數，然後在人們輸入一個矩陣 A 以及一個正整數 N 的情況下，使得我們的函數最終能夠輸出一個泰勒級數的前 N 項和

Algorithm

1. Create an $n \times n$ matrix $A = \begin{bmatrix} x_{11} & \cdots & x_{n1} \\ \vdots & \ddots & \vdots \\ x_{1n} & \cdots & x_{nn} \end{bmatrix}$ and input an integer N.
2. Then compute A^k for $k = 0, \ldots, N$ and also add the term $\frac{1}{k!}A^k$ to the current total.

Explanation

目標是想找到矩陣的指數 e^A 運算過後出來所得的值

對於 $n \times n$ 階矩陣 A，我們可以定義矩陣指數 (matrix exponential)，方法是仿照指數函數的冪級數定義，如下的常數指數：

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

將變數 x 以矩陣 A 取代，常數 1 以單位矩陣 I 取代，矩陣指數 e^A 即為下列冪矩陣級數

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots$$

對於任一 x，指數函數 e^x 總會收斂；同樣地，對於任一 A，矩陣指數 e^A 也總會收斂。證明於下，令：

$$e_m(A) = \sum_{k=0}^{m} \frac{A^k}{k!}$$

若 $m > p$ ，則

$$e_m(A) - e_p(A) = \sum_{k=0}^{m} \frac{A^k}{k!} - \sum_{k=0}^{p} \frac{A^k}{k!} = \sum_{k=p+1}^{m} \frac{A^k}{k!}$$

使用矩陣範數的不等式性質，

$$\|e_m(A) - e_p(A)\| \leq \sum_{k=p+1}^{m} \frac{\|A\|^k}{k!}$$

上式將問題帶回純量的情況。當 m 和 p 同趨於無窮大，不等式右邊趨於零。

Implementation

In [1]:

```
def matrix_exponential(A, N):

    n = A.dimensions()[0]
    current = identity_matrix(n)
    extra = A

    for p in range(1,N+1):
        current = current + extra
        extra = extra * A / (p+1)

    return current
```

In [2]:

```
### 輸入一個矩陣 A

A = matrix([
    [1,2,3,1],
    [2,3,4,1],
    [3,4,5,1],
    [1,0,1,0]
])

N(matrix_exponential(A,100))
```

Out[2]:

```
[2757.73213796075 3830.83971400187 5148.75408439439 1196.01097321682]
[3970.57351849152 5518.37372994342 7414.96317963041 1722.19483056945]
[5184.41489902228 7203.90774588497 9682.17227486642 2248.37868792207]
[809.560920353845 1123.68934396104 1511.13939682401 351.789238235083]
```

Examples

In [3]:

```
### Example 1
def matrix_exponential(A, N):

    n = A.dimensions()[0]
    current = identity_matrix(n)
    extra = A

    for p in range(1,N+1):
        current = current + extra
        extra = extra * A / (p+1)

    return current

A = matrix([
    [1,0,0],
    [0,1,0],
    [0,0,1]
])
print' when A ='
print A
print 'the exponential of A in Taylor 0 is'
print matrix_exponential(A,0)
```

```
 when A =
[1 0 0]
[0 1 0]
[0 0 1]
the exponential of A in Taylor 0 is
[1 0 0]
[0 1 0]
[0 0 1]
```

In [4]:

```
### Example 2
def matrix_exponential(A, N):

    n = A.dimensions()[0]
    current = identity_matrix(n)
    extra = A

    for p in range(1,N+1):
        current = current + extra
        extra = extra * A / (p+1)

    return current

A = matrix([
    [1,0,0,0],
    [0,1,0,0],
    [0,0,1,0],
    [0,0,0,1]
])

print' when A ='
print A
print 'the exponential of A in Taylor 4 is'
print matrix_exponential(A,4)
```

```
 when A =
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
the exponential of A in Taylor 4 is
[65/24     0     0     0]
[    0 65/24     0     0]
[    0     0 65/24     0]
[    0     0     0 65/24]
```

In [5]:

```
### Example 3
def matrix_exponential(A, N):

    n = A.dimensions()[0]
    current = identity_matrix(n)
    extra = A

    for p in range(1,N+1):
        current = current + extra
        extra = extra * A / (p+1)

    return current

A = matrix([
    [1,1,1,1],
    [1,1,1,1],
    [1,1,1,1],
    [1,1,1,1]
])
print' when A ='
print A
print 'the exponential of A in Taylor 1 is'
print matrix_exponential(A,1)
```

```
 when A =
[1 1 1 1]
[1 1 1 1]
[1 1 1 1]
[1 1 1 1]
the exponential of A in Taylor 1 is
[2 1 1 1]
[1 2 1 1]
[1 1 2 1]
[1 1 1 2]
```

In [6]:

```
### Example 4
def matrix_exponential(A, N):

    n = A.dimensions()[0]
    current = identity_matrix(n)
    extra = A

    for p in range(1,N+1):
        current = current + extra
        extra = extra * A / (p+1)

    return current

A = matrix([
    [1,0,0,0],
    [0,0,0,0],
    [0,0,0,1],
    [0,0,0,0]
])
print' when A ='
print A
print 'the exponential of A in Taylor 5 is'
print matrix_exponential(A,5)
```

```
 when A =
[1 0 0 0]
[0 0 0 0]
[0 0 0 1]
[0 0 0 0]
the exponential of A in Taylor 5 is
[163/60      0      0      0]
[     0      1      0      0]
[     0      0      1      1]
[     0      0      0      1]
```

In [7]:

```
### Example 5
def matrix_exponential(A, N):

    n = A.dimensions()[0]
    current = identity_matrix(n)
    extra = A

    for p in range(1,N+1):
        current = current + extra
        extra = extra * A / (p+1)

    return current

A = matrix([
    [1,0,0,0,1],
    [0,0,0,0,0],
    [0,0,0,0,0],
    [0,0,0,0,0],
    [1,0,0,0,1]
])
print' when A ='
print A
print 'the exponential of A in Taylor 10 is'
print matrix_exponential(A,10)
```

```
 when A =
[1 0 0 0 1]
[0 0 0 0 0]
[0 0 0 0 0]
[0 0 0 0 0]
[1 0 0 0 1]
the exponential of A in Taylor 10 is
[19819/4725          0          0          0 15094/4725]
[          0          1          0          0          0]
[          0          0          1          0          0]
[          0          0          0          1          0]
[15094/4725          0          0          0 19819/4725]
```

In [ ]:

Maximum matchings with any edge you want

Overview

First, we need to know a few things about the maximum matching.
Maximum, the largest size of a matching.
A **matching** means a set of pairwise non-adjacent edges, none of which are loops; that is, no two edges share a common vertex.
Maximum matchings is the largest number of pairwise non-adjacent edges in a set.

程式邏輯

Select an edge, select the two points (B, C) of the edge, and regard it as a path. Take the shrimp to extend the path,
use the selected two sides as a waypoint, and extend one edge to create a path.
New path (A, B, C, D) and reverse the label of the entire path.

舉例:標記B-C邊，將原本的路徑擴充行新的路徑A-B-C-D，並將路徑上的標記做反轉，使得標記變為A-B、C-D

Algorithm

1.Input your graph.
2.mark any edges you choose, regarded as a path.
3.make its vertices expand along the edge and the other two vertices and invert the edges of the mark until all edges are selected.

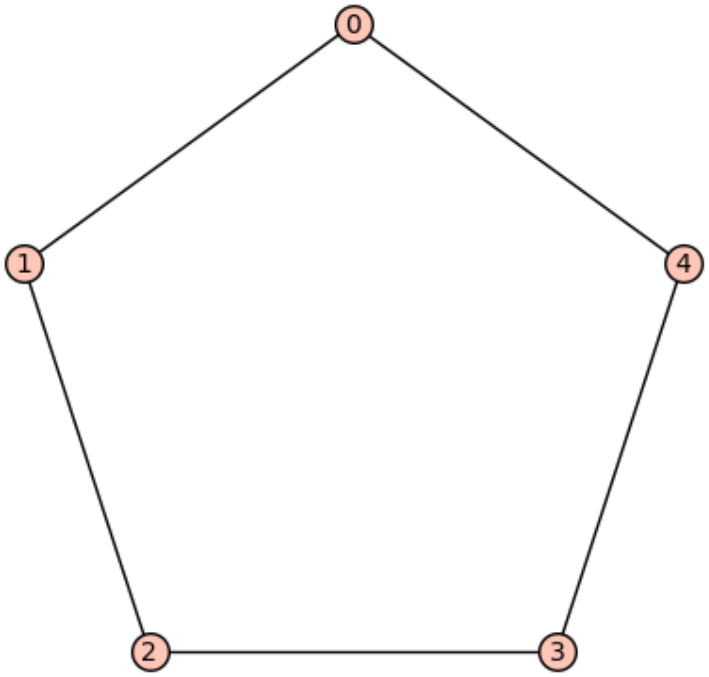
Explanation

可以選取一條邊並找島包含他的最大值，因為它是延你所選的邊一條一條的延伸，所以不會有遺漏任何邊

Implementation

In [1]:

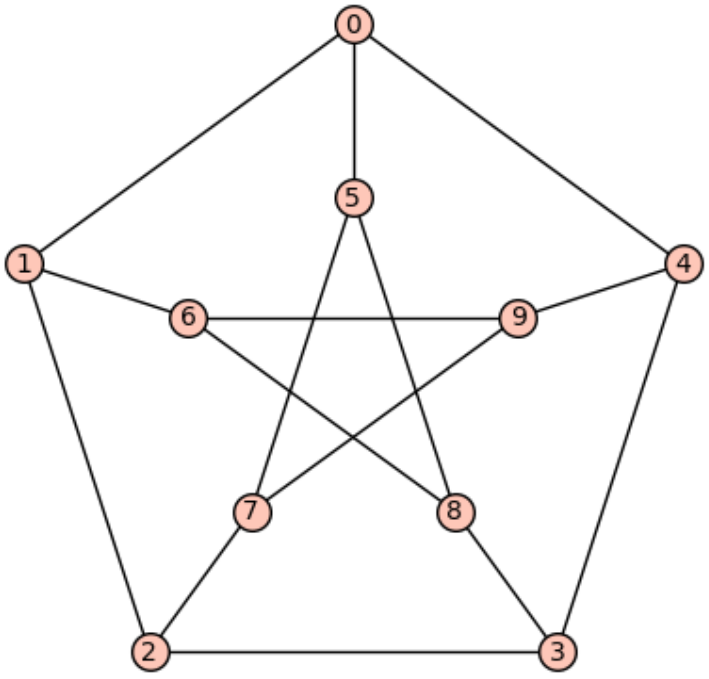
```
### some functions related to your project

### `graphs` contains several built-in graphs
g = graphs.CycleGraph(5)
g.show()

print 'vertices', g.vertices()
print 'edges', g.edges()
print 'edges without label', g.edges(labels=False)

print 'neighbor of 0', g.neighbors(0)

### run through all combinations of k edges
k = 2
for edge_set in Combinations(g.edges(labels=False)):
    pri
```




```
vertices [0, 1, 2, 3, 4]
edges [(0, 1, None), (0, 4, None), (1, 2, None), (2, 3, None), (3, 4,
None)]
edges without label [(0, 1), (0, 4), (1, 2), (2, 3), (3, 4)]
neighbor of 0 [1, 4]

---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
 last)
<ipython-input-1-d6548d32fdd2> in <module>()
     14 k = Integer(2)
     15 for edge_set in Combinations(g.edges(labels=False)):
---> 16     pri

NameError: name 'pri' is not defined
```

In [2]:

```
# g = graphs.CycleGraph(5)
g = graphs.PetersenGraph()
g.show()
```



In [0]:

```

```

Minimal Polynomial

Overview

Cayley–Hamilton定理:

- A 為一個 $n \times n$ 階矩陣且 $p(t)$ 為 A 的特徵多項式($p(t) = \det(A - tI)$)。將矩陣 A 替換 t 得到的矩陣多項式滿足 $p(A) = O$。

最小多項式 $m_A(t)$:

1. 滿足 $p(A) = O$ 的方程式中次數為最小者
2. 最高次項係數為1

最小多項式的重要性質:

1. A 之所有的特徵值 λ_i 皆使 $m_A(\lambda_i) = 0$
2. 若 $p(t) = (t-\lambda_1)^{m_1} \times (t-\lambda_2)^{m_2} \times \cdots \times (t-\lambda_k)^{m_k}, m_1 + m_2 + \cdots m_k = n$
   則 $m_A(t) = (t-\lambda_1)^{d_1} \times (t-\lambda_2)^{d_2} \times \cdots \times (t-\lambda_k)^{d_k}, 1 \le d_i \le m_i, \forall i$
3. A 可對角化 $\iff$ $m_A(t) = (t-\lambda_1) \times (t-\lambda_2) \times \cdots \times (t-\lambda_k)$； i.e 最小多項式沒有重根

Algorithm

Computer $I, A, \ldots, A^n$ and flaten them into vectors in $\mathbb{R}^{n^2}$. Find the smallest k such that $\{I, A, \ldots, A^k\}$ is dependent.
Suppose $A = \left[a_{i,j}\right]$. You may flatten them as a vector $f(A) = (a_{1,1}, \ldots, a_{1,n}, a_{2,1} \ldots, a_{n,n})$ in $\mathbb{R}^{n^2}$
Create a huge matrix H whose columns are $f(I), f(A), \ldots, f(A^{n-1})$ and do Gaussian elimination on H. Find the minimum k such that $\{f(I), f(A), \ldots, f(A^k)\}$ is dependent.
Find coefficients so that $a_k f(A^k) + \ldots a_0 f(I) = 0$ and $a_k = 1$.
Thus $a_k A^k + \ldots a_0 I = O$ and $m(t) = a_k t^k + \cdots + a_0$ is the minimal polynomial.

Explanation

判斷$A_{n\times n}$是否可對角化

$$p(t) = det(A - tI) = (t-\lambda_1)^{m_1} \times (t-\lambda_2)^{m_2} \times \cdots \times (t-\lambda_k)^{m_k}$$

找出$p(t)$的最小多項式$m_A(t)$使得$m_A(A) = 0$
如果$m_A(t)$中$(t-\lambda_i)$的次數皆為1次，則A為可對角化
如果$m_A(t)$中其中一項$(t-\lambda_i)^k, 2 \le k$，則A為不可對角化

In [1]:

```
def minimalpolynomial(data):
    """
    Input:
        data:  給定一方陣A
    Output:
        Output 最小多項式 m_A(t) 中沒有重根，則方陣A為可對角化。
    """
    data.change_ring(QQ)
    m,n = data.dimensions()
    if (m == n):
        I = identity_matrix(n)                                          #單位矩陣
        flattened_I = [I[i,j] for i in range(m) for j in range(n)] #攤平單位矩陣
        flattened_data = [flattened_I] #將I存成[I]

        for k in range(1,m+1):
            data_i=data^k
            flattened_data_i= [data_i[i,j] for i in range(m) for j in range(n)] #da
            flattened_data.append(flattened_data_i) #將矩陣變為[I,data^1,...,data^k]
        M = matrix(flattened_data).transpose()
        show(M)

        r = M.rank()
        sliced_M = M[:,0:r+1]
        show(sliced_M)
        factor=sliced_M.right_kernel()
        show(factor.basis()[0])
        A=matrix(factor.basis()[0])



        poly=0
        for k in range(0,r+1):    #取出係數將其變為一多項式
            item=x^k*A[0,k]
            poly+=item
        if(A[0,r]<0):             #讓首項系數為1
            poly*=-1
        return poly

    else:
        print'data is not square matrix'

def diagonalizable(data):
    poly=minimalpolynomial(data)
    show(poly)
    polyprime=poly.derivative()
    show(polyprime)
    poly.gcd(polyprime)
    show(poly.gcd(polyprime))
    if (poly.gcd(polyprime)==1):
        print'The square matrix is diagonalizable'   # If p and pprime has common f
    else:
        print'The square matrix is non-diagonalizable'
```

Examples

In [2]:

```
### diagonalizable
data= matrix([[3,-3,2],[-1,5,-2],[-1,3,0]])
diagonalizable(data)
```

$$\begin{pmatrix} 1 & 3 & 10 & 36 \\ 0 & -3 & -18 & -84 \\ 0 & 2 & 12 & 56 \\ 0 & -1 & -6 & -28 \\ 1 & 5 & 22 & 92 \\ 0 & -2 & -12 & -56 \\ 0 & -1 & -6 & -28 \\ 0 & 3 & 18 & 84 \\ 1 & 0 & -8 & -48 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 3 & 10 \\ 0 & -3 & -18 \\ 0 & 2 & 12 \\ 0 & -1 & -6 \\ 1 & 5 & 22 \\ 0 & -2 & -12 \\ 0 & -1 & -6 \\ 0 & 3 & 18 \\ 1 & 0 & -8 \end{pmatrix}$$

$(8, -6, 1)$

$x^2 - 6x + 8$

$2x - 6$

1

```
The square matrix is diagonalizable
```

In [3]:

```
### random matrix diagonalizable
import numpy as np
data= matrix(np.random.randint(0,500,[4,4]))
diagonalizable(data)
```

$$\begin{pmatrix} 0 & 113 & 136987 & 101990395 & 76383908518 \\ 1 & 312 & 144811 & 135994876 & 114441983923 \\ 0 & 229 & 135805 & 89051182 & 71051234020 \\ 0 & 438 & 215563 & 128340085 & 104942141941 \\ 0 & 342 & 177923 & 129898988 & 107422019453 \\ 0 & 19 & 278639 & 224322782 & 159353608592 \\ 1 & 185 & 120968 & 128485295 & 104684956379 \end{pmatrix}$$

$(2414326200, -54066099, 166007, -912, 1)$

$x^4 - 912\,x^3 + 166007\,x^2 - 54066099\,x + 2414326200$

$4\,x^3 - 2736\,x^2 + 332014\,x - 54066099$

1

In [4]:

```
### non-diagonalizable
data = matrix([[1,-1],[1,3]])
diagonalizable(data)
```

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & -4 \\ 0 & 1 & 4 \\ 1 & 3 & 8 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & -4 \\ 0 & 1 & 4 \\ 1 & 3 & 8 \end{pmatrix}$$

$(4, -4, 1)$

$x^2 - 4\,x + 4$

$2\,x - 4$

$2\,x - 4$

```
The square matrix is non-diagonalizable
```

In [5]:

```
### non-diagonalizable
data =matrix([[1,0,0,0],[0,2,0,0],[0,0,4,1],[0,0,0,4]])
diagonalizable(data)
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 4 & 8 & 16 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 16 & 64 & 256 \\ 0 & 1 & 8 & 48 & 256 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 16 & 64 & 256 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 4 & 8 & 16 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 16 & 64 & 256 \\ 0 & 1 & 8 & 48 & 256 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 4 & 16 & 64 & 256 \end{pmatrix}$$

$(32,\ -64,\ 42,\ -11,\ 1)$

$x^4 - 11\,x^3 + 42\,x^2 - 64\,x + 32$

$4\,x^3 - 33\,x^2 + 84\,x - 64$

$x - 4$

The square matrix is non-diagonalizable

In [0]:

Monty Fall Problem

Overview

Algorithm

1. 假設有3個箱子，隨機選一個箱子放東西
2. 讓玩家選擇其中一項後，隨機開啟另外兩個其中之一的箱子，
   如果開到有東西的，則此局不算，重新開始。
3. 問玩家在選擇不要換，後開啟選擇的箱子。

Explanation

1. 有3個箱子A、B、C，隨機選一個箱子放球。
2. 讓一位玩家選擇其中一個箱子。
3. 開啟另外兩個其中一個箱子。
4. 如果開到有放球的則此局結束。 沒開到則繼續。
5. 然後詢問玩家要不要換，決定後直接開啟。

Implementation

In [1]:

```
from random import choice

def Monty_Fall_game():
    box = ['A','B','C']

    ball = choice(box)
    player = choice(box)
    not_selected = [x for x in box if x != player]
    opened_box = choice(not_selected)
    if opened_box == ball:
        return Monty_Fall_game()
    else:
        change = [x for x in box if x != player and x != opened_box]

        final = choice(change)
    return (final == ball)
```

In [2]:

```
def probability(k):
    counter = 0;
    for _ in range(k):
        if Monty_Fall_game():
            counter += 1
    return N(counter / k)
```

Examples

In [3]:

```
##重複1000次種球的機率
probability(1000)
```

Out[3]:

```
0.510000000000000
```

Type *Markdown* and LaTeX: α^2

The Monty Hall Problem

BY 顏廷維 Justin Yen

Overview

Algorithm

開始 Monty Hall 的遊戲
先設三個變數1、2、3
再來設兩個隨機的變數selection、prize
再用門去扣掉這兩個變數
定義一個新的變數 Opendoor
由於Opendoor 不會和selection跟prize的門一樣
所以最後面抉擇會是要繼續選擇selection
或是選擇另一個變數 change:
(不會是selection也不會是Opendoor)
再去比較最後
是selection=prize的機會比較高
還是change=prize得機率比較高

Explanation

對我來說這個問題有一個極其重要的地方
那就是遊戲中打開的門，一定會是非車的門
既然打開的門存在車子機率已經不在了
那在原本大局觀中，他的機率就會附加在那些還沒開過的門
舉個常見的例子、那便是一有一千道門裡頭只有一個有大獎
你一開始選了一道門、那目前那道門中獎機率是1/1000
非那道門中獎機率是999/1000、今天我們打開了剩下門中的998道
那些都沒有大獎、那原本999/1000的機率就會附加在剩下那道還沒開也還沒選的
這個就可以大致拿來形容Monty Hall 的問題

Implementation

In [3]:

```
import random

doors = [1,2,3]
##先製造三道門

prize = random.choice(doors)
##獎品在隨機一個門裡面

selection = random.choice(doors)
##自己的選擇也是其中一扇門

L = [x for x in doors if x != prize and x != selection]
##設Open door為不是你的選擇也不是獎品的一個門
opened_door = random.choice(L)
##由於門有可能不只一個可4只能選一個

montyopendoor = L[0]
q = [x for x in doors if x != selection and x != opened_door]
##再設change為非selection和非open door的門

change = q[0]

##老師程式可能還有一點小瑕疵，有可能會跑出Opendoor跟change是一樣的選項
##有可能是因為一開始的opendoor是隨機的L,然後到另一個迴圈時他又會那入考慮
##不過我試過出錯的機率很低，而且對答案不會有干擾
print "Prize: %r" % prize
print "Selection: %r " % selection
print "monty's open door %r " % montyopendoor
print "change: %r " % change
```

```
Prize: 2
Selection: 2
monty's open door 1
change: 1
```

In [4]:

```
##不換門的
def gods_plan():

    doors = [1,2,3]

    prize = random.choice(doors)
    selection = random.choice(doors)

    L = [door for door in doors if door != prize and door != selection]
    opened_door = random.choice(L)
    montyopendoor = L[0]

    q = [door for door in doors if door != selection and door != opened_door]
    change = q[0]

    return (selection == prize)
```

In [5]:

```
def selection(k):
    counter = 0;
    for _ in range(k):
        if gods_plan():
            counter += 1
    return N(counter / k)
```

In [6]:

```
selection(1000)
```

Out[6]:

```
0.323000000000000
```

In [7]:

```
##我是分隔線
```

In [8]:

```
##選擇另一道門的
def good_plan():

    doors = [1,2,3]

    prize = random.choice(doors)
    selection = random.choice(doors)

    L = [door for door in doors if door != prize and door != selection]
    opened_door = random.choice(L)
    montyopendoor = L[0]

    q = [door for door in doors if door != selection and door != opened_door]
    change = q[0]

    return (change == prize)
```

In [9]:

```
def change(k):
    counter = 0;
    for _ in range(k):
        if good_plan():
            counter += 1
    return N(counter / k)
```

In [10]:

```
change(5000)
```

Out[10]:

```
0.658800000000000
```

Torque

Overview

力矩：
我會先給輸入者輸入一個位置，這個位置代表物體的重心 $G(a_1, b_1, c_1)$，
然後再給輸入者輸入兩個向量，一個是受力的點 $O(a_2, b_2, c_2)$，
另一個是力的方向與大小 $F(a_3, b_3, c_3)$

算法：
先計算施力方向與受力點–重心連線的角度（令為 θ），再計算受力點–重心連線的長度（令為 y），
接著計算力的大小（令為 f），然後力矩為

$$fy\sin\theta$$

Algorithm

1. $\cos\theta = \dfrac{(a_1 - a_2) \times a_3 + (b_1 - b_2) \times b_3 + (c_1 - c_2) \times c_3}{\sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2 + (c_1 - c_2)^2} \times \sqrt{a_3^2 + b_3^2 + c_3^2}}$
2. $y = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2 + (c_2 - c_1)^2}$
3. $f = \sqrt{(a_3^2 + b_3^2 + c_3^2)}$
4. $x = fy\sqrt{1 - \cos^2\theta}$
5. output x

Explanation

因為力矩的定義是，力乘上力臂，力臂就是將重心與力臂的延伸線作相連接，且連接點要垂直，重心與這連接點的距離即為力臂。

為了要計算力臂的長度，我們先算出力與重心到受力點練線的角度，但角度無法直接算出來，所以我們藉由餘弦定理cos等於與角相鄰的兩邊向量內積除以兩邊向量長度的乘積，即可求得$\cos\theta$

然後再利用畢達哥拉斯恆等式

$$\sin^2\theta + \cos^2\theta = 1$$

來計算出

$$\sin\theta = \sqrt{1 - \cos^2\theta}$$

求出角度後，接著便要計算受力點與重心的距離，然後令為y

$$y = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2}$$

這時便可求得力臂等於$y\sin\theta = y\sqrt{1 - \cos^2\theta}$

接著將力的大小算出，也就是力的長度，即$f = |F|$，最後帶入力矩的公式：力乘以力臂

$$fy\sqrt{1-\cos^2\theta}$$

其算出的數值便是力矩

Implementation

In [1]:

```
def torque(G, O, F):
    a1,b1,c1 = G
    a2,b2,c2 = O
    a3,b3,c3 = F
    cos_numerator = (a1-a2)*a3 + (b1-b2)*b3 + (c1-c2)*c3
    cos_denominator = sqrt((a1-a2)^2 + (b1-b2)^2 + (c1-c2)^2) * sqrt(a3^2 + b3^2 +
    cos_value = cos_numerator / cos_denominator
    y = sqrt((a1-a2)^2 + (b1-b2)^2 + (c1-c2)^2)
    f = sqrt(a3^2 + b3^2 + c3^2)
    x = f*y*sqrt(1-cos_value^2)
    return N(x)
```

Examples

In [2]:

```
def torque(G, O, F):
    a1,b1,c1 = G
    a2,b2,c2 = O
    a3,b3,c3 = F
    cos_numerator = (a1-a2)*a3 + (b1-b2)*b3 + (c1-c2)*c3
    cos_denominator = sqrt((a1-a2)^2 + (b1-b2)^2 + (c1-c2)^2) * sqrt(a3^2 + b3^2 +
    cos_value = cos_numerator / cos_denominator
    y = sqrt((a1-a2)^2 + (b1-b2)^2 + (c1-c2)^2)
    f = sqrt(a3^2 + b3^2 + c3^2)
    x = f * y * sqrt(1 - cos_value^2)
    return N(x)
```

In [3]:

```
a1=0
b1=0
c1=0
a2=1
b2=1
c2=1
a3=3
b3=2
c3=1

G = [a1,b1,c1]
O = [a2,b2,c2]
F = [a3,b3,c3]

torque(G, O, F)
```

Out[3]:

```
2.44948974278318
```

In [4]:

```
a1=3
b1=4
c1=1
a2=9
b2=8
c2=10
a3=5
b3=6
c3=1

G = [a1,b1,c1]
O = [a2,b2,c2]
F = [a3,b3,c3]

torque(G, O, F)
```

Out[4]:

```
65.3987767469698
```

In [5]:

```
a1=12.5
b1=89
c1=0.5
a2=42
b2=35
c2=36.3
a3=4
b3=5
c3=79

G = [a1,b1,c1]
O = [a2,b2,c2]
F = [a3,b3,c3]

torque(G, O, F)
```

Out[5]:

```
4967.33716794018
```

In [ ]:

In [ ]:

In [2]:

In [ ]:

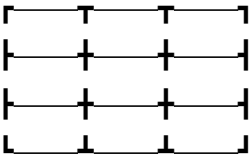
In [ ]:

Rook polynomial

Introduction

首先定義 $r_{B_k}(x)$ 為在棋盤 B 中放入不同行不同列 k 個棋子的可能種數，

例如：當 B 為 3×3 的棋盤



B_n 表示在board置入n個不同行不同列的城堡棋子

$r_{B_0}(x) = 1$, $r_{B_1}(x) = 9$, $r_{B_2}(x) = 18$,$r_{B_3}(x) = 6$

考慮 B 為 $m \times n$ 的棋盤（ $m \leq n$ ）

然後我們定義$R_{(m,n)}(x) = \sum_{k=0}^{m} r_{B_k} x^k$。

考慮一般的square board

Let $R_n(x) = R_{(n,n)}(x)$.

$R_1(x) = r_1(x)x + r_0(x) = x + 1$

$R_2(x) = r_2(x)x^2 + r_1(x)x + r_0(x) = 2x^2 + 4x + 1$

$R_3(x) = r_3(x)x^3 + r_2(x)x^2 + r_1(x)x + r_0(x) = 6x^3 + 18x^2 + 9x + 1$

$R_4(x) = r_4(x)x^4 + r_3(x)x^3 + r_2(x)x^2 + r_1(x)x + r_0(x) = 24x^4 + 96x^3 + 72x^2 + 16x + 1$

Overview

"rook polynomial" 是由John Riordan創造的。

在探討的是，在各種的board中，如何擺置不同行也不同列的城堡棋，依照

擺的數量不同，情況也會有所不同，進而藉由未知數x來表示我們擺的棋子

及其不同的種數，然而每個board也並非為完整沒有限制的。

Algorithm

Let B be a board (a set of grid points).
Suppose (i,j) is a point in B.
Let $B-(i,j)$ be the board obtained from B by removing the point (i,j).
Let $B(i,j)$ be the board obtained from B by removing the i-th row and the j-th column.
Then $R_B(x) = R_{B-(i,j)}(x) + xR_{B(i,j)}(x)$.

for example.

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

針對$(3,3)$格子

$$B-(3,3) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} B(3,3) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$

可得到$R_B(x) = R_{B-(3,3)}(x) + xR_{B(3,3)}(x) = (1+3x+1x^2) + x \times (1+1x) = 1+4x+2x^2$

Explanation

for nxn board

$B_{m,n}$ =

$$\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix}$$

where $b_{ij} = 0$ or 1 (0: black 1:white)

for example : B_1 =

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

means b_{11} is forbiddened and its rook polynomial is $R_{B_1}(x) = 1+8x+14x^2+4x^3$

and in

$$C = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

$$C_1 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$C_2 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

令 $r(C,x)$: rook polynomial,
$r(C_1,x)$, $r(C_2,x)$ 分別表左上及右下角禁區放的方法數,
則 $r(C_1,x) = 1x^0 + 5x^1 + 4x^2$, 其中
$1x^0$ 是表示在此區放 0 個 rook 有 1 種方法

$5x^1$ 是表示在此區放 1 個 rook 有 5 種方法
$4x^2$ 是表示在此區放 2 個 rook 有 4 種方法
同理可以列出 $r(C_2, x) = 1 + 5x + 5x^2$

=>

$$\begin{aligned} r(C, x) &= r(C_1, x) \times r(C_2, x) \\ &= (1 + 5x + 4x^2) \times (1 + 5x + 5x^2) \\ &= 1 + 10x + 34x^2 + 45x^3 + 20x^4 \end{aligned}$$

Implementation

In [1]:

```
### sample code for Rn

def rec_board(m,n):
    board = []
    for i in range(m):
        for j in range(n):
            board.append((i,j))
    return board

print "This is board(2,2)"
print rec_board(2,2)

def rook_polynomial(board):
    """
    Input:
        board: a list of possible positions
    Output:
        the rook polynomial of the board
    """
    ### trivial and base cases
    if len(board) == 0:
        return 1
    elif len(board) == 1:
        return x + 1
    else:
        i,j = board[0] ### pick a position
        board_minus_ij = copy(board)
        ### do something to create board_minus_ij
        ### board_minus_ij = board - (i,j)
        board_minus_ij.remove((i,j))

        board_ij = [pos for pos in board if pos[0] != i and pos[1] != j]

        #pos去掉i列j行 p[0]代表列,p[1]代表行

        ### previous code:
        #board_ij = copy(board)
        ### do something to create board_ij
        ###
        #for pos in board:
        #    if pos[0] == i or pos[1] == j:
        #        board_ij.remove(pos)
        return expand(rook_polynomial(board_minus_ij) + x*rook_polynomial(board_ij))

### This is called recursive programming
```

```
This is board(2,2)
[(0, 0), (0, 1), (1, 0), (1, 1)]
```

In [2]:

```
B = rec_board(2,3)
print B
f = rook_polynomial(B)
print f

f.subs(x=1)
```

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
6*x^2 + 6*x + 1
```

Out[2]:

```
13
```

Examples

In [3]:

```
import random

def rec_board(m,n):
    board = []
    for i in range(m):
        for j in range(n):
            board.append((i,j))
    return board

def random_board(m,n):
    """
    Input:
        m,n: dimensions of the board
        k: number of positions
    Output:
        a board in the m x n board with k random positions
    """
    k= random.randint(0,N(m*n))
    B = rec_board(m,n)
    random.shuffle(B)
    return B[:k]
```

In [4]:

```
A=random_board(5,5)
print A
```

```
[(1, 3), (4, 2), (1, 0), (2, 2), (2, 0), (0, 0), (3, 1), (4, 1), (3,
0), (1, 1), (2, 1), (2, 3), (3, 2), (4, 4), (3, 4), (1, 4), (3, 3)]
```

In [5]:

```
g=rook_polynomial(A)
print g
```

```
11*x^5 + 103*x^4 + 172*x^3 + 90*x^2 + 17*x + 1
```

In [6]:

```
g.subs(x=1)
##可能的種數
```

Out[6]:

394

參考的式子與資料

Jephien: Here are some hint for general board.
Let B be a board (a set of grid points).
Suppose (i,j) is a point in B.
Let $B-(i,j)$ be the board obtained from B by removing the point (i,j).
Let $B(i,j)$ be the board obtained from B by removing the i-th row and the j-th column.
Then $R_B(x)=R_{B-(i,j)}(x)+xR_{B(i,j)}(x)$.
(This is my thought, you may check if that is correct.)

In [7]:

```
def rec_board(m,n):
    board = []
    for i in range(m):
        for j in range(n):
            board.append((i,j))
    return board

#print "This is board(2,2)"
B = rec_board(2,4)
print B
print len(B)
print B[5]
B.remove(B[3])
B
```

```
[(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3)]
8
(1, 1)
```

Out[7]:

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (1, 3)]
```

In [8]:

```
a = [1,2,3]
b = a

print a
b[0] = 5

print a
print b
```



```
[1, 2, 3]
[5, 2, 3]
[5, 2, 3]
```

In [9]:

```
a = [1,2,3]
b = copy(a)

print a
b[0] = 5

print a
print b
```

```
[1, 2, 3]
[1, 2, 3]
[5, 2, 3]
```

In [10]:

```
f = (x+1) * (x+4)
print f
```

```
(x + 4)*(x + 1)
```

In [11]:

```
print expand(f)
```

```
x^2 + 5*x + 4
```

In [12]:

```
sum([k^2 for k in range(1,11)])
```

Out[12]:

385

In [13]:

```
n = 30
sum([k^2 for k in range(1,n+1)])
```

Out[13]:

9455

In [ ]:

Shortest Paths

Overview

在圖上選一點作為出發點，出發點與終點間的路線上不會經過同一個點和同一條邊，為一個 **Path**。

利用 **Breadth first search** 找到兩點之間的最短距離（**shortest path**）。

Algorithm

Breadth first search

1.BFS(g,v,e) 輸入g,v,e 。g為一圖形，v為初始點，e為終點

2.level首先會記錄v點為0，reached中

3.

Explanation

先選定一點作為出發點，再輸入另一點作為結束點，將出發點標記為Level=0，接著找相鄰的點，標記為Level=1。

之後依次將與Level=1相鄰的點標記為Level=2，不斷往下標記到所有點都被標記為止。

Implementation

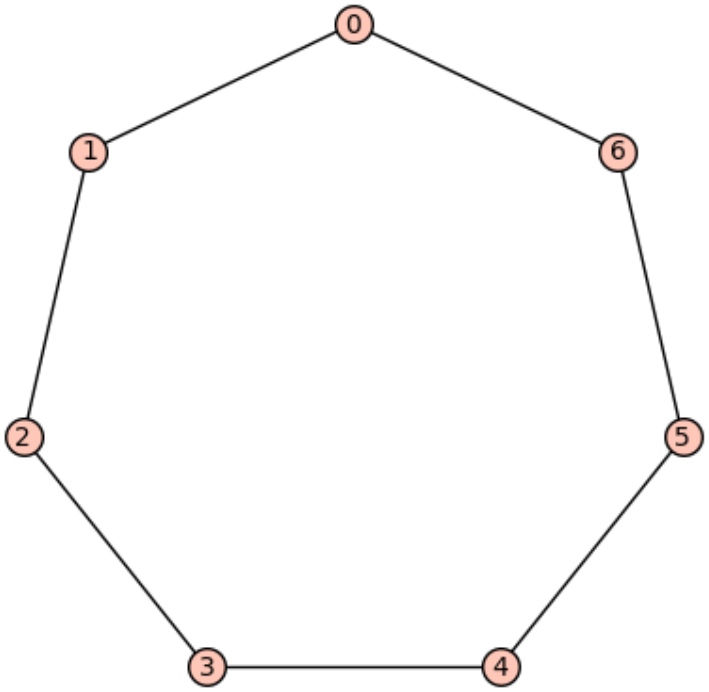
In [1]:

```
### The function below returns the order found by Breadth-First Search
def BFS(g, v, e):
    """
    Input:
        g: a simple graph
        v: a vertex
        e: end vertex
    Output:
        An order given by the BFS & distance of vertex v and e
    """
    level = {}      ### set level as a dictionary
    level[v] = 0    ### set level[v] as 0

    reached = [v]   ### vertices that have been found by BFS
    to_be_searched = [v]   ### vertices that have been found but we have not apply

    while to_be_searched != []:
        vtx = to_be_searched.pop(0) ### set vtx as the zero-th element and remove i
        extra_reached  = [i for i in g.neighbors(vtx) if i not in reached]
        l = level[vtx]
        for u in extra_reached:
            level[u]=l+1;

        reached = reached + extra_reached ### update reached
        to_be_searched = to_be_searched + extra_reached ### update to_be_searched
    print v,"和",e,"的最短距離 =",level[e];
    ###return  level[e];
```

Examples

In [2]:

```
### Example 1
g = graphs.CycleGraph(7)
g.show()
BFS(g,2,6)
```



2 和 6 的最短距離 = 3

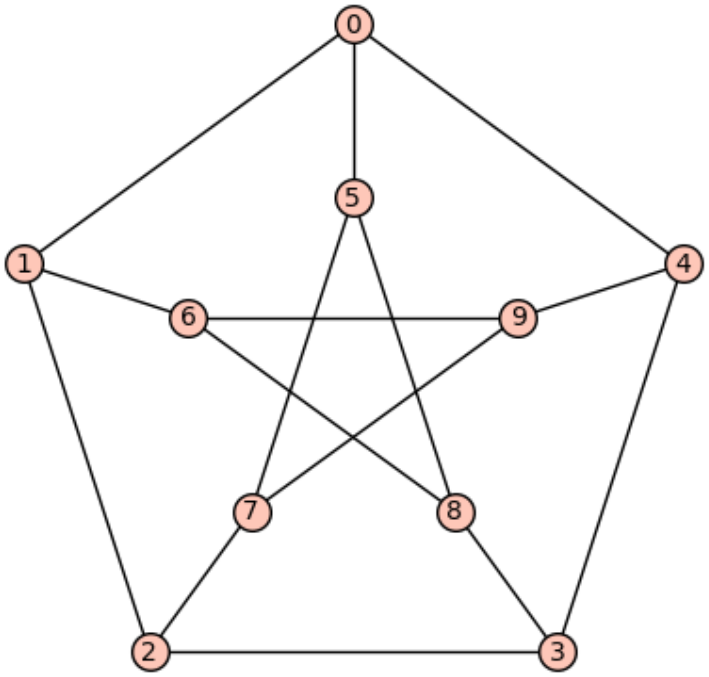
In [3]:

```
### Example 2
g = graphs.PathGraph(5)
g.show()
BFS(g,4,1)
```

4 和 1 的最短距離 = 3

In [4]:

```
### Example 3
g = graphs.PetersenGraph()
g.show()
BFS(g,2,3)
```



2 和 3 的最短距離 = 1

In [0]:

```

```

Spaces related to a matrix

Overview

輸入一個矩陣，利用矩陣的運算來求出該矩陣的行、列空間。

Review liner algebra

$RS(A) = \{xA : x \in \mathbb{F}_{1\times m}\}$ 為 A 之**各列任意組合**
$CS(A) = \{Ax : x \in \mathbb{F}_{n\times 1}\}$ 為 A 之**各行任意組合**

Algorithm

1. 先輸入矩陣 `A=B=matrix()`.
2. 將該矩陣的維度assign給 `m,n`.
3. 將矩陣裡的元素設定成有裡數.
4. 求出簡化列梯陣後即得到列(行)空間.
5. 設定一個迴圈將該矩陣的每一列(行)印出來，並刪除零列(行).
6. 最後便得到簡化後的列(行)空間.

Explanation

1. 目標是在求列空間與行空間,
2. 將矩陣中的元素設成有理數的原因是為了之後在做簡化列梯陣時能出現分數,
3. 而在做簡化列梯陣的過程就是將矩陣中每一列成為線性獨立,便能得到列空間.
4. 最後行空間的作法與列空間相似,只要先將輸入的矩陣轉置並做一樣的步驟便能得到行空間.

Implementation

In [1]:

```
#Row Space
def row_space(matrix):
    m,n = A.dimensions()
    AA = copy(A) ### make a copy so that we won't modify A
    AA = AA.change_ring(QQ) ### change the base ring to rational numbers QQ
    R = AA.echelon_form()
    print "簡化列梯陣為:"
    print R
    print "---"
    print "列空間為:"
    for i in range(m):
        if R[i,:]!=zero_matrix(1,n):
            print R[i,:]
        else:
            R.delete_rows([i])

A = matrix([[1,0,0],[0,1,0],[1,0,0]])
print A
print "---"
print row_space(A)
```

```
[1 0 0]
[0 1 0]
[1 0 0]
---
簡化列梯陣為:
[1 0 0]
[0 1 0]
[0 0 0]
---
列空間為:
[1 0 0]
[0 1 0]
None
```

In [2]:

```
#Column Space
def col_space(matrix):
    m,n=B.dimensions()
    BB=copy(B)
    BB=BB.change_ring(QQ)
    print "轉置矩陣為:"
    C=BB.transpose()
    print C
    D=C.echelon_form()
    print "經由列變換後得出:"
    print D
    print "---"
    print "行空間為:"
    for i in range(n):
        if D[i,:]!=zero_matrix(1,m):
            print D[i,:]
        else:
            D.delete_rows([i])

B = matrix([[1,0,0],[0,1,0],[1,0,0]])
print B
print "---"
print col_space(B)
```

```
[1 0 0]
[0 1 0]
[1 0 0]
---
轉置矩陣為:
[1 0 1]
[0 1 0]
[0 0 0]
經由列變換後得出:
[1 0 1]
[0 1 0]
[0 0 0]
---
行空間為:
[1 0 1]
[0 1 0]
None
```

Example

In [3]:

```
#example
print "Example 1"
A=B=matrix(3,range(9))
print A
print "---"
print row_space(A)
print col_space(B)
print "-----------------------------------------------------"
print "Example 2"
A=B=matrix([[1,0,0],[2,0,0],[0,1,0],[3,4,0],[0,0,1]])
print A
print "---"
print row_space(A)
print col_space(B)
print "-----------------------------------------------------"
print "Example 3"
A=B=matrix([[1,2,8],[3,4,18],[0,1,3]])
print A
print "---"
print row_space(A)
print col_space(B)
```

```
Example 1
[0 1 2]
[3 4 5]
[6 7 8]
---
簡化列梯陣為:
[ 1  0 -1]
[ 0  1  2]
[ 0  0  0]
---
列空間為:
[ 1  0 -1]
[0 1 2]
None
轉置矩陣為:
[0 3 6]
[1 4 7]
[2 5 8]
經由列變換後得出:
[ 1  0 -1]
[ 0  1  2]
[ 0  0  0]
---
行空間為:
[ 1  0 -1]
[0 1 2]
None
-----------------------------------------------------
Example 2
[1 0 0]
[2 0 0]
[0 1 0]
[3 4 0]
[0 0 1]
---
簡化列梯陣為:
[1 0 0]
```

```
[0 1 0]
[0 0 1]
[0 0 0]
[0 0 0]
---
列空間為:
[1 0 0]
[0 1 0]
[0 0 1]
None
轉置矩陣為:
[1 2 0 3 0]
[0 0 1 4 0]
[0 0 0 0 1]
經由列變換後得出:
[1 2 0 3 0]
[0 0 1 4 0]
[0 0 0 0 1]
---
行空間為:
[1 2 0 3 0]
[0 0 1 4 0]
[0 0 0 0 1]
None
--------------------------------------------------
Example 3
[ 1  2  8]
[ 3  4 18]
[ 0  1  3]
---
簡化列梯陣為:
[1 0 2]
[0 1 3]
[0 0 0]
---
列空間為:
[1 0 2]
[0 1 3]
None
轉置矩陣為:
[ 1  3  0]
[ 2  4  1]
[ 8 18  3]
經由列變換後得出:
[   1    0  3/2]
[   0    1 -1/2]
[   0    0    0]
---
行空間為:
[  1   0 3/2]
[   0    1 -1/2]
None
```

Vector space

Overview

Find representation

$P_n := \{\text{all polynomials with degree at most } n\}$ and n 為正整數

Let $B = \{1, x-1, (x-1)^2\}$ be a basis of P_2 then B has $n+1$ polynomials.

Record the coefficients of each polynomial in B as the columns of a matrix A , such that

$$A = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}.$$

Thus A is an 3×3 $((n+1) \times (n+1))$ matrix.

If $f = x^2$, then the goal is to find

$$Rep_B(f).$$

Algorithm

1. If $f = x^2$, record the coefficients of $f = x^2$ --> $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
2. Suppose $Rep_B(f) = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ such that $<=> a \times 1 + b \times (x-1) + c \times (x-1)^2$
3. We know $A = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$,and $\begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$. Then find A^{-1}
4. Hence , we get $Rep_B(f) = \begin{bmatrix} a \\ b \\ c \end{bmatrix} = A^{-1}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Explanation

目標是尋找 $Rep_B(f)$, 要找尋f轉換在B基底底下的座標(表示法)

First we change B=$\{1, (x-1), (x-1)^2\}$ to another basis matrix form $[1] = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, $[-1+x] = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$, and $[1-2x+x^2] = \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix}$

Get $A = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$; And $f = x^2$, do the same change to make f the same form as B did.

記錄f 的係數 , 得 $f = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$. 再來假設 $Rep_B(x^2) = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$

代表 $a \times 1 + b \times (x-1) + c \times (x-1)^2 = x^2$ 又可表示成下面形式

$$\begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

則可得出 $\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

得解 $Rep_B(f) = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

Implementation

In [2]:

```
def get_coefficient(f, n):
    """

    Input:
        f: polynomial
        n: degree
    Output:
        a list of (n+1) coefficients of f,
        (treating f as a polynomial of degree n)
    """
    l = f.coefficients(sparse=False)
    coeffs = [0] * (n+1)
    for i in range(len(l)):
        coeffs[i] = l[i]
    return coeffs
```

In [3]:

```
def representation(basis, poly):
    """
    Input:
        basis: a list of vectors [B0, ... , Bk] ( a basis of all polynomials of degree at most k)
        poly: a polynomial
    Output:
        the representation of poly with respect to basis
    """
    k = len(basis) - 1
    coeffs_list = [get_coefficient(basis[i], k) for i in range(k+1)]
    A = matrix(coeffs_list).transpose()

    v = matrix(k+1, get_coefficient(poly,k))

    print "representation(basis, polynomial) = "
    show(A.inverse()*v)
```

In [4]:

```
### sample code範例程式
## Let basis
B1=1+0*x
B2=(x-1)
B3=(x-1)^2
f = x^2
B={B1,B2,B3}
f_poly = f.polynomial(QQ)

B_1 = B1.polynomial(QQ)
B_2 = B2.polynomial(QQ)
B_3 = B3.polynomial(QQ)

### construct the matrix

A = matrix([get_coefficient(B_1,2),get_coefficient(B_2,2),get_coefficient(B_3,2)]).transpose()
v = matrix(3,get_coefficient(f_poly,2))

print "representation(basis, polynomial) = "
show(A.inverse()*v)
```

```
representation(basis, polynomial) =
```

$$\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

Examples

In [6]:

```
B1=1+0*x
B2=6*x
B3=2*x^2
B4=1+x^3
B5=x^4
B6=x^5
B={B1,B2,B3,B4,B5,B6}
B_1 = B1.polynomial(QQ)
B_2 = B2.polynomial(QQ)
B_3 = B3.polynomial(QQ)
B_4 = B4.polynomial(QQ)
B_5 = B5.polynomial(QQ)
B_6 = B6.polynomial(QQ)

basis = [B_1, B_2, B_3, B_4, B_5, B_6]

representation(basis, (3*x^5+7*x^4+5*x^3-4*x^2+3*x+7).polynomial(QQ))
```

representation(basis, polynomial) =

$$\begin{pmatrix} 2 \\ \frac{1}{2} \\ -2 \\ 5 \\ 7 \\ 3 \end{pmatrix}$$

In [7]:

```
B1=1+0*x
B2=x
B3=x^2
B4=1+x^3
B5=x^4
B6=2*x^5
B7=(x+1)^6
B8=(x-1)^7
B9=x^8
B10=(x-2)^9
B={B1,B2,B3,B4,B5,B6,B7,B8,B9,B10}
B_1 = B1.polynomial(QQ)
B_2 = B2.polynomial(QQ)
B_3 = B3.polynomial(QQ)
B_4 = B4.polynomial(QQ)
B_5 = B5.polynomial(QQ)
B_6 = B6.polynomial(QQ)
B_7 = B7.polynomial(QQ)
B_8 = B8.polynomial(QQ)
B_9 = B9.polynomial(QQ)
B_10 = B10.polynomial(QQ)
basis = [B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_10]

representation(basis, (x^8+3*(x-5)^7+x^6+2*x^5-6*x^4+3*(x+5)^3).polynomial(QQ))
```

representation(basis, polynomial) =

$$\begin{pmatrix} -301097 \\ 328827 \\ -195522 \\ 67183 \\ -11781 \\ 1006 \\ -83 \\ 3 \\ 1 \\ 0 \end{pmatrix}$$

In [ ]:

```

```