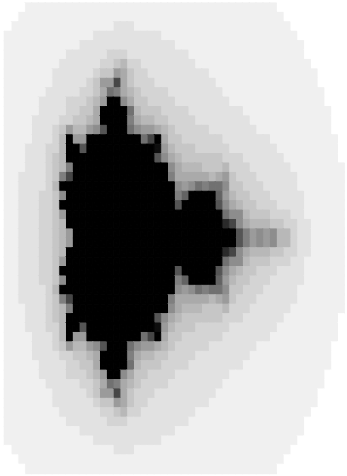
第2章　情報科学の諸問題への応用

2.1　Mandelbrot 集合の描画

課題

以下に示す Mandelbrot 集合 を描け.



課題の背景

専門書をひもとくと (「C 言語によるアルゴリズム事典」, 奥村晴彦著, 技術評論社 1991)

```
Mandelbrot(マンデルブロート) 集合　Mandelbrot set
(中略)
計算機では, ある領域の点 (x,y) について,
z <-- x + i y; count <-- M;
while ( |z| <= 4) and ( count>0) do begin
  z <-- z^2 - (x + i Y); count <-- count -1
```

end;
点 (x,y) に count で決まる色 (count=0 なら黒) を付ける;

とする．黒い部分が Mandelbrot 集合で，それ以外の色は，その点と Mandelbrot 集合との ”近さ ” を表す光背である．

となっています．

Maple の関連コマンド

これを Maple で実現しようと思うと，Mandelbrot(x,y) という関数 (procedure) を作って，count の初期値を例えば 20 として以下のようにして densityplot 関数に食わせます．

```
> with(plots):
> densityplot('Mandelbrot(x,y)',x=-1..2.5,y=-1.5..1.5,
> axes=none,style=patchnogrid,grid=[50,50]);
```

ここで，densityplot の第一引数の関数 Mandelbrot がシングルクォートで囲まれていないと描画の時にエラーがでます[1]．

解法のヒントと発展問題

1. Mandelbrot 関数を作れ．count の初期値を 20 とした場合は，[x,y] が [0,0]，[-0.5,0]，[0,1.5] での値は，それぞれ $0, 14, 17$ となることを確認せよ．
2. plot('Mandelbrot(x,0)',x=-1..2.5); を試せ．
3. densityplot 関数で表示せよ．さらにきれいになるように densityplot の option を，　grid=[200,200]，colorstyle=HUE としてみよ．

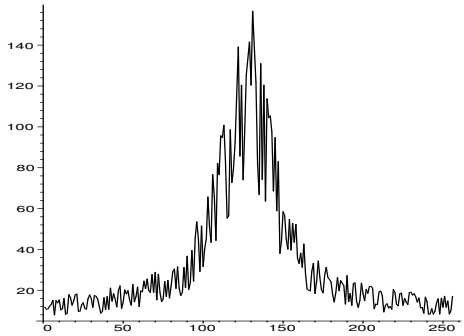
[1] これは値の評価によるエラーです．詳しくはラーニングガイドの「評価と簡単化」を参照ください．

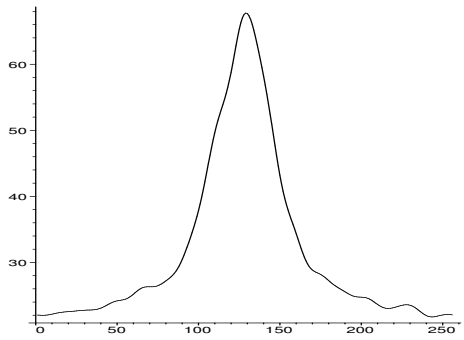
2.2 FFT(Fast Fourier Transfer)

課題

下図の上段パネルのようなデータをフーリエ変換し，三角フィルタによってノイズを除去せよ．データは次のようにして作成・表示しました．

```
> f1:=t->subs({a=10,b=40000,c=380,d=128},a+b/(c+(t-d)^2)):
> T:=[seq(f1(i)*(0.6+0.8*evalf(rand()/10^12)),i=1..256)]:
> with(plots):listplot(T);
```





課題の背景

フーリエ変換を用いたデータのフィルタリングについてです[2]．時系列データ $x(t)$ の解析にはそのフーリエ変換

$$X(f) = \int_{-\infty}^{\infty} x(t) \exp(2\pi i f t) dt \tag{2.1}$$

が多用されます．これは，高速フーリエ変換 (Fast Fourier Transformation:略してFFT) アルゴリズムの発明によって計算機での処理が容易となったためです．時間

[2]参考 河合潤：『化学計測学』 合志陽一編 昭晃堂 1997.

経過と共に変化する時系列測定データ $x(t)$ に含まれるさまざまな周波数成分について，何 Hz の周波数成分がどの程度の割合でその時系列データの中に含まれているのかを明らかにできるからです．時系列データの中のノイズは δ 関数に似ており，そのフーリエ変換は全周波数成分を一様に含んでいます．一方，測定したいデータは一般に，時間に対してゆっくり変化する量です．測定によって得られたデータの時間軸は連続ではなくとびとびの値をとるので，離散フーリエ変換を取ることになります．この例では時間データは 256 チャンネルのデータです．高周波成分（=ノイズ成分）を取り除くために，データに対して窓関数（低域通過フィルタ）をかけてフーリエ逆変換

$$x(t) = \int_{-\infty}^{\infty} X(f) \exp(2\pi i f t) dt \tag{2.2}$$

することによってフィルターを掛けた後のスムーズなカーブが得られます．低域通過フィルタには，方形窓，修正方形窓，フォンハン窓（２乗余弦窓），ハミング窓などさまざまなものがあります．

Maple の関連コマンド

実際の FFT を少ないデータで見てみます．まずデータを用意します．

```
> Rdata:=array([1,2,3,4,5,4,3,2]);
> Idata:=array(1..8,sparse);
```

$$Rdata := [1, 2, 3, 4, 5, 4, 3, 2]$$

$$Idata := \mathrm{array}(sparse, 1..8, [])$$

これに FFT をかけます．

```
> FFT(3,Rdata,Idata);
```

$$8$$

これだけで終わりです．ここで FFT の第一引数の 3 はデータの個数が 2^3 であることを示しています．フーリエ変換された後のデータ構造を見ておきましょう．

```
> print(Rdata):
> print(Idata):
```

$$[24, -6.828427123, 0., -1.171572877, 0, -1.171572877, 0., -6.828427123]$$

$$[0, 0.1\,10^{-8}, 0., 0.1\,10^{-8}, 0, -0.1\,10^{-8}, 0., -0.1\,10^{-8}]$$

逆変換とその結果です．

```
> iFFT(3,Rdata,Idata);
```

$$8$$

```
> print(Rdata);
> print(Idata);
```

[1.000000000, 2.000000001, 3.000000000, 3.999999999, 5.000000000, 3.999999999, 3.000000000, 2.000000001]

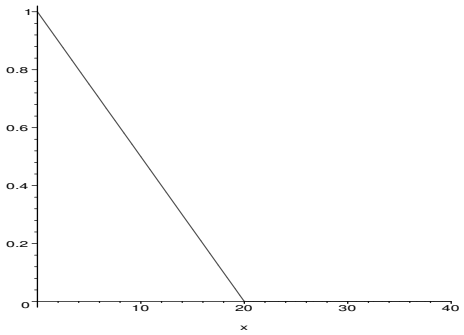
$[0., -0.2500000000\,10^{-9}, 0., -0.2500000000\,10^{-9}, 0., 0.2500000000\,10^{-9}, 0., 0.2500000000\,10^{-9}]$

三角フィルター関数は以下のようにして作られます.

```
> filter:=x->piecewise(x>=0 and x<=20, (1-x/20));
```

$$filter := x \rightarrow \text{piecewise}(0 \le x \textbf{ and } x \le 20, 1 - \frac{1}{20}\,x)$$

```
> plot(filter(x),x=0..40);
```



解法のヒント

1. 256 個のデータを Rdata に読み込め.
2. FFT をかけよ.
3. FFT をかけた後のデータの強度 (`Rdata[i]^2+Idata[i]^2`) を `logplot` せよ.
4. FFT をかけた後のデータに filter 関数を掛けよ.
5. フーリエ逆変換してその結果を表示せよ.

2.3　非線形最小二乗法

課題

以下のスクリプトで与えられたデータ

```
> f1:=t->subs({a=10,b=40000,c=380,d=128},a+b/(c+(t-d)^2)):
> T:=[seq(f1(i)*(0.6+0.8*evalf(rand()/10^12)),i=1..256)]:
> with(plots):listplot(T);
```

を，

$$f(x) = a + \frac{b}{c + (x-d)^2} \tag{2.3}$$

なるローレンツ型関数にカーブフィッティングするプログラムを作成し，そのパラメータを決定せよ．ここで a はバックグラウンドの強度，b はローレンツ関数の強度，c は線幅，d はピーク位置を表す．

課題の背景

(2.3) 式の特徴は，パラメータが線形関係にないということです．非線形の最小二乗法を用いてデータフィットをしてくれる関数は default では Maple には用意されていないようです[3]．そこで，実際に非線形最小二乗法のプログラムを作成し，実際のデータへのフィッティングを試みてみましょう．非線形最小二乗法の注意事項は補足に記しました．

今，パラメータの初期値を $(a_0 + \Delta a_1, b_0 + \Delta b_1, c_0 + \Delta c_1, d_0 + \Delta d_1)$ としましょう．このとき関数 f を真値 (a_0, b_0, c_0, d_0) のまわりでテイラー展開し高次項を無視すると

$$\begin{aligned} \Delta f &= f(a_0 + \Delta a_1, b_0 + \Delta b_1, c_0 + \Delta c_1, d_0 + \Delta d_1) \\ &\quad - f(a_0, b_0, c_0, d_0) \\ &= \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial a}\Delta a_1 + \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial b}\Delta b_1 \\ &+ \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial c}\Delta c_1 + \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial d}\Delta d_1 \end{aligned} \tag{2.4}$$

となります．(2.3) 式の偏微分は計算可能です．'DATA101' のデータは $t = 1$ から 256 までの時刻に対応したデータ点 $f_1 \sim f_{256}$ とします．各測定値とモデル関数か

[3]公式のサポートではありませんが，世界中の科学者，技術者が Maple で開発した library を公開している The Maple Application Center (http://www.mapleapps.com/) の Data Analysis のカテゴリーに，後述する Levenberg-Marguqrdt 法を用いた Data fitting の汎用プログラム Generalized Weighted Non-Linear Regression Using the Levenberg-Marquardt Method by David Holmgren (http://www.mapleapps.com/categories/data_analysis_stats/data/html/genfit_6.html) があります．

ら予想される値との差 $\Delta f_1 \sim \Delta f_{256}$ は，

$$\begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} = J \begin{pmatrix} \Delta a_1 \\ \Delta b_1 \\ \Delta c_1 \\ \Delta d_1 \end{pmatrix} \tag{2.5}$$

となります．ここで 4 行 256 列の行列

$$J = \begin{pmatrix} \left(\frac{\partial f}{\partial a}\right)_1 & \left(\frac{\partial f}{\partial b}\right)_1 & \left(\frac{\partial f}{\partial c}\right)_1 & \left(\frac{\partial f}{\partial d}\right)_1 \\ \vdots & \vdots & \vdots & \vdots \\ \left(\frac{\partial f}{\partial a}\right)_{256} & \left(\frac{\partial f}{\partial b}\right)_{256} & \left(\frac{\partial f}{\partial c}\right)_{256} & \left(\frac{\partial f}{\partial d}\right)_{256} \end{pmatrix} \tag{2.6}$$

はヤコビ行列と呼ばれる行列です．逆行列 J^{-1} は転置行列 J^t を用いて

$$J^{-1} = (J^t J)^{-1} J^t \tag{2.7}$$

と表されます．従って真値からのずれは

$$\begin{pmatrix} \Delta a_2 \\ \Delta b_2 \\ \Delta c_2 \\ \Delta d_2 \end{pmatrix} = (J^t J)^{-1} J^t \begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} \tag{2.8}$$

として計算できます．理想的には $(\Delta a_2, \Delta b_2, \Delta c_2, \Delta d_2)$ は $(\Delta a, \Delta b, \Delta c, \Delta d)$ に一致するはずですが，測定誤差と高次項のために一致しません．初期値に比べ，より真値に近づくだけです．そこで，新たに得られたパラメータの組を新たな初期値に用いて，より良いパラメータに近付けていくという操作を繰り返します．新たに得られたパラメータと前のパラメータとの差がある誤差以下になったところで計算を打ち切り，フィッティングの終了となります．

Maple の関連コマンド

実際にパラメータフィットを行っていきましょう．線形代数計算のためにサブパッケージとして LinearAlgebra を呼びだしておきます．

```
> with(LinearAlgebra):
```

データを読み込みます．

```
> datapoint:=[seq([i,T[i]],i=1..256)]:
```

(2.3) 式のローレンツ型の関数を仮定しておきましょう.

```
> f:=a+b/(c+(x-d)^2);
```

$$f := a + \frac{b}{c + (x-d)^2}$$

```
> f1:=unapply(f,x);
```

$$f1 := x \to a + \frac{b}{c + (x-d)^2}$$

(2.3) 式の関数の微分を求め，新たな関数として定義します.

```
> dfda:=unapply(diff(f,a),x);
> dfdb:=unapply(diff(f,b),x);
> dfdc:=unapply(diff(f,c),x);
> dfdd:=unapply(diff(f,d),x);
```

$$dfda := x \to 1$$

$$dfdb := x \to \frac{1}{c + (x-d)^2}$$

$$dfdc := x \to -\frac{b}{(c + (x-d)^2)^2}$$

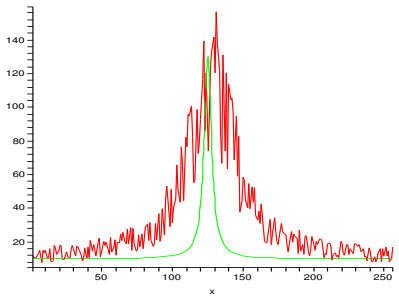
$$dfdd := x \to -\frac{b\,(-2\,x + 2\,d)}{(c + (x-d)^2)^2}$$

初期値を仮定して，表示します.

```
> guess1:={a=10,b=1200,c=10,d=125};
```

$$guess1 := \{a = 10,\ b = 1200,\ d = 125,\ c = 10\}$$

```
> plot([datapoint,subs(guess1,f1(x))],x=1..256);
```



解法のヒント

1. (2.5) 式の左辺の Δf_i を求めよ．`T[i]-f1(i)` を 1..imax まで求め，Vector に入れる．
2. (2.6) 式に従ってヤコビ行列を求めよ．
3. (2.7) 式の公式によるヤコビ行列の逆行列を求めよ．また，(2.8) 式により，新たなパラメータの組を求めよ．
4. 求めたパラメータを用いたモデル関数と，データをプロットしてみよ．前回より近づいているのがわかるでしょう．
5. 上の操作を適当に繰り返し，パラメータを収束させよ．その値とプロットを示せ．

補足:最小二乗法に関するメモ

前述のフィッティング法の説明では，テイラー展開を用いた説明であり，まるで最小二乗法を用いてないような印象を与えたかもしれません．しかしこれは，最小二乗法の χ^2 関数に Newton-Raphson 法を適用し，二次微分を無視した方法と考えることも可能です．この様子を以下に記しておきます．

当てはめたいモデルは x がデータの横軸，$\mathbf{a}$ がパラメータの組とすると，

$$y = f(x; \mathbf{a}) \tag{2.9}$$

です．最小二乗法の χ^2 関数は

$$\chi^2(\mathbf{a}) = S(\mathbf{a}) = \sum_{i=1}^{N} [y_i - f(x_i; \mathbf{a})]^2 \tag{2.10}$$

です．後の式を単純にするために $y_i - f(x_i; \mathbf{a}) \Rightarrow f(x_i; \mathbf{a})$ と置き換え，あらかじめ実験値を引いておきます．S が $\mathbf{a}$ で極小値を持つ (安定である) ための条件は

$$\frac{\partial S(\mathbf{a})}{\partial a_k} = -2 \sum_{i=1}^{N} f(x_i; \mathbf{a}) \frac{\partial f(x_i; \mathbf{a})}{\partial a_k} = 0. \tag{2.11}$$

ヤコビ行列を $J_i(\mathbf{a}) = \left[\frac{\partial f_i}{\partial a_j}\right]$ と表すと

$$\nabla S(\mathbf{a}) = -2 J^t(\mathbf{a}) f(y; \mathbf{a}) = 0 \tag{2.12}$$

と書き換えることができます．これに Newton-Raphson 法を適用すると

$$\nabla^2 S(\mathbf{a})\Delta\mathbf{a}^{(k)} = -\nabla S(\mathbf{a}^{(k)}) \tag{2.13}$$

となります． $S(\mathbf{a})$ のあらわな 2 次微分の表現は

$$\frac{\partial^2 S(\mathbf{a})}{\partial a_k \partial a_{k'}} = -2\left\{\sum_{i=1}^{N} \frac{\partial f(x_i;\mathbf{a})}{\partial a_k}\frac{\partial f(x_i;\mathbf{a})}{\partial a_{k'}} + \sum_{i=1}^{N} \frac{\partial^2 f(x_i;\mathbf{a})}{\partial a_k \partial a_{k'}} f(x_i;\mathbf{a})\right\} \tag{2.14}$$

です．ここで，k 回目の推定パラメータ $\mathbf{a}^{(k)}$ に対する修正値を $\Delta\mathbf{a}^{(k)}$ とすると，

$$\left\{J^t\left(\mathbf{a}^{(k)}\right) J\left(\mathbf{a}^{(k)}\right) + \sum_{i=1}^{N} f\left(x_i;\mathbf{a}^{(k)}\right)\nabla^2 f\left(x_i;\mathbf{a}^{(k)}\right)\right\}\Delta\mathbf{a}^{(k)} = J^t\left(\mathbf{a}^{(k)}\right) f\left(x_i;\mathbf{a}^{(k)}\right) \tag{2.15}$$

が得られます．鉤括弧内の第 2 項を無視すると，先程の（2.5）式の線形的な関係が現れます．この無視する項は，線形の場合には確かに現れないし，1 次の項に比べて小さいと考えられます．さらに $f(x_i;\mathbf{a})$ を掛けて和を取っているため，それらがお互いにキャンセルしてくれる可能性があります．

この Gauss- Newton 法と呼ばれる非線形最小二乗法は線形問題から拡張した方法として論理的に簡明であり，広く使われています．しかし，収束性は高くなく，むしろ発散しやすいので注意が必要です．先程の 2 次の項を無視するのでなく，うまく見積もる方法を用いたのが Levenberg-Marquardt 法です．明快な解説が Numerical Recipes in C(C 言語による数値計算のレシピ）WilliamH.Press 他著，技術評論社 1993 にあります．

2.4 巡回セールスマン問題

課題

ランダムに生成した 16 都市を順にめぐって元に戻ってくる最短経路をアニーリング法 (simulated annealing) を用いて求めよ．$N = 16$ の場合の (a) 初期配置，(b) 単純に総距離が下がった場合だけを採用した結果，および (c) simulated annealing の計算結果を図 2.1 に示す．

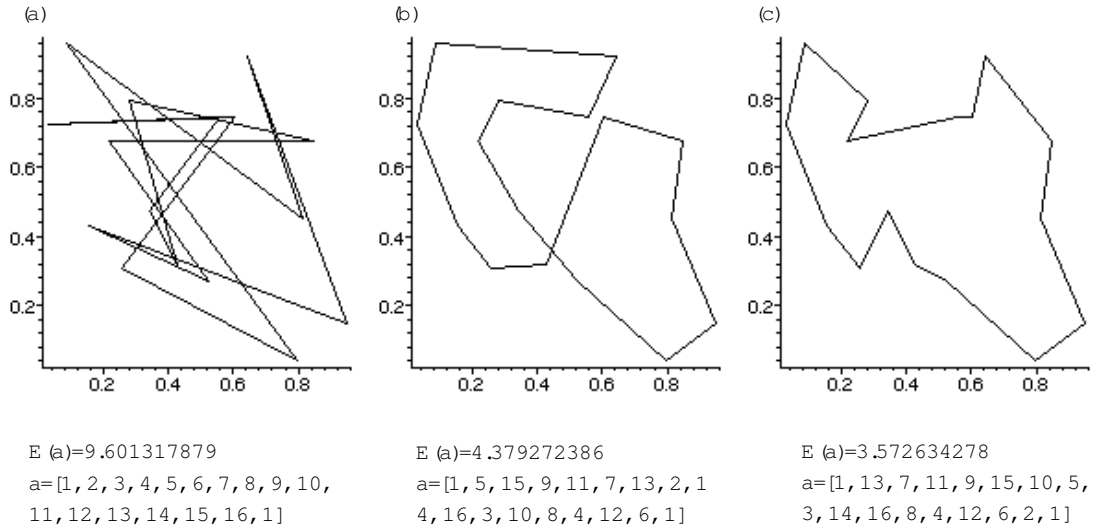


図 2.1: ランダムに生成した 16 個の街に適用した巡回セールスマン問題の (a) 初期配置，(b) 単純に総距離が下がった場合だけを採用した結果，および (c) simulated annealing の計算結果．ただし，(c) は最短経路ではない．

課題の背景

巡回セールスマン問題とは，ある街から出発していくつかの街を次々とめぐって元の街に戻ってくる最短の経路を求める問題です．訪れる街の数が少ないときにはすべての経路を数え上げればいいのですが，数が増えるとその計算時間は指数関数的に増えてしまうと予想される問題です．このような問題はセールスマンだけでなく，コンピュータの CPU の配置や，都市ガスの配管設計などでも出会う問題です．

対象となる経路の長さの合計を

$$E(\mathbf{a}) = \sum_{i=1..N} \|\mathbf{r}[\mathbf{a}_i] - \mathbf{r}[\mathbf{a}_{i+1}]\| \tag{2.16}$$

としておきます．ここで $\mathbf{a}$ はそれぞれの街の順番あるいは配置を示しています．$\mathbf{r}$ はそれぞれ街の座標で $\|\mathbf{r}\|$ で距離を求めます．するとこの関数は模式的に図 2.2 のようになると考えることができます．すべての配置について $E(\mathbf{a})$ を求めれば，最小値が求まります．あるいは初期の配置を

$$\mathbf{a} = [1, 2, 3, \ldots, N, 1] \tag{2.17}$$

として，一定の手順で変更 $\delta\mathbf{a}$ を加え，$E(\mathbf{a})$ が下がった場合にその配置を採用するという方法をとることも出来ます．しかし，この方法は極小値に落ちてしまうことが分かるでしょう．最小値を探すには時として坂を駆け登る必要があるのです．このような問題の一つの解法としてアニーリング法 (simulated annealing) があります．これは格子欠陥を多く含んだ金属を高温へ上げて欠陥を掃き出し柔らかくする熱処理 (焼きなまし，annealing) からの類推で名付けられた手法です．

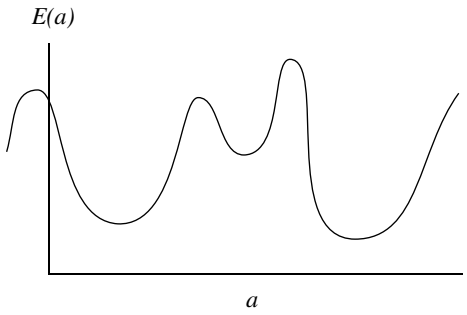


図 2.2: 配置 $\mathbf{a}$ とその経路の総和 $E(\mathbf{a})$ を示す模式図

アルゴリズムは以下のとおりです．

1. 配置 $\mathbf{a}$ を仮定し $E(\mathbf{a})$ を求める．
2. $\mathbf{a}$ からすこし違った配置 $\mathbf{a} + \delta\mathbf{a}$ を作る．
3. $\Delta E = E(\mathbf{a} + \delta\mathbf{a}) - E(\mathbf{a})$ を求める．
4. $\Delta E < 0$ なら新たな配置を採用する．
5. $\Delta E > 0$ なら新たな配置を $\exp(-\Delta E/T)$ の確率で受け入れる．

6. 手順 2 以下を適当な回数繰り返す.

ここで T は温度から類推される制御パラメータで，十分大きい場合はすべての状態が採用されます．一方，T を下げるにつれて採用される試行が少なくなり，徐々に状態が凍結されていきます．$\mathbf{a}+\delta\mathbf{a}$ の生成方法と T の下げ方をうまく取れば最小値に近い状態が確率的に高く出現し，最小値かそれに近い状態へ落ち着くというアルゴリズムです.

Maple の関連コマンド

以下は街の位置をランダムに生成し，経路を表示する Maple script.

```
restart;
with(plots):
N_city:=16;
Path:=[seq(i,i=1..N_city),1];
Position:=seq([evalf(rand()/10^12),evalf(rand()/10^12)],i=1..N_city):
Real_Path:=[seq(Position[Path[i]],i=1..N_city),Position[1]]:
PLOT(CURVES(Real_Path));
```

また,

```
sel_city:=rand(2..N_city):
```

によって 2 から N_city までの整数をランダムに生成する関数が出来る.

解法のヒントと発展問題

1. 街の間の距離を計算した 2 次元のリストを作れ.
2. 配置を受け取って，前問で作成したリストを参照して経路の総距離を返す関数を作れ.
3. 配置 $\mathbf{a}$ (=Path) から任意に 2 都市を取りだして入れ替える試行を $\delta\mathbf{a}$ として上述のアルゴリズムにしたがって最短経路を探すプログラムを作成せよ.
4. 総距離を記録して，その結果を表示せよ.
5. 【発展問題】さらに総距離とそのルートを記録する関数を作って，その結果を表示せよ.

6. 【発展問題】ルートの入れ替えを工夫して 64 都市でも解けるようにせよ．例えば，

   - 切り出したルートを逆順にして埋めもどしたり，
   - ルートの一部を切り出して他へ移す

   等の操作を組み込む．

2.5 ソート

課題

選択ソートとクイックソートアルゴリズムを用いて，配列データ A:=array([6,4,3,1,2,5]) を小さい値順に並べるソートスクリプトを作れ．

課題の背景

簡単な挿入 (insert) ソートを例に，配列に入った値がどのようにソートされていくかを Maple で表示してみます．挿入ソートのスクリプトは，

Maple Script ： 挿入ソート

```
insertsort:=proc(A::array)
local n,x,i,j;
n:=op(2,op(2,eval(A)));
for i from 1 to n do
  print("Before",i,A[i],A);
  x:=A[i];
  for j from i-1 by -1 to 1 do
    if (A[j]<x) then break; end if;
    A[j+1]:=A[j];
  end do;
  A[j+1]:=x;
  print("After ",i,j+1,A);
end do:
eval(A);
end proc:
```

です．

ソートでは配列の並べ替えをおこなうためにどうしても複雑な index の動きをしますが，ゆっくり見ていけば理解できるはずです．上の関数を使ってたとえば

```
> A:=array([6,4,3,1,2,5]);
> insertsort(A);
```

とした場合の結果は，

```
                A := [6, 4, 3, 1, 2, 5]
        "Before", 1, 6, [6, 4, 3, 1, 2, 5]
        "After ", 1, 1, [6, 4, 3, 1, 2, 5]
```

```
"Before", 2, 4, [6, 4, 3, 1, 2, 5]
"After ", 2, 1, [4, 6, 3, 1, 2, 5]
"Before", 3, 3, [4, 6, 3, 1, 2, 5]
"After ", 3, 1, [3, 4, 6, 1, 2, 5]
"Before", 4, 1, [3, 4, 6, 1, 2, 5]
"After ", 4, 1, [1, 3, 4, 6, 2, 5]
"Before", 5, 2, [1, 3, 4, 6, 2, 5]
"After ", 5, 2, [1, 2, 3, 4, 6, 5]
"Before", 6, 5, [1, 2, 3, 4, 6, 5]
"After ", 6, 5, [1, 2, 3, 4, 5, 6]
        [1, 2, 3, 4, 5, 6]
```

となります．挿入する前 (Before) と後 (After) の配列の違いを観察してください．たとえば，4 番目まで終わった，

```
"Before", 5, 2, [1, 3, 4, 6, 2, 5]
```

で内側の for-loop で何をしているかを考えます．5 番目の要素の”2”が選ばれます (x:=A[i])．これを右側のもの (A[j]) と順番に比べていきます．大きければ空いた席 (A[j+1]) と入れ替えます．

Compare 出力

```
"Compare", 5, 5, [1, 3, 4, 6, _, 5]
"Compare", 5, 4, [1, 3, 4, _, 6, 5]
"Compare", 5, 3, [1, 3, _, 4, 6, 5]
"Compare", 5, 2, [1, _, 3, 4, 6, 5]
```

小さくなる (A[j] < x) かあるいは j=1 までいけば比較は終了です．5 番目の要素は 1 番目までいって，A[1] < x が成立して，A[2] に収まります．

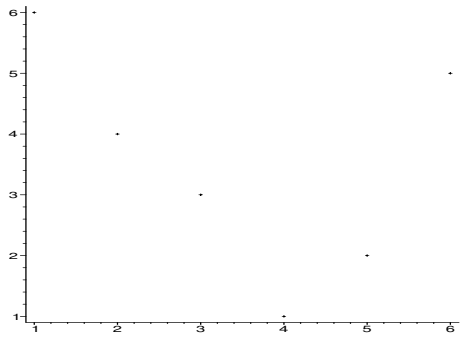
Maple の関連コマンド

ソートにともなう値の並べ換えを視覚化することを試みます．以下のようにすると

```
> A:=array([6,4,3,1,2,5]);
> tmp:=[seq([i,A[i]],i=1..6)];
> with(plots):
> pointplot(tmp);
```

$$A := [6, 4, 3, 1, 2, 5]$$

$$tmp := [[1, 6], [2, 4], [3, 3], [4, 1], [5, 2], [6, 5]]$$

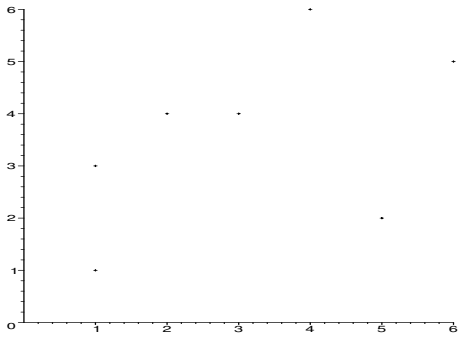


と表示されます．よこがindex，たてが入っている要素です．これを使って先程のinsertsortの途中経過を表示させます．

insertsort2（ソートの視覚化を組み込んだinsertsort）

```
insertsort2:=proc(A::array)
local n,x,i,j;
global tmp;
n:=op(op(2,eval(A)))[2];
for i from 2 to n do
  x:=A[i];
  for j from i-1 to 1 by -1 do
  tmp:=[op(tmp),pointplot([[j,x],seq([ii,A[ii]],ii=1..n)])];
    if (A[j]<x) then break; end if;
    A[j+1]:=A[j];
  od;
  A[j+1]:=x;
od;
end proc:
```

```
> tmp:=[]:
> A:=array([6,4,3,1,2,5]):
> insertsort2(A):
> tmp:=[op(tmp),pointplot([seq([i,A[i]],i=1..6)])]:
> display(tmp,insequence=true,view=[0..6,0..6]);
```



tmp の中に画像が蓄えられています．ここでは 6 番目の画像だけを表示しています．Maple を実際に手元で動かしているときは，display で表示させた絵をつまんで，コマ送りで一つずつどのように位置が置き変わっていくかを追ってみてください．

解法のヒントと発展問題

1. 上記の”Compare”以下が出力されるように insertsort を改良せよ．ただし下線”_”部には違う数字が入っている．その数字はなにか？

2. 次に示した選択ソートの考え方にもとづいて Maple スクリプト (selectsort) を書け．

   (a) まず，全体の中で最小のものを見つけ，それを先頭の `A[1]` と交換する．

   (b) 次に `A[2..n]` の中で最小のものを見つけ，`A[2]` と交換する．

   (c) 以下同様に続ける．

   さらに，視覚化を試みよ．

3. 上の insertsort を参考にして，選んだ数より大きな数を右に，小さな数を左に置きかえる関数 partition(A,first,last) を考えよ．選ぶ数としては A[last] が簡単．これを使えば，quicksort は以下のようなスクリプトで書ける．

```
quicksort:=proc(A::array,first::integer,last::integer)
local p;
if first<last then
  p:=partition(A,first,last);
  quicksort(A,first,p-1);
```

```
  quicksort(A,p+1,last);
end if:
eval(A);
end proc:
```

4. 【発展問題】以下のランダムな整数列の作成の proc を用いて要素数を大きくしたときに，insertsort, selectsort, quicksort それぞれがどの程度の時間がかかるかを計測せよ．

```
BigRandom:=proc(max_n)
local sel,A,i,j,k,tmp;
sel:=rand(1..max_n):
A:=array([seq(i,i=1..max_n)]):
for k from 1 to 2*max_n do
  i:=sel();j:=sel();
  tmp:=A[i];A[i]:=A[j];A[j]:=tmp;
end do:
return A;
end proc:
```

時間計測は以下のようにして出来る．(printf) 文をのぞいておくことを忘れないように．

```
> st:= time():
> n:=160:
> A:=BigRandom(n):
> quicksort(A,1,n);
> time() - st;
```

2.6　RSA 暗号

課題

以下の RSA モジュール N，復号化指数 DD の組み合わせで送られてきた RSA 暗号文 ciphertext を復号化せよ．ただし，数字列から平文への逆変換関数は後述の関数 numtostring を使え．

```
DD :=
971602377879271972683993828475271284 45\
498538373913403577113306467649594367 11\
591703407500757079507769846783079881 66\
436532015569829063892652426693841104 87\
8601025;

N :=
538895607981329106921046915511034649 36\
802959626025412408880118195415973815 48\
473287204656043493800663757534977606 45\
711932907838570181520831781648891826 98\
06475731;

ciphertext :=
356173341936837581214588765635817779 48\
886867886408263016527251919216002281 40\
682631777960221709035310114726103780 53\
226914055689449701871105283280969329 89\
73447073
```

課題の背景

90 年代の終わり頃のゴルゴ 13 に，ゴルゴ 13 が新しい暗号の開発を支援するという逸話（373 話，最終暗号) がありました．暗号が通信の要であり，非常に高度なしかし簡素な数学を使っているという，さいとうたかおのいつもながらの先見性に感心しました．その作り話の元にもなっていた RSA 暗号についてどのような原理なのかを Maple で見ていきます．

公開鍵暗号システムは Whitfield Diffie と Martin Hellmann によって 1976 年にそのアイデアが公表されました．そこでは，暗号化の鍵と復号化の鍵を分けることで鍵配送問題を解決しています．受信者は，前もって「暗号化の鍵」を送信者に知らせておきます．この「暗号化の鍵」は盗聴者に知られてもかまいません．送信者は，その「暗号化の鍵」を使って暗号化して受信者に送ります．復号化できるのは「復号化の鍵」を持っている人 (受信者) だけです．こうすれば「復号化の鍵」を受信者に配送する必要がなくなります．対称暗号の鍵配送問題は，公開鍵暗号を使うことで解決するのです．

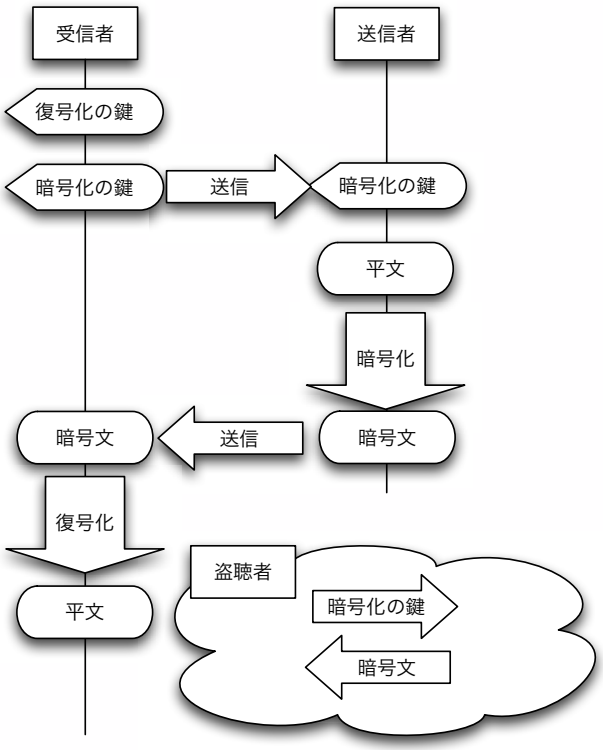


図 2.3: Diffie and Hellmann による公開鍵暗号のアイデア．

この公開鍵暗号のアイデア実現に必要な一方向関数は，1978 年に Ron Rivest, Adi Shamir, Leonard Adleman によって発表され，その RSA 暗号では素因数分解に非常に時間がかかることを利用しています[4]．RSA 暗号は

公開鍵： 数 E と数 N

秘密鍵： 数 D と数 N

[4]素因数分解などの RSA 暗号の基礎的な数学などに興味のある人は付記 1 を参照ください．

暗号化： 暗号文=平文^E mod N (平文を E 乗して N で割った余り)

復号化： 平文=暗号文^D mod N (暗号文を D 乗して N で割った余り)

です．そこで必要となる E,D,N などの鍵ペアは

N： 二つの素数 p,q から，N=p*q で求める．

L： p-1 と q-1 の最小公倍数 (lcm) を求める．

E： L と互いに素な E を求める．

D： E*D mod L =1 となる D を求める．

で生成します．

Maple の関連コマンド

公開鍵，暗号鍵を生成する Maple コマンドを見てみましょう．

N: RSA module, RSA モジュール　`N:=P*Q;`

L:　`L:=lcm(P-1,Q-1);`

厳密には L:=lcm(P-1,Q-1); ですが，P*Q を法とすると，全ての数が自分自身に戻るべき乗数は，n を任意の整数として n*(P-1 と Q-1 の最小公倍数)+1 と表すことができる．(P-1)*(Q-1) は，必ず (P-1 と Q-1 の最小公倍数) の倍数になるのでこちらを用いてもよい．

E:Encryption exponent, 暗号化指数

gcd(E,L)=1 となる E を求めるのですが，これはたくさんあります．以下のスクリプトで全てを表示することが可能です．

```
for i from 1 to L do
  if gcd(i,L)=1 then
    printf("%4d,",i);
  end if;
od;
```

実際は乱数発生器を用いたり，他の適当な素数を使ったりしているようです．

D:Decryption exponent, 復号化指数

E*D mod L=1 となる D を求める．これを Maple では L を法とする E の-1 乗として計算することが出来ます．Maple では D は微分を表す予約語なので DD とかにしておきます．

```
DD:=eval(1/E mod L);
```

現実的なより長い文字でできた平文 (plain text) を暗号文 (cipher text) にする場合を考えます．大きな素数を乱数発生器を使って作る例です．

```
> M1 := rand(10^80)();
> M2 := rand(10^80)();
> P := nextprime(M1);
> Q := nextprime(M2);
> E:= 2^16+1;  isprime(E); gcd(E,L);
```

nextprime は input の数字より大きな次の素数を返してくれる関数です．E をここでは 2 進数で 0 が並んだ数を使っています．

次に平文を大きな数の列に換える関数です．中身は分からんでもいいですが，文字の扱いに興味のある人はばらしてみてください．(Mike May, S. J. の”Using RSA” より)

リスト：文字の 2 桁の 16 進数表示への変換

```
twodigithex := a -> substring(convert(a+256,hex),2..3):
```

リスト：平文の数字列への変換関数

```
shortconverter := proc(messagestring)
  local stringofhex, lengthofmess, hexstring, i:
    #First we convert the ASCII string to a list of decimal equivalents
    #Then we convert the decimal numbers to hex equivalents
  stringofhex := map(twodigithex, convert(messagestring,bytes)):
print(stringofhex);
  lengthofmess := nops(stringofhex):
print(lengthofmess);
    #Now we concatenate the hex numbers
  hexstring := cat(seq(stringofhex[i],i=1..lengthofmess)):
    #Finally we convert the big number back to decimal
  convert(hexstring,decimal,hex):
end:
```

リスト : 数字列から平文への逆変換関数

```
numtostring := proc(bignum)
local hexstr1, listlength, declist, i:
#convert to a hex string
hexstr1 := convert(bignum,hex):
#make sure the hex string has even length
if (length(hexstr1) mod 2) = 1 then hexstr1 := cat(‘0‘,hexstr1) fi;
#compute the number of characters
listlength := length(hexstr1)/2:
#convert to a vector of decimal numbers
declist := linalg[vector](listlength);
for i from 1 to listlength do
declist[i] := convert(substring(hexstr1,2*i-1..2*i),decimal,hex);
od;
#convert the vector to a list, then to an ASCII string
convert(convert(convert(declist,list),bytes),name);
end:
```

実際の操作を試しておきます.

```
> plaintext := "Good evening Kids,  this afternoon we explore around RSA.";
> numberedText := shortconverter(plaintext);
> numtostring(numberedText);
```

解法のヒント

1. P:=3;Q:=11;E:=3; として暗号化関数，復号化関数を定義せよ．平文が 3 の場合に，暗号化した数と，復号化の結果を調べよ.
2. 前問の鍵を使って，N(=33) 以下の全ての自然数について暗号化した数を求めよ．また，復号化可能 (間違いが起こらない) であることを確かめよ.
3. twodigithex, shortconverter, numtostring の変数，関数名などを調整して，短い単語の数字列との変換・逆変換を確かめよ.
4. 冒頭の RSA 暗号の復号化を試みよ.

付記 1：RSA 暗号の基礎理論と関連する Maple コマンド

本文では省略した，RSA の原理となる素数の性質についてエッセンスだけを記しておきます．

先ず基本となる中学で習った（はずの），素数の復習です．自然数 p>1 は，二つの整数 1,p のみを約数とする場合，素数 (prime number) と言われる．素数でない自然数 a>1 は合成数 (composite number) と言われる．なお，1 は素数でも合成数でもない．また，素数 p が整数 a を割り切れれば，p は a の素因数 (prime divisor) と言われる．整数 a,b について gcd(a,b)=1 が成り立つとき，a と b は互いに素 (coprime) であると言われる．全ての正整数は素数の積で表現可能であり，これを素因数分解 (factorization) と言う．

暗号文 C が平文 M に戻る条件を導くと，暗号化と復号化の計算式は，

```
C = M^e (mod n)
M = C^d (mod n)
```

であるから，ここで mod の「大きくなる前に適宜割っていっても，最終的な余りは変わらない」という性質

```
{a * b} (mod n) = {(a (mod n)) * (b (mod n))} (mod n)
```

を使って，前者を後者に代入すると、

```
M = (M^e)^d (mod n)
  = M^(ed) (mod n)
  = M^(ed) (mod pq)
```

となる．この式を変形すると

```
 M^(ed)-M = 0 (mod n)
M*(M^(ed-1)-1)=0 (mod pq)
```

となる．この式がなりたつ条件は Fermat の小定理を使って求まり，RSA 暗号が成立する．

このあたりの詳しくて分かりやすい解説が

- http://pgp.iijlab.net/crypt/rsa.html:はやわかり RSA[5]
- http://www.maitou.gr.jp/rsa/rsa10.php:さるにもわかる RSA 暗号 (10. RSA 暗号の世界)[6]

にある.

素数絡みの Maple のコマンドのいくつかをまとめておきます.

mod, modp n を m で割ったあまりを求める関数.

```
n mod m;
modp(n,m);
```

ifactor ある数の素因数分解を求める関数

```
ifactor(91);
```

isprime ある数が素数なら true，違えば false を返す関数

```
isprime(101);
```

Power mod の「大きくなる前に適宜割っていっても，
最終的な余りは変わらない」という性質

```
 {a * b} (mod n) = {(a (mod n)) * (b (mod n))} (mod n)
```

を使ってベキ計算してくれる関数．mod と組み合わせて使う．
例えば，

```
time(numberedText^E mod N);
```

と

```
time(Power(numberedText,E) mod N);
```

とをくらべよ．通常のべき表示"^"のかわりに"&^"を使っても
Power と同じ効果が得られる．

nextprime ある数より大きな次の素数を返す関数.

```
nextprime(100);
```

lcm(Least Common Multiple) 最小公倍数を返す関数.

```
lcm(35,100);
```

gcd(Greatest Common Divisor) 最大公約数を返す関数.

```
gcd(35,100);
```

gcd の結果が 1，つまり 1 以外に共通に割り切れる数がない場合
二つの数は互いに素であるという.

[5]http://pgp.iijlab.net/crypt/rsa.html
[6]http://www.maitou.gr.jp/rsa/rsa10.php

ithprime i 番目の素数を返す関数.

Fermat の小定理 p を素数, a を整数とすると,
modp (a^p-a , p) =0
が成り立つ. さらに a と p とが互いに素であれば,
modp(a^(p-1), p) = 1
が成り立つ.

定理 自然数 n より小さい自然数 a が n と互いに素であるとき,
mod(ab,n)=1 となる b (0<b<n) がただ一つ存在する.

Euler の定理 (の特別な場合) 自然数 n(=p*q) より小さい自然数 M が n と互いに素であるとき,
mod(M^((p-1)*(q-1)),n)=1 (0<x<n)

Euclid 互除法

拡張 Euclid 互除法

付記 2：グーグル人材募集広告

直接 RSA 暗号に関係するわけではありませんが, 素数つながりで載せておきます.

グーグル、謎の人材募集広告–シリコンバレーのビルボードに
Stefanie Olsen (CNET News.com)
2004/07/12 08:40

先週、シリコンバレーの中心を走るハイウェー 101 沿いのビルボードに、複雑な数学の問題を載せた広告が現れた。(中略)

この広告には、「{e の値中の、最初の連続する 10 桁の素数} .com」("{first 10-digit prime found in consecutive digits e}.com.") と書かれている。この答えの「7427466391.com」にアクセスすると、そのウェブページにはさらに別の問題 (下記参照) が用意されているが、ここにも Google が関与していることを示すものは全くない。

この問題を解くと、Google の研究開発部門「Google Labs」へのページに辿りつく。このページには、「Google の構築を通して我々が学んだことの 1 つに、自分が何かを探しているとき、向こうも自分を探している場合のほうが見つかりやすいということがある。我々が探しているのは、世界最高のエンジニアであり、あなたこそその人なのだ」と書かれている。

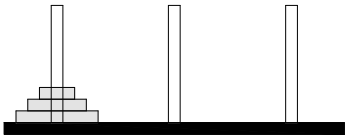
リスト：7427466391.com

```
Congratulations. You've made it to level 2. Go to
www.Linux.org and enter Bobsyouruncle as the login
and the answer to this equation as the password.
f(1)=7182818284
f(2)=8182845904
f(3)=8747135266
f(4)=7427466391
f(5)=__________
```

2.7 ハノイの塔

課題

3本の杭 (peg) があり，最初の杭には，おおきな円盤 (ring) から小さな円盤が下から順に積まれて塔のようになっている．適宜3本の杭に円盤を差しかえて，円盤を最初の杭から最後の杭へすべて移し換えなさい．ただし，一度に一つの円盤しか動かせない．また，小さい円盤の上に大きな円盤を置いてはいけない．



課題の背景

ハノイ (バラモン) の塔伝説

ベナレスにあるバラモン教の大寺院は世界の中心と記されており，そこに，杭が3本立った真鍮製の基盤がある．神はその一本に64枚の純金の円盤をはめた．僧侶たちは昼夜の別なくそれを別の杭に移し替える作業に専念している．そして移し替えが完了したとき，寺院もバラモン僧たちも崩壊し，この世が塵に帰るという．

『ハノイの塔』は1883年にフランスの数学者E. リュカ（Edouard Lucas）が考えたゲームです．リュカがこの物語を創作したのか，はたまたこの物語から発案したのかは不明[7]．

Maple の関連コマンド

ハノイの塔の円盤の動きを視覚化してみます．以下の一連のコマンドを打ち込んでください．

リスト：ハノイの塔の基本構成物 (基盤，杭，円盤) の描画．

```
with(plots):with(plottools):
base:=rectangle([-4,0],[25,-1],color=black):
peg1:=rectangle([0,0],[1,10],color=white):
```

[7] R. Douglas Hofstadter. Metamagical themas. Scientific American, 248(2):16-22, March 1983.

```
peg2:=rectangle([10,0],[11,10],color=white):
peg3:=rectangle([20,0],[21,10],color=white):
ring1:=rectangle([-3,0],[4,1],color=yellow):
ring2:=rectangle([-2,1],[3,2],color=yellow):
ring3:=rectangle([-1,2],[2,3],color=yellow):
display([ring1,ring2,ring3,peg1,peg2,peg3,base],axes=none,scaling=cons
trained);
```

リスト : 初期位置と円盤の大きさ.

```
position:=[[1,1],[1,2],[1,3]];
rw:=[3,2,1]:
```

リスト : 円盤の移動中の適切な描画.

```
move:=proc(input_list)
local rp1,rp2,rr1,tmp,width,i;
global rw;
rr1:=[];
for i from 1 to 3 do
  tmp:=input_list[i];
  width:=rw[i];
  rp1:=[-width+tmp[1]-1,tmp[2]-1];
  rp2:=[width+tmp[1]-1+1,tmp[2]];
  rr1:=[op(rr1),rectangle(rp1,rp2,color=yellow)]:
od:
[op(rr1),peg1,peg2,peg3,base];
end proc:
```

リスト : 円盤の移動位置のコントロール.

```
move_control:=proc(ring_n,ini,fin)
local tmp,ini_p,fin_p,step,i;
global position;
tmp:=[];
ini_p:=(ini[1]-1)*10+1;
fin_p:=(fin[1]-1)*10+1;
if (fin_p<ini_p) then step:=-1; else step:=1; end if;
for i from ini[2] to 12 do #UP
  position[ring_n]:=[ini_p,i];
```

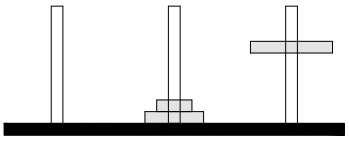
```
  tmp:=[op(tmp),display(move(position))]:
od:
for i from ini_p to fin_p by step do #Lateral
  position[ring_n]:=[i,12];
  tmp:=[op(tmp),display(move(position))]:
od:
for i from 12 to fin[2] by -1 do #DOWN
  position[ring_n]:=[fin_p,i];
  tmp:=[op(tmp),display(move(position))]:
od:
end:
```

リスト：円盤の移動の途中経過.

```
t1:=move_control(3,[1,3],[3,1]):
t1:=[op(t1),op(move_control(2,[1,2],[2,1]))]:
display(t1,insequence=true,axes=none);
```

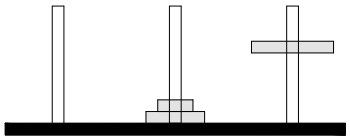
リスト：三つの円盤の最終段階までの表示．動画.

```
t1:=[op(t1),op(move_control(3,[3,1],[2,2]))]:
t1:=[op(t1),op(move_control(1,[1,1],[3,1]))]:
t1:=[op(t1),op(move_control(3,[2,2],[1,1]))]:
t1:=[op(t1),op(move_control(2,[2,1],[3,2]))]:
t1:=[op(t1),op(move_control(3,[1,1],[3,3]))]:
display(t1,insequence=true,axes=none);
```



リスト：外部ファイルへの動画の書きだし．QuickTime とかに食わせる.

```
plotsetup(gif,plotoutput=‘/Users/bob/Desktop/motion.gif‘);
display(t1,insequence=true,scaling=constrained,axes=none);
plotsetup(default);
```



解法のヒント

1. まず杭と円盤を抽象化して，配列に置き換えます．次のような peg 配列と top array を作れ．ここで配列の行が杭の番号を，配列の値がそこに刺さっている円盤の大きさを表す．また，top array は円盤が刺さっているもう一つ上の位置を示す．

$$\begin{bmatrix} 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, [4, 1, 1]$$

2. 円盤を動かす前後の杭の番号を受け取って，peg 配列，top array を操作する関数を作れ．この関数を MakeMove とすると以下のようなスクリプトによって移動が完了する．なお，ZeroArray は前問の初期配置を生成する関数である．

```
> max_disk:=3;
> ZeroArray(max_disk);
> MoveList:=[[1,3],[1,2],[3,2],[1,3],[2,1],[2,3],[1,3]]:
> n:=nops(MoveList):
> for i from 1 to n do
> MakeMove(op(MoveList[i]));
> end do:
> print(peg):
```

3. 一番小さな円盤の動きをよく観察し，どうすれば自動化できるかを考えよ．

4. 一番小さな円盤の動きと，他の円盤の動きをコントロールする関数を考えよ．それぞれの関数を SmallestToWhere, DiskToWhere とすると以下のスクリプトによって円盤移動の自動化が達成される．なお，PositionSmallest は一番小さな円盤がどこにいるかを憶えておく変数．また MakeMove2 は先程の MakeMove に PositionSmallest の操作を加えた関数．

```
> max_disk:=3;
> count:=0;
> ZeroArray(max_disk);
> PositionSmallest:=1;
> MakeMove2(op(SmallestToWhere()));
> while peg[3,max_disk]=0 do
> MakeMove2(op(DiskToWhere()));
> MakeMove2(op(SmallestToWhere()));
> end do:
> print("Congratulations!");
```

5. 【発展問題】一枚の金貨を別の杭に移すのに 1 秒かかるとすると，バラモン僧たちが 64 枚の金貨を移すのに何年かかるか．