Python講習会：Python使ってやったこと（やりたいこと）、研究やら、教育やら、遊びやら

2014.12.15 於：海洋大学　　東邦大学理学部情報科学科 並木誠

環境と主なモジュール：Mac OS 10.9.5, Python2.7.6, VPython 6.10, numpy 1.8.0, matplotlib 1.1.1, ipython 2.2.0, PuLP 1.5.6, networkx 1.9.1, , sympy 0.7.5

内容

研究やら：

- 何はともあれ線形計画
  - PuLP を使って問いてみる.
  - VPythonを用いた可視化.
- 線形計画を使ったP行列判別
  - P行列判別
  - P行列とLCP (principal povoting method）
  - 隠れ行列によるとP行列サンドイッチ（LPを用いた隠れ行列の判別）

教育やら：

- 卒論指導（オペレーションズ・リサーチ）
  - 包絡分析法（DEA，線形計画）
  - オーボエの運指最適化（2013年度卒論，最短路問題）
  - 整数計画を用いたなでしこリーグの対戦日程計画（2012年度卒論，0,1整数計画）
- 授業での実習課題（Mathematica の代替となりうるか？）
  - 星形を描く（簡単な線図形，VPython）
  - 接線のアニメーション（数式処理，VPython）
  - 三角形の内接円，外接円（数式処理，VPython）
  - 987654321 のすべての約数
  - 虫食い算（in consideration）

遊びやら：（SSHや出張授業）

- 正多面体，半正多面体，半正多面体の双対をきれいに描く
- 任意の有界な3次元凸多面体を描く
- 多面体のスケルトングラフの全域木を描く（in preparation）.
- 多面体の展開図を描く（in consideration）
- Zometoolで遊ぶ

何はともあれ線形計画

PuLPを使って解いてみる

本講演で必須の数理最適化問題：線形計画問題を解いてみよう．線形計画問題とは，「目的関数が線形関数で，制約条件も線形関数の等式または不等式で表される最適化問題」である．以下のような不等式標準形問題を扱う.

$$
\left|\begin{array}{ll}
\text{最大化} & c_1x_1+c_2x_2+\cdots+c_nx_n \\
\text{条　件} & \left\{\begin{array}{rcl} a_{11}x_1+a_{12}x_2+\cdots+a_{1n}x_n & \leq & b_1 \\ & \vdots & \\ a_{m1}x_1+a_{m2}x_2+\cdots+a_{mn}x_n & \leq & b_m \end{array}\right. \\
 & x_1,x_2,\ldots,x_n \geq 0
\end{array}\right.
$$

上の問題は，行列とベクトルを使って次のようにも表せる.

$$
\left|\begin{array}{ll}
\text{最大化} & c^\top x \\
\text{条　件} & Ax \leq b,\ x \geq 0
\end{array}\right.
$$

行列*A*，コストベクトル*c*，右側ベクトル*b*を次のように具体的に決めてpythonで解いてみる．使うモジュールは numpy と pulp．まずは pulp読み込みとテスト．さらにコストベクトル*c*，行列*A*，右側ベクトル*b*を引数としてLPを解く関数 LPStandardForm() を定義している.

In [ ]:
```
from pulp import *
pulp.pulpTestAll() # pulpのテストプログラム
```

In [1]:
```
from pulp import *
from numpy import *

def LPStandardForm(c,A,b):
    m, n = A.shape

    prob = LpProblem('sample',LpMaximize) #最大化のモデル'sample'を用意

    # LpVariableを変数分用意しリストにする.
    x = [LpVariable('x'+str(j), lowBound=0, cat='Continuous') for j in range(n)]

    #目的関数を設定
    prob += lpDot(c,x)

    #制約条件を設定
    for i in range(m):
        prob += lpDot(A[i],x) <= b[i], 'lneq'+str(i)

    #モデルを表示してみて，さらに解く
    print prob
    prob.solve()

    print LpStatus[prob.status]
    for  v in prob.variables():
        print v.name,"=",v.varValue
    print "opt. value =",value(prob.objective)
```

In []:
```
# 問題を決めるベクトルや行列を用意.
c = array([2,1])
A = array([[1,2],[1,1],[3,1]])
b = array([10,6,12])

LPStandardForm(c,A,b)
```

解けない問題も解いてみる.

最適解が存在しない問題を解いてみる（実行不可能な問題）.

In []:
```
c = array([1,1])
A = array([[1,-1],[-1,1]])
b = array([-1,-1])

LPStandardForm(c,A,b)
```

最適解が存在しない問題を解いてみる（非有界な問題）.

In []:
```
c = array([1,1])
A = array([[-2,1],[1,-2]])
b = array([2,2])

LPStandardForm(c,A,b)
```

VPythonを用いた可視化.

VPython とは？ 主に3Dグラフィックスやアニメーションの扱いに長けたモジュール.

ちなみに VPython のホームページはこちら.

http://vpython.org

VPythonのサイトでも手に入る非常に簡単な3Dサンプルプログラム

In []:
```
from visual import *

D = display() #display を用意．省略すると scene という displayが用意される.
D.width =600; D.height = 600 #幅と高さ
D.center = vector(1,1,1) #点(1,1,1)がdisplayの中心
D.range =12 #最初に display に表示される範囲
D.up = vector(0,1,0) #上方向のベクトル
D.forward = vector(-1,-1,-1) #画面奥の方向は (-1,-1,-1)
D.background = color.white #
D.title = "Bouncing" #窓のタイトル

floor = box(length=4, height=0.5, width=4, color=color.blue)

ball = sphere(pos=(0,10,0), radius=0.5, color=color.red)
ball.velocity = vector(0,-1,0)

dt = 0.05
while True:
    rate(50) # 計算割合．1秒に100回の命令を実行.
    ball.pos = ball.pos + ball.velocity*dt
    if ball.y < 1:
        ball.velocity.y = -ball.velocity.y
    else:
        ball.velocity.y = ball.velocity.y - 9.8*dt
```

先ほどの2変数のLPの例：

MAXIMIZE 2x0 + 1x1 + 0
SUBJECT TO
lneq0: x0 + 2 x1 <= 10
lneq1: x0 + x1 <= 6
lneq2: 3 x0 + x1 <= 12
VARIABLES
x0 Continuous
x1 Continuous

について，実行可能領域と最適解を描いてみる．2次元であるが，VPython の３次元グラフィックスとして描く．最初にあるdisplayに関する細々とした設定は，モジュール化して myGraphicModules.py (myGraphicModules.py) に入れておき，必要なときに読み込むようにする.

In [ ]:
```
from visual import *
from myGraphicModules import *

D = display()
myXYAxes(D, '2DLPExample', True, -1, 8, -1, 8, 50, 50)


l1 = curve(pos= [(x, -1/2.0*x+5, 0) for x in arange(-1.0,7.5, 0.1)], color=color.red,radius=0.01)
l2 = curve(pos= [(x, -x+6, 0) for x in arange(-0.5,6, 0.1)], color=color.red,radius=0.01)
l3 = curve(pos= [(x, -3.0*x+12, 0) for x in arange(1,4.5, 0.1)], color=color.red,radius=0.01)
l0 = curve(pos= [(x, -2.0*x+9, 0) for x in arange(1,4.5, 0.1)], color=color.red,radius=0.03)
points = [(0,0,0),(0,5,0),(2,4,0),(3,3,0),(4,0,0)]
poly = convex(pos=points,color=color.blue)
optimal = sphere(pos=(3,3,0), color=color.yellow,radius=0.2)
optlabel = label(pos=optimal.pos,text='Optimal', xoffset=20,
      yoffset=12, space=optimal.radius,height=24, border=2,background=color.black)

while 1:
  rate(10)
```

3変数のLPについてもやってみる.

In [ ]:
```
from numpy import *
from pulp import *
from myLP import LPStandardForm

c = array([2,3,2])
A = array([[1,1,2],[3,1,0],[0,2,1]])
b = array([24,16,12])

LPStandardForm(c,A,b)
```

In [ ]:
```
from visual import *
from myGraphicModules import *

D = display()
myXYZAxes(D,'LPExample', -1, 8, -1, 8, -1, 12, 50, 50)

points = [(0,0,0),(16.0/3,0,0),(16.0/3,0,28.0/3), \
       (0,6,0),(10.0/3,6,0),(24.0/5,8.0/5,44.0/5),(0,0,12)]
poly = convex(pos=points,color=color.green)
optimal = sphere(pos=(24.0/5,8.0/5,44.0/5), color=color.red,radius=0.3)
optlabel = label(pos=optimal.pos,text='Optimal', xoffset=20,
      yoffset=12, space=optimal.radius,height=24, border=2,background=color.black)

while 1:
  rate(10)
```

線形計画を使ったP行列判別

P行列の定義:

$$n \text{ 次正方行列 } A \text{ が P 行列である} \iff \det(A_{II}) > 0 \text{ for all } I \subseteq \{1,2,\cdots,n\}$$

（行集合と列集合が同じである部分正方行列の行列式が全て正）

例えば

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix} \text{ は P 行列であるが, } A' = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix} \text{ は P 行列ではない.}$$

なぜならば

$$I = \{1,3\} \text{ とすれば } \det(A_{II} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}) \leq 0$$

だからである.

以下は, この定義による P行列判別の関数：pmatrixQ(matrix)の定義である.

In [ ]:
```
import itertools
from numpy import *
MEPS=1.0e-8

def all_nonempty_subsets(ls):
  return [list(x) for i in range(1,len(ls)+1)
       for x in itertools.combinations(ls,i)]

def pmatrixQ(matrix):
  """
  returns True if the given matrix is a P-matrix o.w. False.
  """
  n = len(matrix)
  indexsets = all_nonempty_subsets(range(n))
  for I in indexsets:
    if linalg.det(matrix[ix_(I,I)])<=MEPS:
      return False
  return True
```

In [ ]:
```
A1 = array([[1,2,0],[0,1,2],[2,0,1]])
A2 = array([[1,2,2],[0,1,2],[2,0,1]])
print pmatrixQ(A1)
print pmatrixQ(A2)
```

上の判定プログラムは指数オーダーの計算量なので, 当然のことながら \(n\) が大きいと終わらない. 乱数で生成した行列に対してやってみる.

In [ ]:
```
from prdd import *

A = random_kprddout(19,2)

%time f = pmatrixQ(A)
print f
```

違うアプローチが必要である．その前に，

Q. なぜP行列を考えるのか？　　A. 線形相補性問題 と深く関係するから．

線形相補性問題

$$M \in \mathbb{R}^{n\times n},\ q \in \mathbb{R}^n \text{ とする．}$$

*M*と*q*によって定義される線形相補性問題：LCP(*M,q*)とは以下のような 問題である．

$$\left|\begin{array}{ll} \text{find} & x, y \in \mathbb{R}^n \\ \text{s. t.} & y = Mx + q, \qquad y, x \geq 0 \\ & x_i \cdot y_i = 0 \quad \text{for all } i = 1, 2, \ldots, n \end{array}\right.$$

線形計画問題や凸2次計画問題は，それらの双対問題を同時に解くと考えると，線形相補性問題に帰着できる．線形相補性問題とP行列に関して、次の性質が成り立つ．

$$M \in \mathbb{R}^{n\times n} \text{ が P 行列である} \iff \text{任意の } q \in \mathbb{R}^n \text{ に対して LCP}(M,q) \text{ が唯一解を持つ．}$$

ちなみに，効率の保証はないが，線形方程式を解くプロシージャーさえあれば， P行列によって定義されるLCPを解くプログラムを簡単に作ることは可能である． 以下は，P行列LCPを解くための Least Index Principal Pivoting Method（Murty) の簡単な実装である．

In [ ]:
```
from numpy import *

def PrincipalPivotingMethod(M,q):
    n = len(M)
    A = c_[-M,identity(n)]  # 行列の列ごとの連結，行ごとの連結は r_[a,b]
    nbas = range(n)
    bas = range(n,2*n)
    R = range(n)

    while 1:
        Ab = A[ix_(R,bas)]
        An = A[ix_(R,nbas)]
        bq = linalg.solve(Ab,q)

        flag = 0
        for i in R:
            if bq[i]<0:
                flag = 1
                break
        if flag == 1:
            bas[i],nbas[i] = nbas[i],bas[i]
        else:
            x = zeros(2*n)
            x[ix_(bas)] = bq
            return x[ix_(range(n))]
```

In [ ]:
```
M = array([[1,2,0],
           [0,1,2],
           [2,0,1]])
q = array([-1,1,-1])

x = PrincipalPivotingMethod(M,q)

print x
print dot(M,x)+q
```

In [ ]:
```
from numpy import *
from prdd import *

A = random_kprddout(500,16)
b = ones(len(A))
b = -b

x = PrincipalPivotingMethod(A,b)
print x
print dot(x,dot(A,x)+b)
```

prdd行列

話をP行列に戻す．P行列であるかどうかの判定は難しいので（co-NP完全問題）P行列クラスの部分クラス*L*とP行列クラスのスーパークラス*U*を見つける．これをP行列サンドイッチと言うことにしよう．

$$\text{部分クラス } L \subseteq \text{P 行列クラス} \subseteq \text{スーパークラス } U$$

特に部分クラス*L*やスーパークラス*U*に **隠れ行列クラス**を用いる．

隠れ行列の定義：☆をある行列のクラスとすると，隠れ☆行列は次のように定義される．

$$A \text{ は隠れ☆行列である} \iff \exists B, C \in \text{☆ s. t. } AC = B$$

ベースとなる行列クラス☆として，次の二つを考える．

$$\text{prdd-1 行列:}\quad C \in \mathbb{R}^{n\times n} \text{ が prdd-1 行列である} \iff C_{ii} > \sum_{j\neq i} |C_{ij}|\ \forall i$$

$$\text{prdd-}\infty \text{ 行列:}\quad C \in \mathbb{R}^{n\times n} \text{ が prdd-}\infty \text{ 行列である} \iff C_{ii} > |C_{ij}|\ \forall i \neq j$$

例えば，*n=3* の場合の例でCの1番目の行ベクトル *(C11,C12,C13)=(x,y,z)*が prdd-1とprdd-∞を満たす領域を図示してみると，prdd-1が内側の緑の多面錐，prdd-∞が外側の青の多面錐のそれぞれ内部となる．ソースコードはこちら(prdd.py) (prdd.py)

In [ ]:
```
from prdd import view_prddcones

view_prddcones()
```

隠れprdd行列によるP行列サンドイッチ

次のprdd-1行列クラスとprdd-∞行列クラスを使った隠れ行列を考える.

隠れprdd-1行列:

$A \in \mathbb{R}^{n\times n}$ が隠れ prdd-1 行列である $\iff$ $AC = B$ を満たす prdd-1 行列 C, B が存在する.

隠れprdd-∞行列:

$A \in \mathbb{R}^{n\times n}$ が隠れ prdd–∞ 行列である $\iff$ $AC = B$ を満たす prdd–∞ 行列 C, B が存在する.

サンドイッチ定理：

$$\text{隠れ prdd-1 行列クラス} \subseteq \text{P 行列クラス} \subseteq \text{隠れ prdd-}\infty \text{ 行列クラス}$$

が成り立つ.

さらに，以下の二者択一の定理が成り立つ.

隠れprdd-1行列に関する二者択一の定理：

任意の正方行列*A*に対し以下のいずれか一方かつ一方のみが成り立つ.

(i) $AC = B$ を満たす prdd-1 行列 C, B が存在する.
(ii) 以下を満たす２つの正方行列 $(R, S) \neq (0, 0)$ が存在する.

$$R + A^\top S = 0, R_{ii} \geq |R_{ij}| \text{ for all } i, j(i \neq j) \text{ and } S_{ii} \geq |S_{ij}| \text{ for all } i, j(i \neq j)$$

隠れprdd-\(\infty\)行列に関する二者択一の定理：

任意の正方行列*A*に対し以下のいずれか一方かつ一方のみが成り立つ.

(i) $AC = B$ を満たす prdd-∞ 行列 C, B が存在する.
(ii) 以下を満たす２つの正方行列 $(R, S) \neq (0, 0)$ が存在する.

$$R + A^\top S = 0, R_{ii} \geq \sum_{i\neq j} |R_{ij}| \text{ for all } i \text{ and } S_{ii} \geq \sum_{i\neq j} |S_{ij}| \text{ for all } i$$

上の２つの定理は「与えられた正方行列 *A* が 隠れprdd-1行列（または隠れprdd-∞行列）であるかどうかの判別は，線形計画問題を 解くことで可能である．以下はpulpを使った隠れprdd行列の判別プログラム(prdd.py) (prdd.py)の実行例である.

In [ ]:
```
from numpy import *
from pulp import *
from prdd import *

A1 = array([[1,2,0], [0,1,2], [2,0,1]])
A2 = array([[1,2,2], [0,1,2], [2,0,1]])
A3 = array([[0,0,1,1], [0,0,1,1], [-1,-1,0,0], [-1,-1,0,0]])

f,m11,m12= hidden_prdd1Q(A1)
print f, m11,m12
f,m21,m22= hidden_prdd1Q(A2)
print f, m21,m22
f,m31,m32 = hidden_prddinfQ(A2)
print f, m31,m32
f,m41,m42 = hidden_prddinfQ(A3)
print f, m41,m42
```

卒論指導

包絡分析法，Data Envelopment Analysis (DEA)

事業体などの評価に使われる手法（線形計画問題を使って分析される）.

例題

例えば，\(0\)から\(9\)までの事業体（DEAでは特に，DMU Decision Making Unitという）が，入力1（例えば資本金），入力２（例えば売り場面積），入力３（例えば従業員数）をもとに，出力1（例えば売り上げ），出力２（例えば株価）を得ているとしよう.

DMU	入力１	入力２	入力３	出力１	出力２
0	90.5	57.9	68.5	85.2	92.1
1	65.5	56.0	63.5	54.5	50.5
2	92.0	63.0	65.5	52.0	99.5
3	52.0	98.5	98.5	80.9	54.5
4	74.5	51.0	81.5	72.0	84.0
5	78.0	69.5	69.9	64.0	91.5
6	72.5	71.0	93.5	73.5	97.5
7	54.5	70.5	85.0	84.0	94.5
8	56.5	53.5	64.5	96.0	81.0
9	80.5	94.5	77.5	57.5	56.0

DEA（包絡分析法各）では，各DMUについて

$$\frac{\text{仮想出力}}{\text{仮想入力}} = \frac{u_1\times\text{出力 1}+u_2\times\text{出力 2}}{v_1\times\text{入力 1}+v_2\times\text{入力 2}+v_3\times\text{入力 3}}$$

で相対評価する．これを効率値という．

\(u_1,u_2, v_1, v_2, v_3\) は出力及び入力に対する重みで， 各DMUが自身の都合のよいように（評価が最大になるように）決めてよい．

例えば，DMU0が自身の都合のよいように重みを決める問題は，次のような分数計画問題になる．

$$
\begin{aligned}
\text{最大化} \quad & \frac{85.2u_1+92.1u_2}{90.5v_1+57.9v_2+68.5v_3} \\
\text{DMU0 の条件:} \quad & \frac{85.2u_1+92.1u_2}{90.5v_1+57.9v_2+68.5v_3} \leq 1 \\
\text{DMU1 の条件:} \quad & \frac{54.5u_1+50.5u_2}{65.5v_1+56.0v_2+63.5v_3} \leq 1 \\
\text{DMU2 の条件:} \quad & \frac{52.0u_1+99.5u_2}{92.0v_1+63.0v_2+65.5v_3} \leq 1 \\
& \vdots \\
\text{DMU9 の条件:} \quad & \frac{57.5u_1+56.0u_2}{80.5v_1+94.5v_2+77.5v_3} \leq 1 \\
& u_1, u_2, v_1, v_2, v_3 \geq 0
\end{aligned}
$$

上の分数計画問題は，下の線形計画問題に帰着される．

$$
\begin{aligned}
\text{最大化} \quad & 85.2u_1 + 92.1u_2 \\
\text{DMU0 の入力の条件:} \quad & 90.5v_1 + 57.9v_2 + 68.5v_3 = 1 \\
\text{DMU0 の条件:} \quad & 85.2u_1 + 92.1u_2 \leq 90.5v_1 + 57.9v_2 + 68.5v_3 \\
\text{DMU1 の条件:} \quad & 54.5u_1 + 50.5u_2 \leq 65.5v_1 + 56.0v_2 + 63.5v_3 \\
\text{DMU2 の条件:} \quad & 52.0u_1 + 99.5u_2 \leq 92.0v_1 + 63.0v_2 + 65.5v_3 \\
& \vdots \\
\text{DMU9 の条件:} \quad & 57.5u_1 + 56.0u_2 \leq 80.5v_1 + 94.5v_2 + 77.5v_3 \\
& u_1, u_2, v_1, v_2, v_3 \geq 0
\end{aligned}
$$

numpy と pulp モジュールを使って解いてみる．

In [ ]:
```
from numpy import *
from pulp import *

#各DMUのinput, output  をIN, OUTに入力
IN = array([[90.5,57.9,68.5], [65.5,56.0,63.5],  [92.0,63.0,65.5],
         [52.0,98.5,98.5],  [74.5,51.0,81.5], [78.0,69.5,69.9],
         [72.5,71.0,93.5],  [54.5,70.5,85.0], [56.5,53.5,64.5],
         [80.5,94.5,77.5]])
OUT = array([[85.2,92.1], [54.5,50.5],  [52.0,99.5],  [80.9,54.5],
          [72.0,84.0],  [64.0,91.5], [73.5,97.5],  [84.0,94.5],
          [96.0,81.0], [57.5,56.0]])
#DMUの個数をp，input の個数をm, output の個数を n とする．
p, m = IN.shape
p, n = OUT.shape

prob = LpProblem('DEASample',LpMaximize) #最大化のモデル'DEAample'を用意
# 変数を入力と出力の個数分，辞書として用意
v = {j: LpVariable('v'+str(j), lowBound=0, cat='Continuous') for j in range(m)}
u={j: LpVariable('u'+str(j), lowBound=0, cat='Continuous') for j in range(n)}

#どのDMUについて評価するかをd で表す

d = 0

#目的関数を設定
prob +=lpSum(OUT[d,j]*u[j] for j in range(n))

#DMU0の入力=1の条件
prob += lpSum(IN[d,i]*v[i] for i in range(m)) == 1, 'DMU'+str(d)+':Input=1'

#それぞれのDMUについて，出力＜＝入力の条件
for k in range(p):
   prob += lpSum(OUT[k,j]*u[j]  for j in range(n)) <= lpSum(IN[k,i]*v[i]  for i in range
(m)), 'DMU'+str(k)

   #モデルを表示して，さらに解く
print prob
prob.solve()

print LpStatus[prob.status],':'
for  v in prob.variables():
   print v.name,"=",v.varValue
print "opt. value =",value(prob.objective)
```

すべてのDMUについて効率値を求める関数 DEA(IN,OUT)を定義．

In [2]:
```
from numpy import *
from pulp import *

def DEA(IN,OUT):
    """
    returns evaluation values for all DMUs.
    """
    p, m = IN.shape
    p, n = OUT.shape

    ev = []
    for s in range(p):
        prob = LpProblem('DEASample'+str(s),LpMaximize)
        v = {j: LpVariable('v'+str(j), lowBound=0, cat='Continuous') for j in range(m)}
        u={j: LpVariable('u'+str(j), lowBound=0, cat='Continuous') for j in range(n)}

        prob +=lpSum(OUT[s,j]*u[j] for j in range(n))
        prob += lpSum(IN[s,i]*v[i] for i in range(m)) == 1, 'DMU'+str(s)+':Input=1'

        for k in range(p):
            prob += lpSum(OUT[k,j]*u[j]  for j in range(n)) <= lpSum(IN[k,i]*v[i]  for i in r
ange(m)), 'DMU'+str(k)

        prob.solve()
        ev.append(value(prob.objective))

    return ev
```

In []:
```
ev = DEA(IN,OUT)
[round(i,2) for i in ev]
```

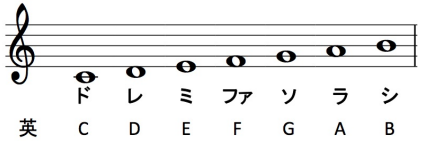
オーボエの運指最適化（東邦大学理学部情報科学科　2013年度卒業論文　長谷川涼）

はじめに

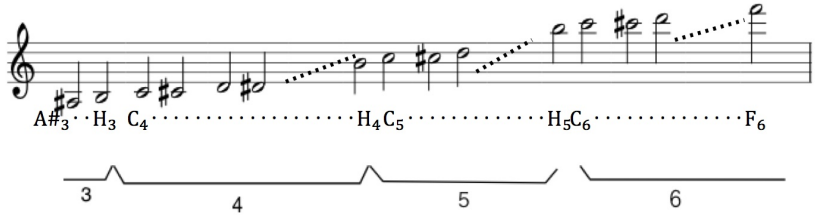
- オーボエという楽器と音色　https://www.youtube.com/watch?v=0lOcJzwAKEA
- オーボエという楽器は、1つの音に対して複数通りの運指(指使い)が存在する場合がある.
- 複数通りの運指を使い分けることで・・・
  ① 音色や音質を調節する ② テンポの速いフレーズ(音の並び)を楽に演奏する
- 「オーボエの演奏におけるなめらかな指使い」を最適化の手法を用いて導くことを考える．具体的には，楽譜上での音に対するそれぞれの指使いを有向グラフの点と考え，有向枝をある指使いからある指使いへの移動と考え，枝の重みは指の運びのコストと考える．これをフレーズ（曲）の運指ネットワークと呼び，運指ネットワーク上で最短路問題を解く.

音について

- 音名はアルファベット表記とする.



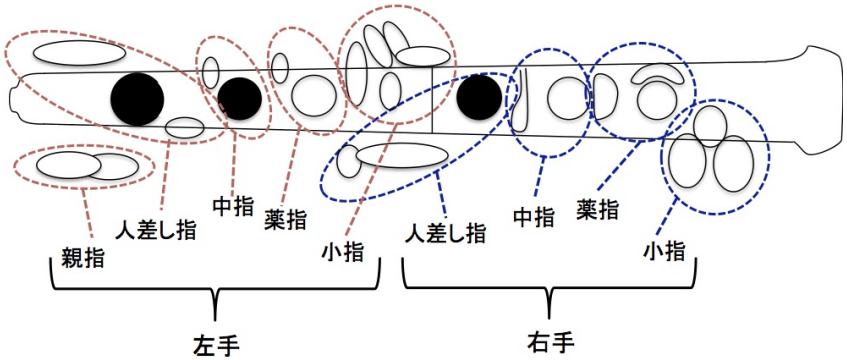
- ピアノの真ん中のCから始まるオクターブを4とし「C4 」で表す．オクターブがあがるごとに数字が増える.



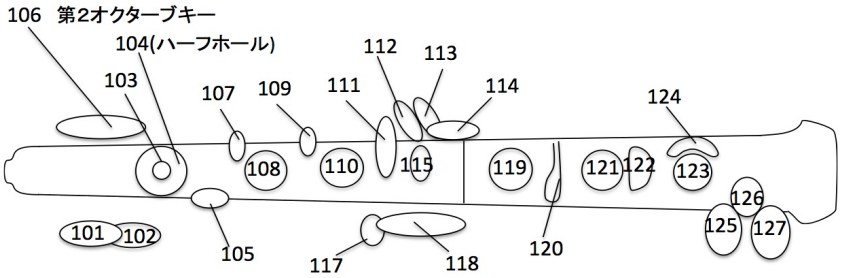
- 今回扱う音は，A#3～F6の32音.

オーボエの運指

- オーボエという楽器の構成.



- **運指ベクトル：** 指がどのキーを押しているかをベクトルで表したもの． 運指ベクトルファイル: Unshi.txt　(fig/Unshi.txt) 値が 0,1(離す，押さえる）でないのは，押さえる，離す，移動が成分の差分ですぐわかるようにするため．運指ベクトルの数値とオーボエのキーの割当は以下の通り.



- **基本運指：** 初心者が最初に覚える運指であり，その音を最も安定した音色・音程で吹くことができる運指．各音に必ず1つある．基本運指はその音の運指の一番最初のもの．ちなみに音による運指の種類数は以下の通り.

音域	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
3											1	1
4	1	1	2	2	3	5	3	3	2	3	1	4
5	5	4	3	2	2	4	3	3	3	2	1	3
6	2	2	2	4	4	5						

表 1　各音に対する運指の種類数

- 次のプログラムは，運指のデータを辞書として読み込む関数定義．soundname に音の名前を，unshiは，音の名前をキーとした運指ベクトルのリストからなる辞書．

In [1]:
```
from numpy import *

def read_unshi_data(file_name):
    f = open(file_name, 'r')
    lines = []
    for line in f:
        lines.append(line)
    f.close()
    lines.pop(0)

    soundname = []
    unshi = {}
    for line in lines:
        ll = line.split()
        s = ll.pop(0)
        ll = [int(x) for x in ll]
        if s not in soundname:
            soundname.append(s)
            unshi[s] = [ll]
        else:
            unshi[s].append(ll)
    return (soundname, unshi)

soundname, unshi = read_unshi_data('fig/Unshi.txt')
```

In []:
```
print soundname
print unshi
```

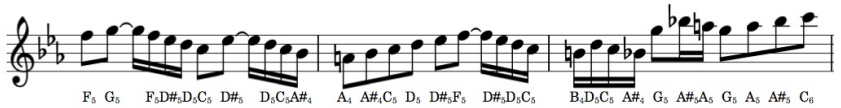
運指間距離の定義

新たに指で押さえる場合, 0.15, 離す場合 0.1, ずらす場合 0.75としてみる．運指間距離は，すべての指に関してその値の和をとったもの．

In [3]:
```
def unshi_dist(a,b):
    OSAERU = 0.15; HANASU = 0.1; ZURASU = 0.75
    d = array(b) - array(a); dist = 0
    for f in d:
        if f > 100:
            dist += OSAERU
        elif f < -100:
            dist += HANASU
        elif abs(f) > 0 and abs(f) < 10:
            dist += ZURASU
    return dist
```

運指ネットワークの作成

- 最適化するのは，ワーグナー作曲「ニュルンベルクのマイスタージンガー」のあるフレーズ．楽譜は次の通り．



- 次のプログラムでそのフレーズに対する運指ネットワークを，NetworkXモジュールを利用して作成する．

In [4]:
```
import networkx as nx

# フレーズの入力
phraze = ['F5','G5','F5','D#5','D5','C5','D#5','D5','C5','A#4','A4',\
        'A#4','C5','D5','D#5','F5','D#5','D5','C5','B4','D5','C5','A#4',\
        'G5','A#5','A5','G5','A5','A#5','C6']
n = len(phraze)
ul = [len(unshi[phraze[i]]) for i in range(n)]

# 有向グラフを作る
DG=nx.DiGraph()

node_labels = {}; edge_labels = {}
#ダミーノード:StartとEndとそのラベル
DG.add_node('Start', pos=(-5,2)); DG.add_node('End', pos=(n+6,2))
node_labels['Start'] = 'Start'; node_labels['End'] = 'End'

#各音の各運指に対するノードを加える
for i in range(n):
   for j in range(ul[i]):
      DG.add_node((i,j), pos=(i+1,j))
      node_labels[(i,j)] = 'Sound'+str(i)+':'+str(j)

#Startから0番目の音の各運指への枝
p = phraze[0]
for j in range(ul[0]):
   DG.add_edge('Start',(0,j),weight=0)
   edge_labels['Start',(0,j)] = '0'

#i番目の音の各運指からi+1番目の各運指への有向枝
for i in range(n-1):
   for j in range(ul[i]):

      for k in range(ul[i+1]):
         DG.add_edge((i,j),(i+1,k),weight=unshi_dist(unshi[phraze[i]][j],unshi[phraze[i
+1]][k]))
         edge_labels[(i,j),(i+1,k)] = str(unshi_dist(unshi[phraze[i]][j],unshi[phraze[i+1
]][k]))

# n-1番目の音の各運指から'End'までの有向枝
q = phraze[n-1]
for j in range(ul[n-1]):
   DG.add_edge((n-1,j),'End',weight=0)
   edge_labels[(n-1,j),'End']  = '0'
```

運指ネットワークの表示

In [5]:
```
import matplotlib.pylab as plt

# Retrieve the positions from graph nodes and save to a dictionary
pos=nx.get_node_attributes(DG,'pos')

#nx.draw_networkx_labels(DG,pos,node_labels,font_size=4,font_family='sans-serif')
nx.draw_networkx_nodes(DG,pos,DG.nodes(), node_size=20, node_color='red')

# draw edges
nx.draw_networkx_edge_labels(DG,pos,edge_labels,font_size=10)

nx.draw_networkx_edges(DG,pos,DG.edges(), width=0.5, edge_color='green')

plt.show()
```

最適化

NetworkXモジュールの dijkstra法を使って最短路を求める.

In [ ]:
```
sp = nx.dijkstra_path(DG, 'Start', 'End')
print sp
splen = nx.dijkstra_path_length(DG, 'Start', 'End')
print splen
```

運指ネットワークに最短路を表示してみる.

In [7]:
```
# Retrieve the positions from graph nodes and save to a dictionary
pos=nx.get_node_attributes(DG,'pos')

#nx.draw_networkx_labels(DG,pos,node_labels,font_size=4,font_family='sans-serif')
nx.draw_networkx_nodes(DG,pos,DG.nodes(), node_size=20, node_color='red')

# make shortest path edges
spedge = [(sp[i], sp[i+1]) for i in range(len(sp)-1)]

# draw edges
nx.draw_networkx_edge_labels(DG,pos,edge_labels,font_size=10)
nx.draw_networkx_edges(DG,pos,DG.edges(), width=0.5, edge_color='green')
nx.draw_networkx_edges(DG,pos,edgelist=spedge, width=2, edge_color='blue')

plt.show()
```

結果の考察

基本運指　それぞれの音には，基本運指という基本的な音の出し方がありそれを並べたもの 教師運指　教則本などに書いてある慣れ親しんだ運指と運指最適化による結果を比較

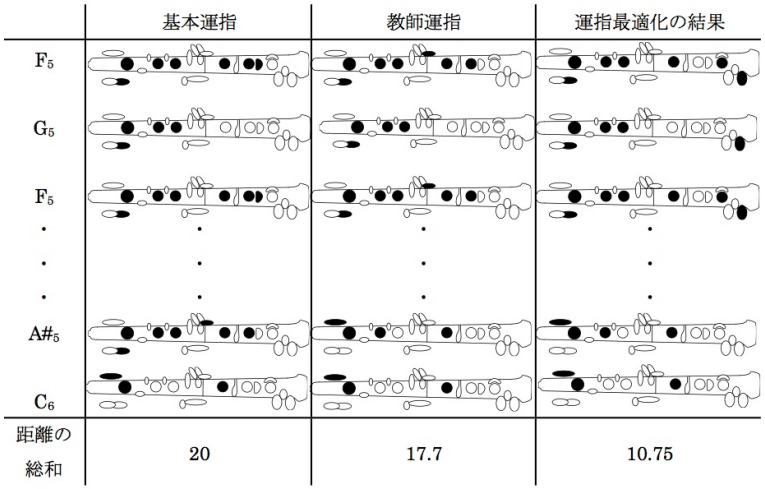


表　基本運指と教師運指と運指最適化の結果

参考文献： オーボエの最適運指　　東邦大学理学部情報科学科　2013年度卒業論文　長谷川涼

整数計画を用いたなでしこリーグの対戦日程計画（東邦大学理学部情報科学科 2012年度卒業論文　伊藤仁勝）

考慮すべき点（2012年度時点）

- 10チームによる二重総当たり：　ホーム & アウェイ 方式を採用．あるチームが他のチームと２回対戦する．１回は本拠地（ホーム，H）もう１回は，相手本拠地（アウェイ，A）．\(n\)チームの場合，各試合日に \(n/2\) 試合ずつ \(2(n-1)\)試合日で終了するのが条件．
- ミラーリング：　下図のように一重総当たり戦のスケジュールを作り，後半を対戦場所を入れ替えた（H A を入れ替えた）スケジュールにする．これをミラーリングといい，なでしこリーグでは採用されている．同じチームとの対戦間隔が十分とれるというメリットがある．

チーム＼試合日	第１節	第２節	第３節	第４節	第５節	第６節
A	B.	C	D.	B	C.	D
B	A	D.	C	A.	D	C.
C	D	A.	B.	D.	A	B
D	C.	B	A	C	B.	A.

- HAパターン：　長さ \((n-1)\)の H（ホーム）と A（アウェイ）からなる列．各チームはこのHAパターンに従い移動すると考える．なでしこリーグでは，以下の条件を満たすHAパターンを採用するものとする（最後の3つの条件で，第8,9,10,11節でのブレイクを避けられる）．
  - ９個の列のうち，Hは５つまたは4つ（Aは４つまたは５つ）とする．
  - 部分文字列 "HH" や "AA"が表れる回数をブレイク数といい，ブレイク数は少ない方がよい．ここでは HH, AA ともに1回ずつを上限とする．
  - 第1節と第2節は異なるHA状態とする．
  - 第1節と第9節を同じHA状態とする（第9節と第10節は異なる）．
  - 第8節と第9節を異なるHA状態にする．
- パターンセット　パターンをチーム数と同じだけ組み合わせたもの．スケジュール可能であるための必要条件は，すべての試合日（パターンセットの列）に出てくるHとAの個数が同じであることである．
- パターンセットのスケジューリング可能性　そのパターンセットに従ったとき，スケジューリング可能かどうかは，以下の例のように整数計画を解くことでチェックできる．

例：チーム数が4で次のパターンセットの場合を考える

チーム＼試合日	1	2	3
A	H	H	A
B	A	A	H
C	A	H	H
D	H	A	A

0,1 変数 x_{ijk}（$i,j \in \{\mathrm{A,B,C,D}\}, i \neq j,\ k = 1,2,3$ ）を用意し，

$$x_{ijk} = \begin{cases} 1 & \text{チーム } i \text{ とチーム } j \text{ が第 } k \text{ 節で試合をする} \\ 0 & \text{チーム } i \text{ とチーム } j \text{ が第 } k \text{ 節で試合をしない} \end{cases}$$

と考える．次の 0,1整数計画問題が最適解（すべての実行可能解が最適解）を持てば，そのパターンセットでスケジューリングが可能であることがわかる.

$$
\begin{array}{ll}
\text{最大化} & 0 \\
\text{条　件} & \left\{\begin{array}{ll}
\sum_{k=1}^{3} x_{ijk} = 1 & \forall i,j \in \{\mathrm{A,B,C,D}\}, i \neq j \\
 & (i \text{ と } j \text{ が } 1 \text{ 度だけ対戦する}) \\
x_{AB1} + x_{AC1} + x_{AD1} = 1 & \mathrm{A} \text{ の第 } 1 \text{ 節での対戦相手は } 1 \text{ チームだけ} \\
x_{AB1} + x_{BC1} + x_{BD1} = 1 & \mathrm{B} \text{ の第 } 1 \text{ 節での対戦相手は } 1 \text{ チームだけ} \\
\vdots & \\
x_{AD3} + x_{BD3} + x_{CD3} = 1 & \mathrm{D} \text{ の第 } 3 \text{ 節での対戦相手は } 1 \text{ チームだけ} \\
x_{AD1} = 0,\ x_{BC1} = 0, \cdots, x_{BC3} = 0 & \text{各節で，HA パターンが同じチームは対戦できない.}
\end{array}\right. \\
 & x_{ijk} \in \{0,1\}
\end{array}
$$

まとめると，なでしこリーグのスケジューリングでは,

1. 上の条件を満たすHAパターン（長さ9の'H', 'A'列）をすべて列挙
2. 1. でできたHAパターンから必要条件を満たすパターンセットを列挙
3. 2のそれぞれに対し，整数計画を解きスケジューリングを求める.

を順に実行する（チーム数が10と少ないため，このような列挙解法が可能である）.

HAパターンの列挙

次の条件を満たすHAパターン（長さ9の'H', 'A'列）をすべて列挙

- ９個の列のうち，Hは５つまたは4つ（Aは４つまたは５つ）とする.
- 部分文字列 "HH" や "AA"が表れる回数をブレイク数といい，ブレイク数は少ない方がよい．ここでは HH, AA ともに1回ずつを上限とする.
- 第1節と第2節は異なるHA状態とする.
- 第1節と第9節を同じHA状態とする（第9節と第10節は異なる）.
- 第8節と第9節を異なるHA状態にする.

In [ ]:
```
import itertools
from numpy import *
from pulp import *

n=10
days = range(n-1)
teams = ['A','B','C','D','E','F','G','H','I','J']

# まずは 'H'が４個または５個のパターンを作る．実際は 'H'=1, 'A'=0の0,1パターン.
p = [list(x) for x in itertools.combinations(days,4)]+\
   [list(x) for x in itertools.combinations(days,5)]
pat = zeros((len(p),n-1))
for i in range(len(p)):
   pat[i,p[i]] = 1

#条件に合わないものをふるいにかける
patterns = []
for p in pat:
   if p[0] != p[1] and p[0] == p[8] and p[7] != p[8]:
      hb = 0; ab = 0
      for j in range(len(p)-1):
         if p[j]==1 and p[j+1]==1:
            hb += 1
         if p[j]==0 and p[j+1]==0:
            ab += 1
      if hb <=1 and ab <=1:
         patterns.append(list(p))

# 出力
patterns = array(patterns)
print "条件に合うHA (1,0) パターンは全部で", len(patterns),"個"
patterns
```

上のパターンは(1.0,0.0)からなるものであり，行列として扱っている（演算のしやすさのため）． これを'H','A'パターンに変換することができる.

In [ ]:
```
res = [[patterns[i][j]*'H'+(1-patterns[i][j])*'A' for j in days] for i in range(len(patterns))]
res
```

パターンセットの列挙

次に 1. でできたHA (1,0) パターン から必要条件を満たすパターンセットを列挙

14個のHAパターンから10個選ぶ組み合わせのうち（\(_{14}C_{10} = 1001\)個ある），各節で'H'と'A'が同じ数だけあるものを選んでそれらのリストを作る.

In [ ]:
```
np = len(patterns)
pind =  [list(x) for x in itertools.combinations(range(np),n)]

pattern_sets = []
for p in pind:
    q = array(patterns[p])
    q = dot(q.transpose(),ones(n))
    r = array([n/2.0 for i in range(n-1)])
    s = [q[i]==r[i] for i in range(len(q))]
    if all(s):
        pattern_sets.append(array(patterns[p]))

print '必要条件を満たすパターンセットは全部で', len(pattern_sets), '通り．'
print 'パターンセット自体の出力は大きすぎるので省略'
```

In [ ]:
```
pattern_sets
```

整数計画を用いてスケジュール可能なパターンセットとスケジュール自体を見つける

最後に 2. で作った各パターン から整数計画問題を作って，スケジュール可能かどうかをチェック．

スケジュール可能ならば，そのパターンセットとスケジュール自体をリストに保存する．

In [ ]:
```
n=10
days = range(n-1)
teams = ['A','B','C','D','E','F','G','H','I','J']

ps_avail = []
sch_avail = []
for ps in pattern_sets:

    #最小化のモデル'Sports'を用意
    prob = LpProblem('Sports',LpMinimize)

    # 0,1変数を用意
    x = {}
    for i in range(n-1):
        for j in range(i+1,n):
            for k in days:
                x[i,j,k]= LpVariable('x('+str(i)+','+str(j)+','+str(k)+')', lowBound=0, cat=LpBinary)

    #目的関数を設定
    prob += 0

    # teams[i]とteams[j]が1度だけ対戦するという制約.
    for i in range(n-1):
        for j in range(i+1,n):
            prob += lpSum(x[i,j,k] for k in days) == 1, 'Const:team:'+str(teams[i])+',team:'+str(teams[j])

    # 各 team[i] の第k節での対戦相手は1チームだけという制約.
    for i in range(n):
        for k in days:
            prob += lpSum(x[j,i,k] for j in range(i)) + lpSum(x[i,j,k] for j in range(i+1,n)) == 1,\
                'Const:team:'+str(teams[i])+',day:'+str(k)

    #各k節で，パターンが同じチームは対戦しないという制約.
    for k in days:
        for i in range(n-1):
            for j in range(i+1,n):
                if ps[i,k] == ps[j,k]:
                    prob += x[i,j,k] ==0, \
                        'Const:samepattern:team:'+str(teams[i])+',team:'+str(teams[j])+',day:'+str(k)

    # 問題を解く．最適解があれば、status == 1
    status = prob.solve()

    if status == 'Optimal': #最適解があれば，変数の値からスケジューリング
        ps_avail.append([[ps[i][j]*'H'+(1-ps[i][j])*'A' for j in days] for i in range(n)])
        sch = [[str(k) for k in days] for i in range(n)]
        for k in days:
            for i in range(n-1):
                for j in range(i+1,n):
                    if x[i,j,k].varValue == 1.0:
                        if ps[i][k] == 1:
                            sch[i][k] = teams[j]+'.'
                            sch[j][k] = teams[i]
                        else:
                            sch[i][k] = teams[j]
                            sch[j][k] = teams[i]+'.'
        sch_avail.append(sch)

print 'スケジュール可能なパターンセットは'+str(len(sch_avail))+'個．'
```

スケジュール可能なパターンセットと，実際のスケジュール（他にも存在する可能性はある）を出力する．

In [ ]:
```
ps_avail[0]
```

In [ ]:
```
sch_avail[0]
```

考察

実際の日程との比較.

評価項目	旧	新
3連続アウェー	1	0
3連続ホーム	1	0
第1節，第2節にアウェー	2	0
第1節，第18節にアウェー	2	0
第17節，第18節にアウェー	0	0
強豪チームとの連続対戦	14	2

参考文献：

スポーツスケジューリング問題 ～２０１２年なでしこリーグの再スケジューリング～

東邦大学理学部情報科学科　2012年度卒業論文　伊藤仁勝（C言語と lp_solveを用いて）

授業で出題する演習課題を Python でやってみる（Mathematicaの代替となりうるか？）

- 星形を描く（簡単な線図形，VPython）
- 接線のアニメーション（数式処理，VPython）
- 三角形の内接円，外接円（数式処理，VPython）
- 987654321の約数
- 虫食い算（permutation）

星形を描け（簡単な線図形，VPython）

In [ ]:
```
from visual import *
from myGraphicModules import myXYAxes #自分で定義したXY軸

D = display()
myXYAxes(D, 'Star', False, -2,2,-2,2)

def points(i,t):
    return (2*cos((i%5)*2*pi/5.0+t),2*sin((i%5)*2*pi/5.0+t),0)

t = pi/10
ln = [curve(pos=[points(i,t),points((i+2)%5,t)],color=color.black,radius=0.03) for i in range(5)]

while True:
    rate(100)
```

星形を描き，さらに回転するアニメーションを作れ（簡単な線図形，VPython）
上のプログラムに，下の3行を最後の部分に付け加える.

In [ ]:
```
    # 差はこれ以下．線分の始点、終点の座標を変更して描き直す.
    t += 2*pi/1000
    for i in range(5):
        ln[i].pos = [points(i,t),points((i+2)%5,t)]
```

関数 \(f(x) = \frac{\sin x}{x^2+1}\) のグラフを描け

In [ ]:
```
from visual import *
from visual.graph import *

gdisplay(width=600,height=400,title="FunctionSample",foreground=color.white,background=color.black)

f=lambda x: sin(x)/(x**2+1)

g = gcurve()

for x in arange(0.0,2.0*pi,pi/100):
    g.plot(pos=(x,f(x)))

while 1:
    rate(100)
```

関数 \(f(x) = \frac{\sin x}{x^2+1}\) のグラフを描き，さらに \(x=\frac{\pi}{8}\) での接線を描け

自分で計算してもよいが，sympyを使って導関数を求めてみる.

In [ ]:
```
from sympy import *
x = Symbol('x')
f=lambda x: sin(x)/(x**2+1)
diff(f(x),x)
```

\(f(x)\) と接線の方程式を同時に描く.

In [ ]:
```
from visual import *
from visual.graph import *

gdisplay(width=600,height=400,title="FunctionSample",foreground=color.black,background=color.white)

f=lambda x: sin(x)/(x**2+1)
g=lambda x: -2*x*sin(x)/(x**2 + 1)**2 + cos(x)/(x**2 + 1)

g1 = gcurve()
g2 = gcurve()

a = pi/8
for x in arange(0.0,2.0*pi,pi/100):
    g1.plot(pos=(x,f(x)))

    g2.plot(pos=(x,g(a)*(x-a)+f(a)))
while 1:
    rate(100)
```

さらに接線を変化させて描くアニメーションをつくれ

plot機能を使うと，接線が残ってしまうのでアニメーションに適さない（matplotlib のplot機能も同じであろう）．VPythonのグラフィック機能を使う．

In [ ]:
```
from visual import *
from myGraphicModules import myXYAxes #自分で定義したXY軸

D = display()
myXYAxes(D,'接線のアニメーション', False, -1,6,-1,2,70,60)

f=lambda x: sin(x)/(x**2+1)
g=lambda x: -2*x*sin(x)/(x**2 + 1)**2 + cos(x)/(x**2 + 1)

p = [(x,f(x),0) for x in arange(0.0,2*pi, pi/30)]
a = 0.0
q = [(x, g(a) *(x-a)+f(a), 0) for x in arange(a-pi/2,a+pi/2, pi/30)]
func = curve(pos=p, color=color.black,radius=0.02)
line = curve(pos=q, color=color.red,radius=0.02)

while True:
    rate(10)
    a += pi/50
    line.pos = [(x, g(a) *(x-a)+f(a), 0) for x in arange(a-pi/2,a+pi/2, pi/50)]
    if a > 2*pi:
        a -= 2*pi
```

三角形の内接円，外接円を描け

一般に，三点 A(*x1,y1*)，B(*x2,y2*)，C(*x3,y3*) を引数とし，三角形ABCと三角形ABCの内接円，外接円を描く関数を定義せよ．

外接円の中心の座標と半径を，sympy（数式処理）の機能を使い，下の方程式を解くことによって得る．内接円の中心の座標と半径は，公式（Webで検索）より．

In [ ]:
```
from sympy import *

x, y, x1, y1, x2, y2, x3, y3, r2 = symbols('x y x1 y1 x2 y2 x3 y3 r2')
s =solve([(x-x1)**2+(y-y1)**2 - r2, (x-x2)**2+(y-y2)**2 - r2,(x-x3)**2+(y-y3)**2 - r2], [x, y, r2])
```

In [ ]:
```
print s[0][0]
print s[0][1]
print s[0][2]
```

上の結果を利用した，外接円（と内接円）を描くプログラムは こちら（myDrawing.py） (myDrawing.py).

In [ ]:
```
from visual import *
import myDrawing as md

p1=[-4.0,-1.0]; p2=[0.0,3.0]; p3=[4,-2]
md.Circles(p1,p2,p3)
```

整数 987654321 の約数をすべて求めよ

（この問題は 特に python でなくともよい）．

In [ ]:
```
for i in range(1,987654321):
    if 987654321 % i == 0:
        print i
```

なるべく効率よく計算せよ（計算をさぼる）．

In [ ]:
```
import math

for i in range(1, int(math.sqrt(987654321))+1):
    if 987654321 % i == 0:
        print i, 987654321/i
```

遊びやら

- 正多面体，半正多面体，半正多面体の双対をきれいに描く
- 任意の有界な3次元凸多面体を描く
- 多面体のスケルトングラフの全域木を描く（in preparation）．
- 多面体の展開図を描く（in consideration）
- Zometoolで遊ぶ

正多面体を描く

VPython（visualモジュール）を使って正十二面体を描いてみる．

In [ ]:
```
from visual import *

D = display() #display を用意．省略すると scene という displayが用意される．

D.width =600; D.height = 600 #幅と高さ
D.center = vector(0,0,0) #点(0,0,0)がdisplayの中心
D.range =3 #最初に display に表示される範囲
D.up = vector(0,0,1) #上方向のベクトル
D.forward = vector(-1,-1,-1) #画面奥の方向は (-1,-1,-1)
D.background = color.black #
D.title = "Dodeca" #窓のタイトル

points = [(-1.376381920, 0, 0.2628655561), (1.376381920, 0, -0.2628655561),
        (-0.4253254042, -1.309016994,  0.2628655561), (-0.4253254042, 1.309
016994,  0.2628655561),
        (1.113516364, -0.8090169944, 0.2628655561), (1.113516364, 0.809016
9944, 0.2628655561),
        (-0.2628655561, -0.8090169944, 1.113516364), (-0.2628655561, 0.8090
169944, 1.113516364),
        (-0.6881909602, -0.5000000000, -1.113516364), (-0.6881909602, 0.500
0000000, -1.113516364),
        (0.6881909602, -0.5000000000, 1.113516364), (0.6881909602, 0.50000
00000, 1.113516364),
        (0.8506508084, 0, -1.113516364), (-1.113516364, -0.8090169944, -0.26
28655561),
        (-1.113516364, 0.8090169944, -0.2628655561), (-0.8506508084, 0, 1.11
3516364),
        (0.2628655561, -0.8090169944, -1.113516364), (0.2628655561, 0.8090
169944, -1.113516364),
        (0.4253254042, -1.309016994, -0.2628655561), (0.4253254042, 1.3090
16994, -0.2628655561)]

edges = [[0, 13], [0, 14], [0, 15], [1, 4], [1, 5], [1, 12], [2, 6], [2,13],
       [2, 18], [3, 7], [3, 14], [3, 19], [4, 10], [4, 18], [5,11], [5, 19],
       [6, 10], [6, 15], [7, 11], [7, 15], [8, 9], [8,13], [8, 16], [9, 14],
       [9, 17], [10, 11], [12, 16], [12, 17], [16,18], [17, 19]]

faces = [[14, 9, 8, 13, 0], [1, 5, 11, 10, 4], [4, 10, 6, 2, 18], [10, 11, 7,15, 6],
       [11, 5, 19, 3, 7], [5, 1, 12, 17, 19], [1, 4, 18, 16, 12], [3, 19, 17, 9, 14],
       [17, 12, 16, 8, 9], [16, 18, 2, 13, 8], [2,6, 15, 0, 13], [15, 7, 3, 14, 0]]

# 頂点をランダムな色で描く．
#pts = [sphere(pos=v, color=(random(),random(),random()),radius=0.1,opacity=0.5)
 for v in points]

#稜線を白で描く．
eds = [curve(pos=[points[e[0]],points[e[1]]],color=color.yellow,radius=0.05) for e in edges]

# 面をランダムな色で描く．
#fa = [convex(pos=[points[f] for f in fcs] ,color=(random(),random(),random())) for fcs in faces]

# 多面体自体をシアンで描く．
#convex(pos=points,color=color.cyan)

while 1:
   rate(10)
```

正多面体（プラトンの多面体），半正多面体（アルキメデスの多面体），半正多面体の双対多面体

頂点の座標などのデータベースはこちら（PolyhedronData.py） (PolyhedronData.py)

In [1]:
```
from PolyhedronData import *

Names["Types"]
```

Out[1]: ['Platonic', 'Archimedean', 'ArchimedeanDual', 'All']

In [2]:
```
Names['ArchimedeanDual']
```

Out[2]:
```
['TriakisTetrahedron',
 'TriakisIcosahedron',
 'TetrakisHexahedron',
 'SmallTriakisOctahedron',
 'RhombicTriacontahedron',
 'RhombicDodecahedron',
 'PentakisDodecahedron',
 'PentagonalIcositetrahedron',
 'PentagonalHexecontahedron',
 'DisdyakisTriacontahedron',
 'DisdyakisDodecahedron',
 'DeltoidalIcositetrahedron',
 'DeltoidalHexecontahedron']
```

In [ ]:
```
from visual import *
from PolyhedronData import *

name = 'TetrakisHexahedron'

D = display() #display を用意．省略すると scene という displayが用意される．
D.width =600; D.height = 600 #幅と高さ
D.center = vector(0,0,0) #点(0,0,0)がdisplayの中心
D.range =2 #最初に display に表示される範囲
D.up = vector(0,0,1) #上方向のベクトル
D.forward = vector(-1,-1,-1) #画面奥の方向は (-1,-1,-1)
D.background = color.black #
D.title = name  #窓のタイトル

points = Vertices[name]
edges = Edges[name ]
faces = Faces[name ]

# 頂点をランダムな色で描く．
#pts = [sphere(pos=v, color=(random(),random(),random()),radius=0.1,opacity=0.5)
 for v in points]

#稜線を白で描く．
eds = [curve(pos=[points[e[0]],points[e[1]]],color=color.white,radius=0.05) for e in
 edges]

# 面をランダムな色で描く．
#fa = [convex(pos=[points[f] for f in fcs] ,color=(random(),random(),random())) for f
cs in faces]

# 多面体自体をシアンで描く．

convex(pos=points,color=color.cyan)

while 1:
   rate(10)
```

ランダムな着色

In [ ]:
```
from visual import *
from PolyhedronData import *
from random import *
name = 'DeltoidalIcositetrahedron'

D = display() #display を用意．省略すると scene という displayが用意される．
D.width =600; D.height = 600 #幅と高さ
D.center = vector(0,0,0) #点(0,0,0)がdisplayの中心
D.range =4 #最初に display に表示される範囲
D.up = vector(0,0,1) #上方向のベクトル
D.forward = vector(-1,-1,-1) #画面奥の方向は (-1,-1,-1)
D.background = color.black # バックグラウンドの色
D.title = name  #窓のタイトル

points = Vertices[name]
edges = Edges[name ]
faces = Faces[name ]


# 頂点をランダムな色で描く．
pts = [sphere(pos=v, color=(random(),random(),random()),radius=0.1,opacity=0.5)
for v in points]

#稜線を白で描く．
eds = [curve(pos=[points[e[0]],points[e[1]]],color=color.white,radius=0.05) for e in
 edges]

# 面をランダムな色で描く．
#fa = [convex(pos=[points[f] for f in fcs] ,color=(random(),random(),random())) for f
cs in faces]

# 多面体自体をシアンで描く．
#convex(pos=points,color=color.cyan)

while 1:
   rate(100)
```

一般的な3次元の有界凸多面体の描画

有限個の不等式で定義される3次元有界凸多面体 $P = \{x \mid b+A\,x >= 0\}$ を描画する． ただし、A は$m \times n$行列，bはm次元ベクトル，xは3次元ベクトルである．そのために，外部のプログラム lrs を呼び出して， 端点の座標，稜線と面のインデックスを計算する関数定義は こちら（VertexEnum.py） (VertexEnum.py).

In [ ]:
```
from VertexEnum import *
from visual import *
from random import *

D = display() #display を用意．省略すると scene という displayが用意される．
D.width =600; D.height = 600 #幅と高さ
D.center = vector(0,0,0) #点(0,0,0)がdisplayの中心
D.range =2 #最初に display に表示される範囲
D.up = vector(0,0,1) #上方向のベクトル
D.forward = vector(-1,-1,-1) #画面奥の方向は (-1,-1,-1)
D.background = color.black #
D.title = "ランダムな多面体"  #窓のタイトル



# R^3中の原点中心の半径1の球面に接する50個の超平面の作成
#seed(54321)
n = 200
b = [100000 for i in range(n)]
A = [[uniform(-1.0,1.0), uniform(-1.0,1.0), uniform(-1.0,1.0)] for i in range(n)]
for i in range(n):
    A[i] = A[i] /linalg.norm(A[i])
    A[i] = [int(100000*j) for j in A[i]]
A = array(A)


# 端点の座標，稜線, 面 の Indexの計算
points, edges, faces = VertexEnumeration(b,A)


# 頂点をランダムな色で描く．
#pts = [sphere(pos=v, color=(random(),random(),random()),radius=0.1,opacity=0.9)
 for v in points]

#稜線を白で描く．
eds = [curve(pos=[points[e[0]],points[e[1]]],color=color.white,radius=0.01) for e in
 edges]

# 面をランダムな色で描く．
fa = [convex(pos=[points[f] for f in fcs] ,color=(random(),random(),random())) for f
cs in faces]

# 多面体自体をシアンで描く．
#convex(pos=points,color=color.cyan)


while 1:
    rate(100)
```

In [ ]: