ベジエ曲線とベジエマトリクスフォーム

SHIMURA Masato
JCD02773@nifty.ne.jp

2012 年 10 月 15 日

目次

概要

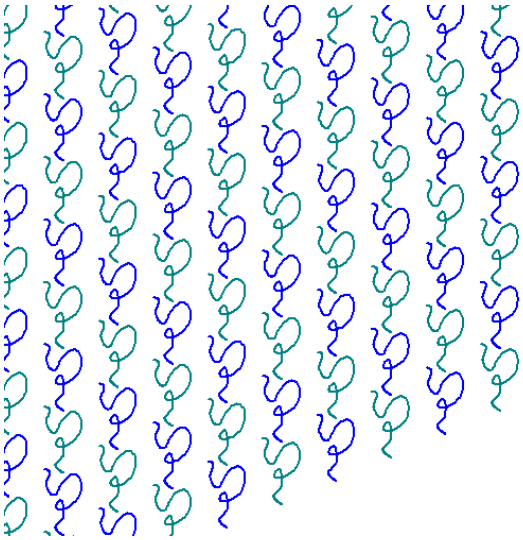
コンピューターで曲線を描くには数式による幾何曲線やツールでマウスやペンを用いたドローイングと並んでエンジニアリング由来のスプライン曲線やベジエ曲線もよく用いられている。マトリクスを用いるとベジエ曲線は実にシンプルなスクリプトで表現でき、 難解な数式や長いプログラムなしで自由にベジエ曲線を扱うことができる。simple is beautiful!

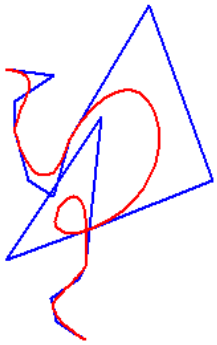
はじめに

次の左の図は北斎が「新形小紋帳」でデザインしたもので、右のような図を並べ重ねて作成する。数学の幾何曲線では図形は限定されるので、もっと自由に図形を描け、フォントデザインや CAD に用いられているスプライン曲線やベジエ曲線を用いることになる。スプライン曲線は Truetype フォントに、ベジエ曲線は Postscript フォントに用いられている。操り人形でいえばスプラインは手繰り、ベジエ曲線は糸繰りの様であるがベジエ曲線のほうが自由度が高いと思われる。

ベジエ曲線 (Cubic Bezier Curve) の計算は次の 4 行=4 つの関数にまとめることができた。plot を入れても 10 行足らずである。

1. `mat_bezier4=: 1 0 0 0,_3 3 0 0,3 _6 3 0,:_1 3 _3 1` NB.Cubic BezierMatrixForm
2. `calc_bezier4=: 3 : '(|: mat_bezier4 +/ . * y)&p. " 0 steps 0 1 20'`
3. `form_bezier =: 4 : '(}: tmp), L:0 }.{. L:0 tmp=.(- <: x)<\ y'`
4. `calculus_bezier4 =: [: ;("2)@,. calc_bezier4(L:0)@form_bezier4`





以下はベジエ曲線を学習した経過の記録である。

1　ベジエ曲線 (マトリクスフォーム)

スプライン曲線は第 2 次大戦後航空機産業で用いられ始めた。ベジエ曲線はシトロエン社のドン・カスティヨ、ルノー社のベジエが開発したが、直ぐには社外には出なかったようだ。*Pierre Bezier(1910-1999)* は生粋のパリっ子のエンジニアでルノー社に *42* 年在籍した。

ベジエ曲線はマトリクスフォームを用いると簡潔に表すことができる。マトリクスフォームを文献 *(1)* により提示してみよう。

1.1　Cubic Bezier Curve

4 点のポイントから *3* 次多項式を作成するので *Cubic* と呼ばれる。本稿ではポイント数で ***4*** として取り扱う。

コントロールポイント　*4* 点からなる。 P_0, P_3 は固定される。

$$ControlPoints: \begin{array}{ll} P_0 = & [x_0, y_0] \\ P_1 = & [x_1, y_1] \\ P_2 = & [x_2, y_2] \\ P_3 = & [x_3, y_3] \end{array}$$

ベルンシュタインの式　.

[*1]

$$J_{n,i} = \binom{3}{i} t^i (1-t)^{3-i}$$

[*1] Sergei Bernstein ユダヤ系ロシア人の数学者。パリで数学教育を受けロシアで教えた

t は媒介変数で $[0,1]$ の間の値をとる。パスカルの 3 角形が現れる。

$$BernsteinCubics: \begin{array}{ll} B_0(t) = & (1-t)^3 \\ B_1(t) = & 3(1-t)^2 t \\ B_2(t) = & 3(1-t)t^2 \\ B_3(t) = & t^3 \end{array}$$

ベジエの公式の導出 .

$$Bezier(t) = B_0(t)P_0 + B_1(t)P_1 + B_2(t)P_2 + B_3(t)P_3$$

$$Bezier(t) = (1-t)^3[x_0, y_0] + 3(1-t)^2 t[x_1, y_1] + 3(1-t)t^2[x_2, y_2] + t^3[x_3, y_3]$$

$$Bezier(t) = [x(t), y(t)]$$

$$x(t) = x_3t^3 + 3x_2t^2 - 3x_2t^3 + 3x_1t - 6x_1t^2 + 3x_1t^3 + x_0 - 3x_0t + 3x_0t^2 - x_0t^3$$

$$x(t) = x_0 + (-3x_0 + 3x_1)t + (3x_0 - 6x_1 + 3x_2)t^2 + (-x_0 + 3x_1 - 3x_2 + x_3)t^3$$

ベジエのマトリクスフォーム .

$$BezierMatrixForm: \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = AX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_0 \\ -3x_0 + 3x_1 \\ 3x_0 - 6x_1 + 3x_2 \\ -x_0 + 3x_1 - 3x_2 + x_4 \end{pmatrix}$$

$$x(t) = c_0 + c_1t + c_2t^2 + c_3t^3$$

$$y(t) = y_3t^3 + 3y_2t^2 - 3y_2t^3 + 3y_1t - 6y_1t^2 + 3y_1t^3 + y_0 - 3y_0t + 3y_0t^2 - y_0t^3$$

$$y(t) = y_0 + (-3y_0 + 3y_1)t + (3y_0 - 6y_1 + 3y_2)t^2 + (-y_0 + 3y_1 - 3y_2 + y_3)t^3$$

$$BezierMatrixForm: \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = AY = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ -3y_0 + 3y_1 \\ 3y_0 - 6y_1 + 3y_2 \\ -y_0 + 3y_1 - 3y_2 + y_3 \end{pmatrix}$$

$$y(t) = d_0 + d_1t + d_2t^2 + d_3t^3$$

ベジエの多項式と係数 .

$$Bezier(t) = [x(t), y(t)] = [c_0 + c_1t + c_2t^2 + c_3t^3\ ,\ d_0 + d_1t + d_2t^2 + d_3t^3]$$

*2

*2

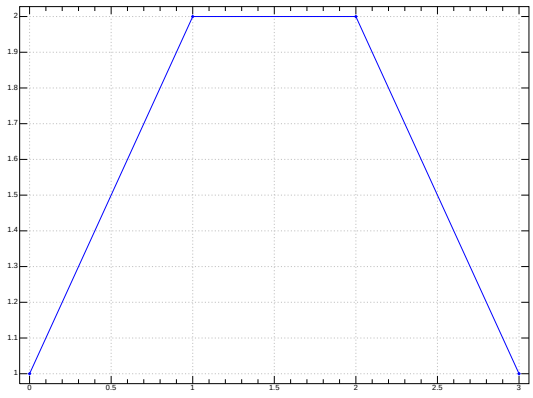
1.2 数値例と経過

J で描くからには簡潔なマトリクスフォームで直ちに多項式の係数を得て、*p.* で多項式の値 *(*ベジエ曲線*)* を求め、グラフィックスに渡すようにする

1. *Example 1* P_0, P_1, P_2, P_3

```
'line, marker'  plot {@|: L0
   pd 'eps c:/temp/besier_L0base.eps'
```

```
   L0
x y
0 1
1 2
2 2
3 1
```



2. *Bezier Matrix*

```
mat_bezier4=: 1 0 0 0,_3 3 0 0,3 _6 3 0,:_1 3 _3 1 NB.Cubic BezierMatrixForm
```

```
   mat_bezier4
 1  0  0 0
_3  3  0 0
 3 _6  3 0
```

- Bernstein の公式 (3 次式) からマトリクスの係数を求める

$$J_{n,i} = \binom{3}{i} t^i (1-t)^{3-i}$$

$$\begin{array}{lccc} J_{3,0}(t) = & (1-t)^3 & = & 1 - 3t + 3t^2 - t^3 \\ J_{3,1}(t) = & 3t(1-t)^2 & = & 3t - 6t^2 + 3t^3 \\ J_{3,2}(t) = & 3t^2(1-t) & = & 3t^2 - 3t^3 \\ J_{3,3}(t) = & t^3 & & \end{array}$$

- ベジエマトリクスフォーム (3 次=4 ポイント)。J 流に構成している

$$BezierMatrixForm : \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = AX = \begin{array}{c} \begin{array}{cccc} t^3 & t^2 & t & 1 \end{array} \\ \hline \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \end{array} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_0 \\ -3x_0 + 3x_1 \\ 3x_0 - 6x_1 + 3x_2 \\ -x_0 + 3x_1 - 3x_2 + x_3 \end{pmatrix}$$

```
_1  3 _3 1
```

3. 内積演算 *x,y* を同時に行う

```
   mat_bezier4 +/ . * L0
```

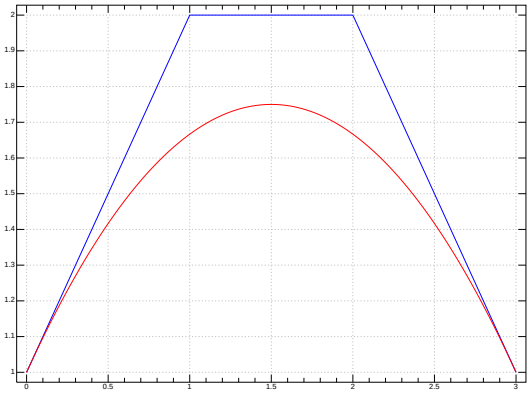
```
x(t),y(t)
0  1
3  3
0 _3
0  0
```

$$x(t) = 3t$$
$$y(t) = 1 + 3t - 3t^2$$

4. *x(t),y(t)* の多項式を作成し *(x,y)* での値を計算する。
   J の *p.* を用いる。*3* 次多項式になる。*t* は区間を *[0,1]* として *steps* で細分する。

```
   (0 0 3 &p. ,. 1 3 _3&p.  )steps 0 1 10
```

```
   0    1
0.03 1.27
0.12 1.48
0.27 1.63
0.48 1.72
0.75 1.75
1.08 1.72
1.47 1.63
1.92 1.48
2.43 1.27
   3    1
```

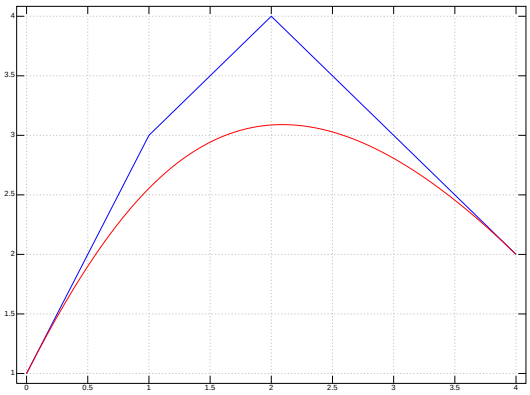


5. *Bezier* 曲線の特色
   - P_0, P_1, P_2, P_3 のうち両端の P_0, P_2 は固定
     中間の P_1, P_2 はコントロールポイント *(=*操り人形の紐*)* として用いる
   - P_0, P_1 をベクトルと見て中間点を取る。P_A
     P_1, P_2 の中間点を取る。P_B
     P_2, P_3 もベクトルと見て中間点を取る。P_C
   - P_A と P_B を結びその中間点 P_D をとる。
     P_B と P_C を結びその中間点 P_E をとる。
     ¶$_D$と P_E を結びその中間点 P_F をとるとこの *F* がアーチの頂上となる。*(*概ね *7* 割り強*)*

コントロールの P_1, P_2 は実例を沢山描き勘で覚える必要がある

- *Example 2*

```
   L1
0 1
1 3
2 4
4 2
```



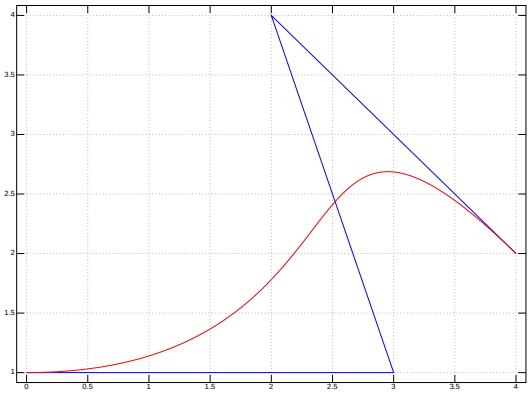
```
   mat_bezier4 +/ . *  L1
0  1
3  6
0 _3
1 _2
```

$x(t) = 3t + t^3$

$y(t) = 1 + 6t - 3t^2 - 2t^3$

- *Example 3*

```
    L2
0 1
3 1
2 4
4 2
```



```
   mat_bezier4 +/ . *  L2
  0  1
  9  0
_12  9
  7 _8
```

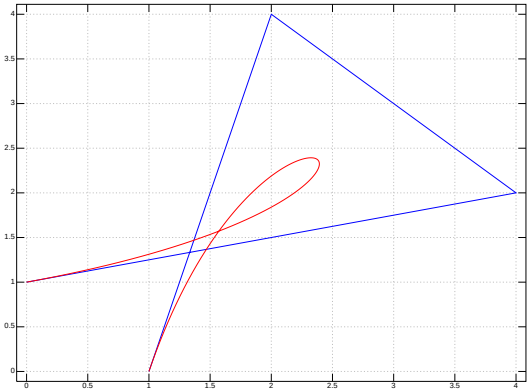
$x(t) = 9t - 12t^2 + 7t^3$

$y(t) = 1 + 9t^2 - 8t^3$

- *Example 4*

```
   L3

0 1
4 2
2 4
1 0
```



```
   mat_bezier4 +/ . *  L3
  0  1
 12  3
_18  3
  7 _7
```

$x(t) = 12t - 18t^2 + 7t^3$

$y(t) = 1 + 3t + 3t^3 - 7t^3$

1.3 Script

Cubic Bezier のコアは次の数行で簡潔に記述できる

```
mat_bezier4=:1 0 0 0,_3 3 0 0,3 _6 3 0,:_1 3 _3 1 NB.Cubic BezierMatrixForm
```

丁寧に計算過程の *Script* を書き下すと次のようになる。

```
calc_bezier4p1=: 3 : 0
tmp0=.|: mat_bezier4 +/ . * y
fx=. ({. tmp0)&p.
fy=. ({: tmp0)&p.
fx0 =. fx steps 0 1 20    NB. divede [0,1] for smooth Bezier curves
fy0 =. fy steps 0 1 20
fx0,.fy0
)
```

これは次のように簡潔にまとめる事ができる。$f(x), f(y)$ は同時計算。

```
calc_bezier4=: 3 : '(|: mat_bezier4  +/ . * y)&p. " 0 steps 0 1 20'
```

1.3.1 J Grammar

- ラミネート 最後のリストの前に *(,:)* を置き、横に積み重ねる。2×4 の行列にする

```
 3 _6  3 0 ,: _1 3 _3 1
```

- `+/ . *` 内積演算
- `|:` *transpose* 縦横を転置する。 多項式の係数を横行列にする

```
   |: mat_bezier4 +/ . * L0
0 3  0 0     NB.  fx= 3t
1 3 _3 0     NB.  fy=1+3t-3t^2
```

- `p.` 両項は多項式を当てはめ 右引数 *(t)* ごとの値を計算する。 fxy で $f(x), f(y)$ を同時に計算する
- *steps* 区間 `[0 1]` を *n* 分割する。*numeric.ijs* に入っている関数

```
   (|: mat_bezier4 +/ . * L0) &p."0  steps 0 1 10

  0    1
0.3 1.27
0.6 1.48
0.9 1.63
1.2 1.72
1.5 1.75
1.8 1.72
2.1 1.63
2.4 1.48
2.7 1.27
  3    1
```

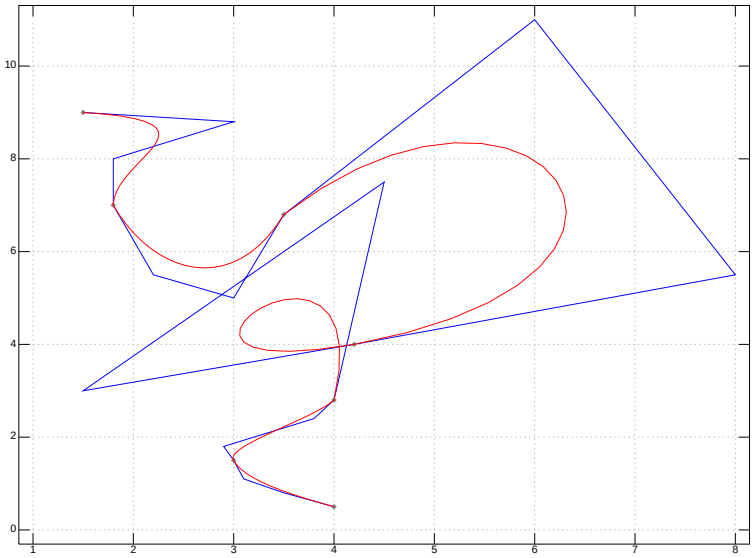
- ランク `" 1 0` は最初にベクトル方向、即ち *fx* を、次に *fy* を計算するように指定。

2 小紋の作図データの構成

1. ベジエの計算に与えるデータは方眼紙に描いた図から最初に括弧の $P_0, P_3, P_6, P_9, \cdots$ を採る。
   4 ポイントずつのデータを *1* 個ずつボックスでオーバーラップさせて構成する。各ボックスのデータの

最初と最後の *2* 個は固定される。入力はオーバーラップ分は除く 各象限は *7* 個ずつのデータで構成される

```
   RANOJI0
(1.5   9)  NB. P0
 3     8.8
 1.8   8
(1.8   7)  NB. P3
 2.2   5.5
 3     5
(3.5  6.8) NB. P6
 6    11
 8    5.5
(4.2   4)  NB. P9
 1.5   3
 4.5   7.5
(  4   2.8)
 3.8   2.4
 2.9   1.8
(  3   1.5)
 3.1   1.1
 3.5   0.8
(  4   0.5)
```



2. *J* に入力する。読みやすいように固定ポイントは括弧で囲む。
   RANOJI0 はボックスにしたときにプログラムでオーバーラップさせるので、入力ではオーバーラップさせる必要はない。

```
NB. Hokusai No29 らの字
RANOJI0=: (1.5 9),3 8.8 , 1.8 8 ,(1.8 7),2.2 5.5,3 5 ,:(3.5 6.8) NB. 0/1/2
RANOJI0=: RANOJI0, 6 11,8 5.5 ,(4.2 4),1.5 3 ,4.5 7.5 ,:(4 2.8) NB. (2)/3/4/5
RANOJI0=: RANOJI0, 3.8 2.4 ,2.9  1.8 ,(3 1.5), 3.1 1.1 , 3.5 0.8,: (4 0.5)   NB. 6/7
```

3. *J* の *calculus_bezier4* に与える *4* ポイントごとのデータを作成する。最下行と最上行はオーバーラップさせる

```
  4 form_bezier RANOJI0
+-------+-------+-------+-------+-------+-------+
|1.5   9|1.8   7|3.5 6.8|4.2   4|  4 2.8|  3 1.5|
|  3 8.8|2.2 5.5|  6  11|1.5   3|3.8 2.4|3.1 1.1|
|1.8   8|  3   5|  8 5.5|4.5 7.5|2.9 1.8|3.5 0.8|
|1.8   7|3.5 6.8|4.2   4|  4 2.8|  3 1.5|  4 0.5|
+-------+-------+-------+-------+-------+-------+
```

4. *form_bezier* の *Script*
   `form_bezier =: 4 : '(}: tmp), L:0 }.{. L:0 tmp=.(- <: x)<\ y'`
   J のボックスと *infix* 機能を用いてオーバーラップ無し (-3<\) の次の配列を作成し、オーバーラップを付加する

```
  _3<\i. 20 2
+---+-----+-----+-----+-----+-----+-----+
|0 1| 6  7|12 13|18 19|24 25|30 31|36 37|
|2 3| 8  9|14 15|20 21|26 27|32 33|38 39|
|4 5|10 11|16 17|22 23|28 29|34 35|     |
+---+-----+-----+-----+-----+-----+-----+
```

5. *Cubic Bezier* の計算。*4* 点計算の基本形とロングデータ用
   `calculus_bezier4 RANOJI0`

```
calc_bezier4=: 3 : '(|: mat_bezier4  +/ . * y)&p. " 0 steps 0 1 20'  NB. basement

calculus_bezier4=: 3 : ';("1) calc_bezier4 (L:0) 4 form_bezier y'    NB. long data
```

3　ベジエカーブの仕組み

3.1　パスカルの 3 角形と加藤の三角錐

パスカルの *3* 角形を *3* 角錐の各面に貼り付けて、次数毎に輪切りにすると次のような *3* 角形が現れる。アマチュアの数学愛好家加藤一郎氏が高校時代に思いついたと言う *3* 項展開から導いた「加藤の三角錐」を発表しておられる。これが絶対値でベジエのマトリクスフォームと一致する。赤字はパスカルの *3* 角形の表面には出てこない数値で全て *3* 方向から吸引されている。これがベルンシュタインの式を展開したベジエのマトリク

スフォームに現れる。

ポイント	次数	加藤の三角錐の断面	数式展開
3	*2*	1 2 2 1 2 1	$(a+b+c)^2 = a^2+b^2+c^2$ $+2ab+2ac+2bc$
4	*3*	1 3 3 3 6 3 1 3 3 1	$(a+b+c)^3 = a^3+b^3+c^3$ $+3a^2b+3a^2c+3ab^2$ $+3b^2c+3ac^2+3bc^2$ $+6abc$
5	*4*	1 4 4 6 12 6 4 12 12 4 1 4 6 4 1	$(a+b+c)^4 = a^4+b^4+c^4$ $+4a^3b+4a^3c+4ab^3$ $+4b^3c+4ac^3+4bc^3$ $+6a^2b^2+6a^2c^2+6b^2c^2$ $+12a^2bc+12ab^2c+12abc^2$
6	*5*	1 5 5 10 20 10 10 30 30 10 5 20 30 20 5 1 5 10 10 5 1	$(a+b+c)^4 = a^5+b^5+c^5$ $+5a^4b+5a^4c+5ab^4$ $+5b^4c+5ac^4+5bc^4$ $+10a^3b^2+10a^3c^2+10b^3c^2$ $+10b^3c^2+10a^2c^3+10b^2c^3$ $+20a^3bc+20ab^3c+20abc^3$ $+30a^2b^2c+30a^2bc^2+30ab^2c^2$

*3

3.2 Cubic Bezier 曲線の導出

2 次 (3 ポイント) ベジエ曲線　*2* 次曲線は *1* 次 *(2* ポイント*)* ベジエ曲線 *(*直線*)* を一個ずらして導出する。

$$\begin{aligned} B(t) = & \quad (1-t)((1-t)P_0 + tP_1) + t((1-t)P_1 + tP_2) \\ & = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \end{aligned}$$

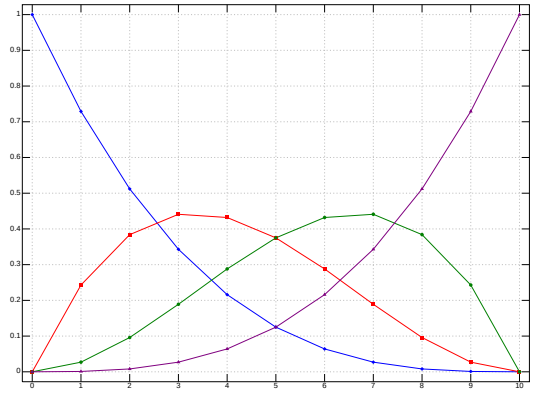
3 次 (4 ポイント) ベジエ曲線　*3* 次曲線は *2* 次 *(3* ポイント*)* ベジエ曲線を一個ずらして導出する。

$$\begin{aligned} B(t) = & \quad (1-t)((1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2) \\ & +t((1-t)^2 P_1 + 2t(1-t)P_2 + t^2 P_3) \\ & = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \end{aligned}$$

この作業を繰り返せば高次のベルンシュタイン関数からベジエ曲線を求めることができる

3.3 もう一つの Cubic Bezier 曲線の計算法

ベルンシュタインの式を展開したマトリクスフォームを用いないで直ちにベルンシュタインの式を計算する方法。



1. *Bernstein* の公式の *Script*

```
b3_0=: (^&3@-.);(3&* * ^&2@-.);(*&3@^&2 * -.);^&3  NB. Bernstein formula
NB.      (1-t)^3  3t(1-t)^2      3t^2(1-t)       t^3
```

J Grammar

- ベルンシュタイン関数を *4* 項に分離して各項ごとにボックスに入れ、同時計算を行う *tacit* 型関数定義
- /-./ $(1-t)$ の単項簡略型
- /&@/ 数字や関数を連結して一つの関数にする

*3 加藤氏は、4 項定理 $(a+b+c+d)^n$(正 4 面体になる) も展開しておられる。

2. *Bernstein* の公式に *t(= steps 0 1 10)* を投入 。先に *[0,1]* を計算する。縦に合計すると *1* になる。

```
 (steps 0 1 10),  ;"1 ,. b4_0 steps 0 1 10

0   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9 1  NB. steps 0 1 10
-------------------------------------------------------------------------
1 0.729 0.512 0.343 0.216 0.125 0.064 0.027 0.008 0.001 0  NB. (1-t)^3
0 0.243 0.384 0.441 0.432 0.375 0.288 0.189 0.096 0.027 0  NB. 3t(1-t)^2
0 0.027 0.096 0.189 0.288 0.375 0.432 0.441 0.384 0.243 0  NB. 3t^2(1-t)
0 0.001 0.008 0.027 0.064 0.125 0.216 0.343 0.512 0.729 1  NB. t^3
```

3. ベジエ *(3 次)* の簡潔な *Script*

```
b3=: 3 : 0
t0=.;"1 ,. b3_0 steps 0 1 10
+/ t0 * "0 1 y
)
```

4. ベジエの *4* ポイント P_0, P_1, P_2, P_3 を当てはめる。マトリクスフォームで計算した場合を右においた。

```
 L0; (b4 L0); calc_bezier40 L0
+---+--------+--------+
|0 1|  0    1|  0    1| NB. P1
|1 2|0.3 1.27|0.3 1.27|
|2 2|0.6 1.48|0.6 1.48|
|3 1|0.9 1.63|0.9 1.63|
|   |1.2 1.72|1.2 1.72|
|   |1.5 1.75|1.5 1.75|
|   |1.8 1.72|1.8 1.72|
|   |2.1 1.63|2.1 1.63|
|   |2.4 1.48|2.4 1.48|
|   |2.7 1.27|2.7 1.27|
|   |  3    1|  3    1| NB. P3
+---+--------+--------+
```

5. 長いデータに適用

```
b3_long=: 3 : ';"2,. b3 (L:0) 4 form_bezier_all y'
```

4　Cubic 以外のポイント数のベジエ曲線

4.1　3 ポイント (2 次) のベジエマトリクス

あまり複雑でない図形は *3* 次のベジエマトリクスで対応できる。

Berstein の公式から *2* 次式を求める

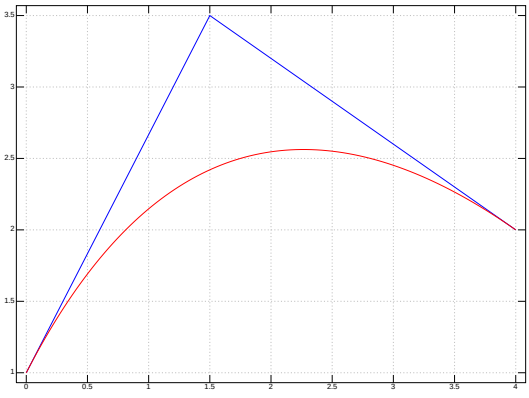
$$\begin{array}{rcl} (1-t)^2 & = & 1-2t+t^2 \\ 2t(1-t) & = & 2t-2t^2 \\ t^2 & & \end{array}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix}$$

```
     mat_bezier3
 1  0 0
_2  2 0
 1 _2 1
```

```
3 plot_bezier0  0 1,1.5 3.5,: 4 2
```

```
  pd 'eps c:/temp/besier_t3.eps'
```



4.2　4 次 (5 ポイント) のベジエ曲線

1. ベルンシュタインの公式

$$\begin{array}{rcl} (1-t)^4 & \rightarrow & 1-4t+6t^2-4t^3+t^4 \\ 4t(1-t)^3 & \rightarrow & 4t-12t^2+12t^3-t^4 \\ 6t^2(1-t)^2 & \rightarrow & 6t^2-12t^3+6t^4 \\ 4t^3(1-t) & \rightarrow & 4t^3-4t^4 \\ t^4 & & \end{array}$$

2. *Bernstein Formura* を直接に *t* を先に計算する方法

```
b5_0=: (^&4@-.);(4&* * ^&3@-.);(*&6@^&2 * ^&2@-.);(*&4@^&3  * -. );^&4
        (1-t)^4  4t(1-t)^3     6t^2(1-t)^2         4t^3(1-t)       t^4
```

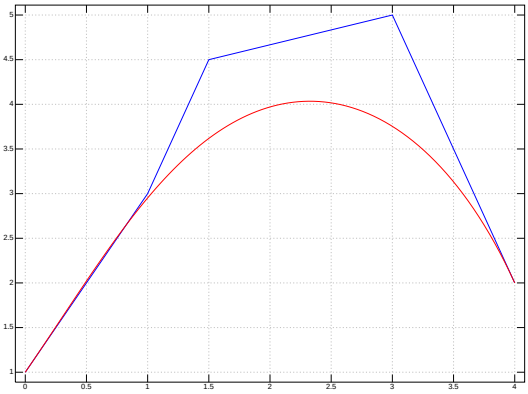
3. *4* 次のマトリクスフォーム

```
   mat_bezier5
 1   0   0  0 0
_4   4   0  0 0
 6 _12   6  0 0
_4  12 _12  4 0
 1  _4   6 _4 1
```

4. 2 の方法の比較

```
   (b5 L0,4 0),.  (|: mat_bezier5 +/ . * L0, 4 0)&p. " 0 steps 0 1 10
  0      1   0      1    NB.  P0
0.4 1.3401   0.4 1.3401
0.8 1.5616   0.8 1.5616
1.2 1.6681   1.2 1.6681
1.6 1.6656   1.6 1.6656
  2 1.5625   2   1.5625
2.4 1.3696   2.4 1.3696
2.8 1.1001   2.8 1.1001
3.2 0.7696   3.2 0.7696
3.6 0.3961   3.6 0.3961
  4      0   4      0   NB.  P3
```

```
5 plot_bezier0  0 1, 1 3, 1.5 4.5,3 5 ,: 4 7
   pd 'eps c:/temp/besier_t5.eps'
```



4.3 2 ポイント (1 次式) のベジエ

2 ポイントは直線になる。わざわざ直線をベジエで書くのは、直線区間を一筆書きで描くのに都合が良いからである。

1. ベルンシュタインの式

$$(1-t)+t=\begin{pmatrix}1 & 0\\ -1 & 1\end{pmatrix}\begin{pmatrix}t\\ 1\end{pmatrix}$$

2. マトリクス フォーム

```
   mat_bezier2
 1 0
_1 1
```

4.4 5 次 (6 ポイント) のベジエ曲線

Bernstein の公式と加藤の三角錐の美しい内部構造が明らかになったところで *5* 次のベジエのマトリクスフォームを作成してみよう。

1. ベルンシュタインの公式

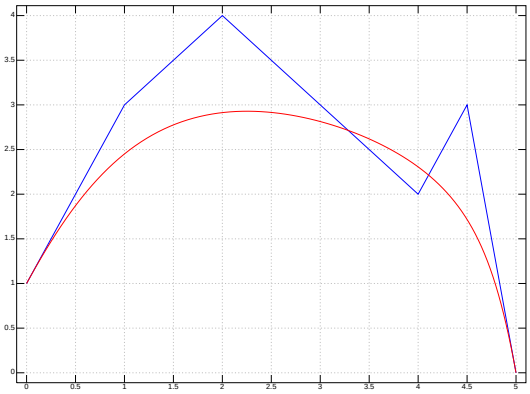
$$\begin{array}{ccc}(1-t)^5 & \rightarrow & 1-5t+10t^2-10t^3+5t^4-t^5\\ 5t(1-t)^4 & \rightarrow & 5t-20t^2+30t^3-20t^4+5t^5\\ 10t^2(1-t)^3 & \rightarrow & 10t^2-30t^3+30t^4-10t^5\\ 10t^3(1-t)^2 & \rightarrow & 10t^3-20t^4+10t^5\\ 5t^4(1-t) & \rightarrow & 5t^4-5t^5\\ t^5 & & \end{array}$$

2. 5 次のマトリクスフォーム

```
   mat_bezier6
  1   0   0   0  0 0
 _5   5   0   0  0 0
 10 _20  10   0  0 0
_10  30 _30  10  0 0
  5 _20  30 _20  5 0
 _1   5 _10  10 _5 1
```

3. plot

```
6 plot_bezier0  L1,4.5 3,:5 0
   pd 'eps c:/temp/besier6t.eps'
```



4.5 色々な次数の Script

1. マトリクスフォーム *(1,2,3,4,5 次)*

```
NB. ---0.----BezierMatrixForm-------------------------
mat_bezier4=: 1 0 0 0,_3 3 0 0,3 _6 3 0,:_1 3 _3 1 NB.Cubic BezierMatrixForm
mat_bezier3=: 1 0 0 , _2 2 0,: 1 _2 1
mat_bezier2=: 1 0,: _1 1  NB. bezier2 is linear & divide[0 1] by steps
NB. ------5/6 is Kato's triangle------------------------------
mat_bezier5=: 1 0 0 0 0 ,_4 4 0 0 0, 6 _12 6 0 0 ,_4 12 _12 4 0,:1 _4 6 _4 1
mat_bezier6=: 1 0 0 0 0 0,_5 5 0 0 0 0 , 10 _20 10 0 0 0 ,: _10 30 _30 10 0 0
mat_bezier6=: mat_bezier6, 5 _20 30 _20 5 0,: _1 5 _10 10 _5 1
```

2. 基本のベジエスクリプト

```
NB. ----1 calc single 2/3/4/5/6 points bezier---------------
NB. midify steps yourself e.g. stepd (0 1), 100
calc_bezier4=: 3 : '(|: mat_bezier4  +/ . * y)&p. " 0 steps 0 1 20'
calc_bezier3=: 3 : '(|: mat_bezier3  +/ . * y)&p. " 0 steps 0 1 20'
calc_bezier2=: 3 : '(|: mat_bezier2  +/ . * y)&p. " 0 steps 0 1 20'
calc_bezier5=: 3 : '(|: mat_bezier5  +/ . * y)&p. " 0 steps 0 1 20'
calc_bezier6=: 3 : '(|: mat_bezier6  +/ . * y)&p. " 0 steps 0 1 20'
```

3. データフォームつきの汎用のベジエスクリプト

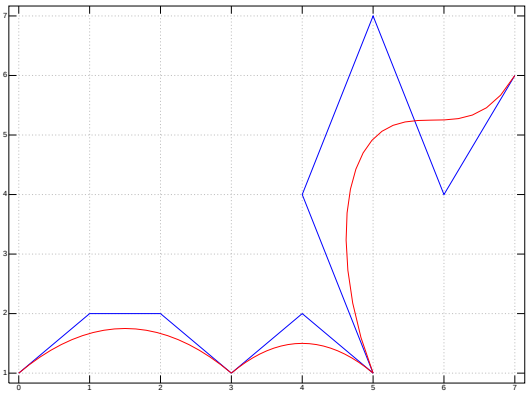
```
NB. --2---main calc Bezier long data-------------------------------
calculus_bezier4=: 3 : ';("1) calc_bezier4 (L:0) 4 form_bezier y'
```

```
calculus_bezier3=: 3 : ';("1) calc_bezier3 (L:0) 3 form_bezier y'
calculus_bezier2=: 3 : ';("1) calc_bezier2 (L:0) 2 form_bezier y'
calculus_bezier5=: 3 : ';("1) calc_bezier5 (L:0) 5 form_bezier y'
calculus_bezier6=: 3 : ';("1) calc_bezier6 (L:0) 6 form_bezier y'
```

4.6 異なる次数ベジエ曲線の混合

ベジエ曲線は基本は *Cubic(4* ポイント*)* で描くが、図形の途中で次数を変えたり、直線を用いたりすることがある。一筆書きが切れると、別のラインで描画関数も別に指定しなければならないのでなるべく一筆書きにしたい場合に *mixed form* が便利である

- .
  L4=: L0,4 2,5 1,4 4, 5 7,6 4,:7 6
  4 3 5 plot_bezier_mix L4



- ミックスフォームの作成

```
  4 3 5 form_bezier_mix L4
+---+---+---+
|0 1|3 1|5 1|
|1 2|4 2|4 4|
|2 2|5 1|5 7|
|3 1|   |6 4|
|   |   |7 6|
+---+---+---+
```

- *mix_form* 用の関数一覧
  1. *form_bezier_mix*
  2. *calc_bezier_mix*

3. *calculus_bezier_mix*
4. *plot_bezier_mix*

付録 A　isigraph でベジエ曲線を描く

北斎の江戸小紋は *C.Reiter* の ***dwin*** を用いた。早くから開発されているので原点は左下である。*J* はバージョン *5* から仕様を変更し、原点を左下から左上へ移した。小紋の座標は左下に取っているので *J* の *isigraph* で描くと天地が逆になる。蝙蝠のようにぶら下った図を鳥が木に止まったように補正する *Script* を作成した

1. *isigraph* の概要を整理しておこう
   - 原点 *(*始点*)* は左上 *(0,0)*
   - 終点は *(1000,1000)* 位*?*で右下。
   - マイナスは描けない。プラスに位置を補正する。
2. *isigraph* のキャンバス (200×200) を作る。*Form Editor* で簡単に作れるが、次のように手で書いても良い。

```
HOKUSAI0=: 0 : 0
pc hokusai0;
xywh 0 0 200 200;cc hokusai0 isigraph;
pas 6 6;pcenter;
rem form end;
)
```

3. *isigraph* の絵筆の例

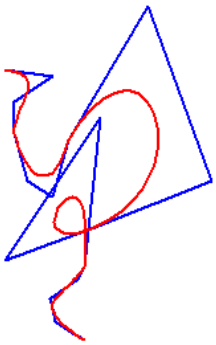
```
draw_hokusai=: 4 : 0
'tmp0 tmp1'=.  x fit_bezier_screen  y
NB. ----------------------------
glrgb 0 0 255
glpen 2,PS_SOLID
gllines   , tmp0
glrgb 255 0 0
glpen 2,PS_SOLID
gllines  , tmp1
glpaint ''
)
```

4. 逆立ちは回転で行う。*x* 軸を回すので回転行列を作り、$1p1 = \pi$ 回すとマイナスが現れる

```
rotx0=: (1:,0:,0:), (0:,cos, sin),:(0:,-@sin,cos)
rotx=: 3 : ' }:"1 (y,.1) +/ . * rotx0 1p1'
```

$$\begin{matrix} 1 & 0 & 0 \\ 0 & cos & sin \\ 0 & -sin & cos \end{matrix}$$

5. マイナスをプラスにし、スクリーンの中央になるように補正する



```
compose_bezier_rotx=: 4 : 0
NB. compare bezier origin
tmp0=. rotx y
tmp1=. rotx x calculus_bezier y
sign=. * +/  min=. ({:<./ tmp1 ) ,{:<./ tmp0
 if. _1 = sign do.
   min=. <./ min
   tmp0=. (0,- min) +"1 tmp0
   tmp1=. (0,- min)+"1 tmp1
end.
tmp0;tmp1
)

minmax=:  <./ ; >./

fit_bezier_screen=: 4 : 0
'tmp0 tmp1'=: x compose_bezier_rotx y
NB. SCREEN=: 200 200
tmp2=. minmax tmp1
SIZE=.>./ (>{: tmp1) - >{. tmp1
TIMES=: +: >.  100 % SIZE
ROCATION=:2# 100- TIMES
(ROCATION +"1 TIMES * tmp0); ROCATION +"1 TIMES * tmp1
)
```

```
4 hokusai0_run RANOJI0
```

References

http://kinetigran.com/mck/Calculus/BezierCurves/BezierMatrix.html