JAPLA 研究会資料 2020/10/17

Processing プログラムをトライしてみよう
－J ユーザから見た使い易さ、難さの体験－

西川　利男

はじめに

前回の JAPLA 例会で志村正人氏より「Proccessing による数学－ベクトルの理解」の記事を元に、J におけるベクトルなどについて報告された[1]。

[1] 志村正人「プログラミングで理解する数学(ベクトル)－日経ソフトウェア 2020/09 号」JAPLA 研究会, 2020/9/10.

実は私自身、Processing システムは以前から興味を持っていて、小さなメモを JAPLA の会で報告したことがある[2]。　今回はあらためて、もう少していねいな紹介と J ユーザから見た使い易さ、難さについて述べたいと思う。

[2] 西川利男「スマホ向け GUI プログラミング言語－Processing 言語のすすめ」JAPLA 研究会資料 2016/3/12

１．Processing とは

Processing は MIT のメディアラボで Benjamin Fry と Casey Reas を中心に始まったオープンソースのプログラミング言語で、広く教育分野で使われている。

Processing は私に言わせれば、プログラミング言語というよりプログラミング環境と言ったほうが良い。大きな特徴は大まかに次のように把握している。

・計算するのと同じレベルで、すぐグラフィックス画像表示ができる。

　結果を画像で見る立場からは、J よりずっと手軽で良い。一方、J ではプログラミングは *.ijs で、実行は*.ijx であり、グラフィックスは plot で行うか、あるいは gl2 などであらためてプログラミングすることが必要になる。

・Processing は環境を提供するだけで、作業言語は Java か Python を使うことになる。

Java 版＝C 言語ライクで、データ、関数定義などすべてについて型宣言が必要である。また行末のセミコロン(;)が必要でありうっとうしい。

Python 版＝BASIC ライクで、行末のセミコロン(;)も型宣言は不要であり、手っ取り早い。

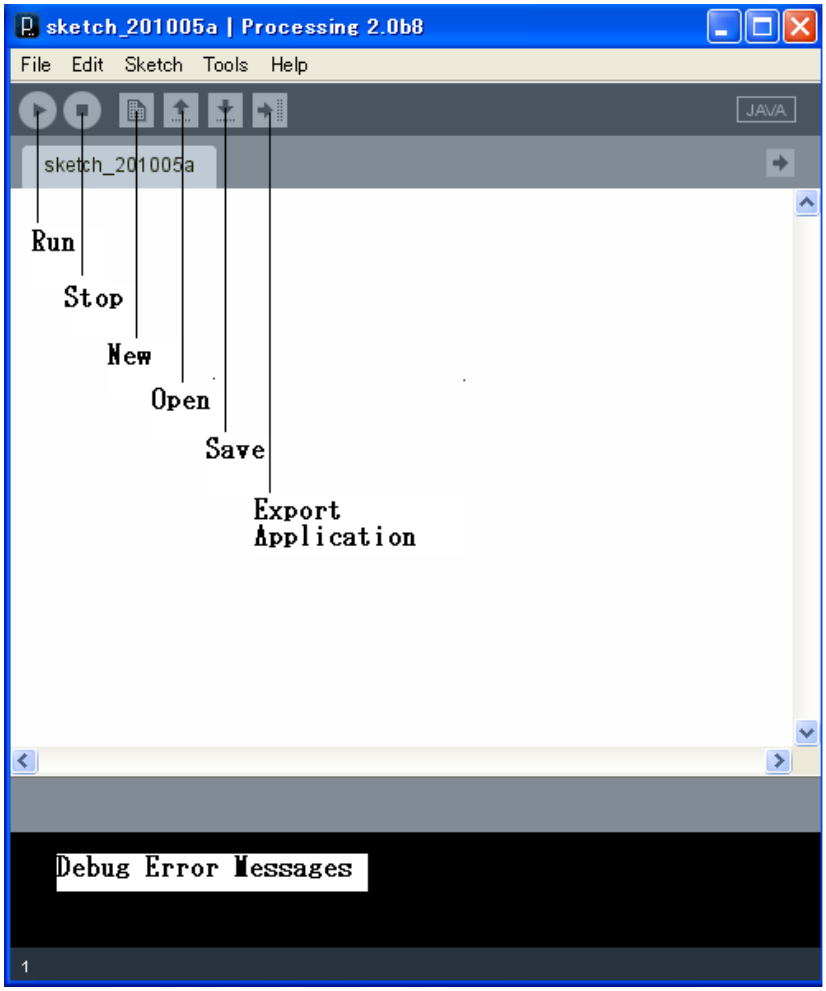
今回、私は手持ちの Java 版 Processing を用いて、次の解説書によりおこなった[3]。

[3]　西田竜太「遊んで作るスマホゲームプログラミング for Android」秀和システム(2014).

２．Processing の立ち上げとプログラミング、実行、保存などの実際

Processing のアイコンを起動する。

すると次の画面が現れる。



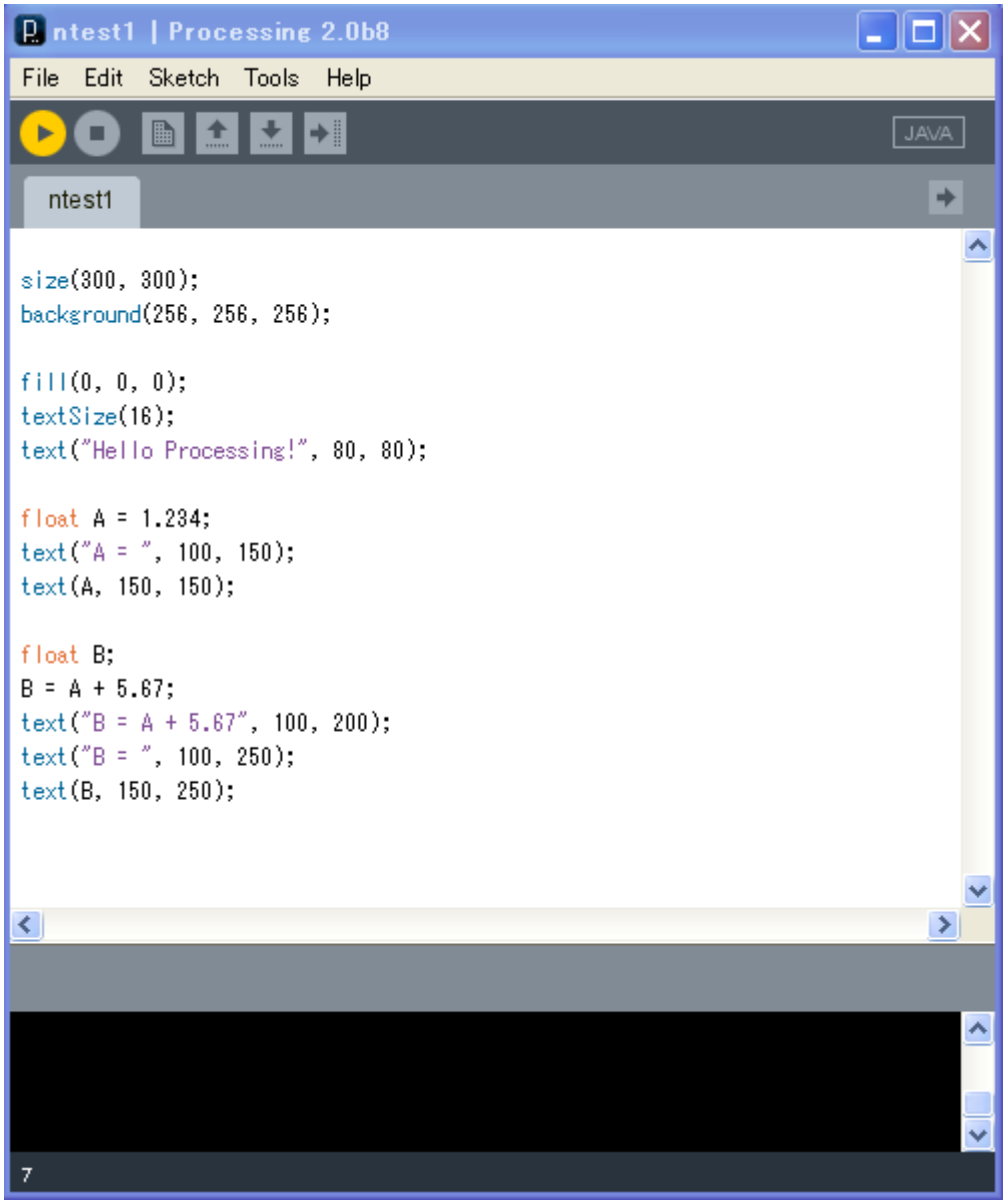
Prcessing の作業ファイルは Sketchbook と呼ばれ、***.pde というファイル名で行われる。プログラミング作業は上部のツールバーでも出来るが、その下のアイコンを使うほうがプログラムを入力便利である。[New]で新規作成、プログラムを入力し行ったら、[Run]で実行する。エラーメッセージは下の黒い領域に示される。完成したら[Save]でファイルを保存する。既存ファイルは[Open]で開く。

[Run]で実行すると、直ちに新しく別のグラフィックス画面現れる。J と違うところはグラフィックスはもちろん、計算結果の表示などの文字列もこの画面上で行われる。

たしかに、この点は J システムより手軽である。

３．Processing の簡単なプログラミングと実行

まずは簡単な Processing のプログラムを ntest1.pde という名前で作ってみよう。



まず、左から３番目のボタン New を押してはじめる。あるいは、起動時に開く sketchbook と表示のまま、Processing のコードを入力していく。

コーディングを一行ずつ説明する。プログラムの最初にには

```
size(300, 300);
```

として、グラフィック画面の大きさをきめる。文字もグラフィックスも同じ画面で表示する。行末のセミコロン(;)が必要である。Ｊユーザには、忘れがちである。

つづいて、

```
background(256, 256, 256);
```

として、画面の背景色をきめる。決めないとグレイになってしまう。

この２行はProcessingではいつも必要である。まとめてsetupとすることが良くおこなわれる。

ここでは最初のプログラムとして、文字列のみの表示をおこなう。

```
fill(0, 0, 0) ;
textSize(16);
```

では、書く文字を黒として、文字の大きさもきめる。

```
text(“Hellow Processing !”, 80, 80);
```

では、上のメッセージを、画面上の位置(80, 80)から書き出すことをしめす。

いうまでもなく、textでは、グラフィックス画面上に文字列を書く。

このように、文字列でも表示する位置を決めてやらねばならない。

```
float A = 1.234;
```

で、浮動小数点数としてAをきめて、値をいれる。整数のときはintとする。

javaではいちいち型宣言が必要なので、Jユーザにとってはわずらわしい。

```
text(“A =”, 100, 150);
```

として、位置をきめて、文字列を書き出す。

```
text(A, 150, 150);
```

として、数値を書く。

つぎの簡単な計算は、説明は不要であろう。

```
float B;
B = A + 5.67;
```

この処理過程を画面上に文字列で示す。

```
text(“B = A + 5.67”, 100, 200);
```

この計算結果を、画面上にかきだす。

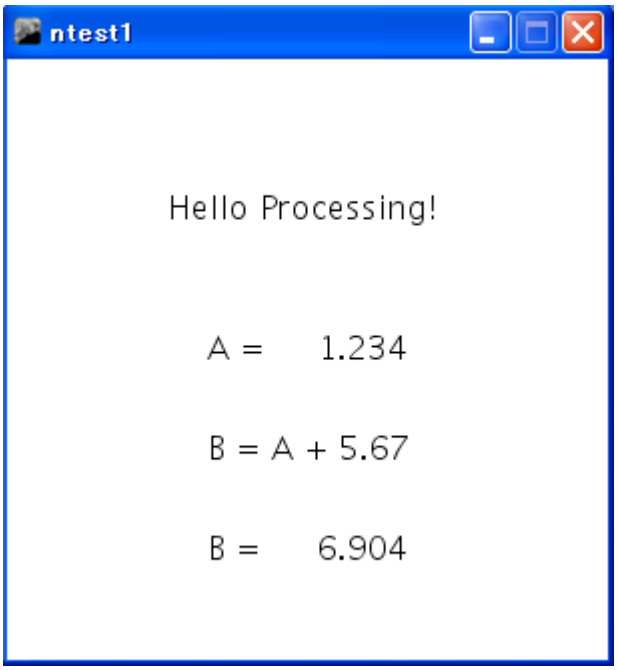
```
text(“B =”, 100, 250);
text(B, 150, 250);
```

以上のようにProcessingのプログラムが出来た。実行は左手の黄色三角をクリックしてすると、ただちに別の画面が現れ表示される。当然、文法の間違い、セミコロンの忘れなどのときは実行されず、エラーメッセージが、下の黒い領域に表示される。

プログラムにエラーがなければ、下向き矢印のボタンを押して、セーブする。

ここでは、ntest1.pdeという名前でセーブした。

実行した画面は次のようになる。



４．Processingのグラフィックス－日の丸とサインカーブ

簡単なグラフィックスとして、長方形(rect)や円(ellipse)を使って、日の丸の旗を描く。計算のグラフとしては、サインカーブを描く。その値も示してみた。

Processingグラフィック画面では、位置座標の取り方が、画面の左上が(0,0)で右下がsizeで指定した値になっている。特にタテ座標が上から下になっているのが、数学の座標系とちがっていて、混乱しやすくて、私にとってはいやだ。

```
// ntest2.pde - graphics
size(400, 400);
background(256, 256, 256);

rect(50, 100, 50, 50);    // Hinomaru
fill(255, 0, 0);          // Red
ellipse(75, 125, 20, 20); // Hinomaru
line(20, 200, 380, 200); // x-axis
line(200, 20, 200, 380); // y-axis

// sine curve graphics
fill(0, 0, 255);
float[] X = new float[100];
int[] XI = new int[100];
float[] Y = new float[100];
int[] YI = new int[100];
for (int i=0; i<13 ; i++){
  X[i] = 15*i;                          // X-grids in 15 degree
   XI[i] = round (X[i]);                     // required integer values for line
function
  text(XI[i], (28*i+20), 320);          // writes x-values
  Y[i] = (20 * sin ((3.14/180)*X[i])) ; // Y-sine values in radian
   YI[i] = round (2 * Y[i]);                 // required integer values for line
function
  text(YI[i], (28*i+20), 340);          // writes y-values
}

// draw graphics - attention to negate y-values ! x-positive region
for (int i=0; i<12 ; i++){
```

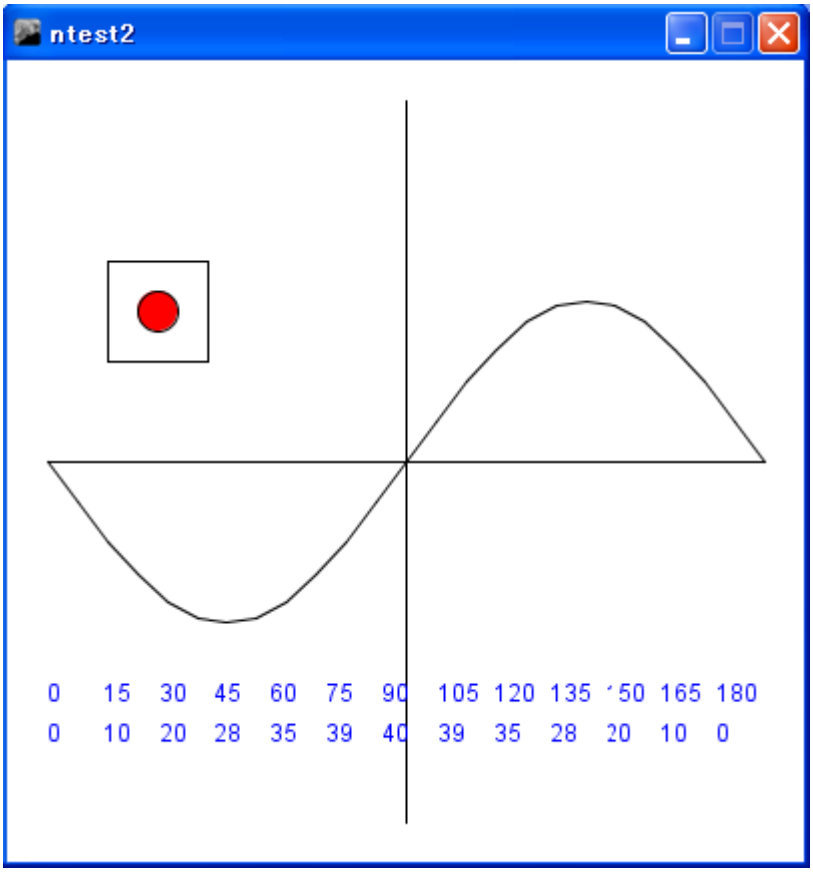
```
line ((200 + (XI[i])), (200 - (YI[i]*2)), (200 + (XI[i+1])), (200 - (YI[i+1]*2)));
}

// draw graphics - attention to negate y-values ! x-negative region
for (int i=0; i<12 ; i++){
line ((20 + (XI[i])), (200 + (YI[i]*2)), (20 + (XI[i+1])), (200 + (YI[i+1]*2)));
}
```

５．Processing のグラフィックス「ベクトル和とベクトル数学

志村氏が参照したベクトルを使って、日経グラフィックスの記事、数学ーベクトル和と内積」のプログラムを私もやってみた。

課題を Processing のプログラムは次のように なる。




```
// vector2

void setup(){
  size(600, 600);
  background(255, 255, 255);
}

int AX = 2;
int AY = 4;
int BX = 5;
```

```
int BY = -1;

void draw(){
  stroke(0, 0, 0);
  strokeWeight(1);
  line(10, 400, 400, 400);    // x-axis
  line(100, 50, 100, 500);    // y-axis
  stroke(255, 0, 0);          // Red
  strokeWeight(4);
  line(100, 400, (100 + 40 * AX), (400 - 40 * AY));
  stroke(0, 255, 0);          // Green
  line(100, 400, (100 + 40 * BX), (400 - 40 * BY));
  stroke(0, 0, 255);          // Brue
  line(100, 400, (100 + 40 * (AX + BX) ), (400 - 40 * (AY + BY)) );

  stroke(0, 0, 0);            // Black
  strokeWeight(1);
  line((100 + 40 * AX), (400 - 40 * AY), (100 + 40 * (AX + BX) ), (400 - 40 *
(AY + BY)) );
  line((100 + 40 * BX), (400 - 40 * BY), (100 + 40 * (AX + BX) ), (400 - 40 *
(AY + BY)) );

  fill(0, 0, 0);
  textSize(18);
  text("VA = ("+ AX + ", " + AY + ")" , 110, 70);
  text("VB = ("+ BX + ", " + BY + ")" , 110, 100);
  text("VA + VB = ("+ (AX + BX) + ", " + (AY + BY) + ")" , 110, 130);

  int IP = (AX * BX) + (AY * BY);
  text("Inner Product = " + (IP) , 110, 170);

  float AA = atan2(AY, AX);
  float BB = atan2(BY, BX);

  float P1 = AA - BB;
  float P2 = (180f / PI) * P1;
```

```
  text ("Angle of VA nad VB = " + P2    , 110, 200);
  text ("deg", 400, 200);
}
```

ベクトルVA, VBを作って、その和VA+VBを作る。さらにベクトルVAとVBとの内積とベクトルVAとVBとの成す角ををを計算した。

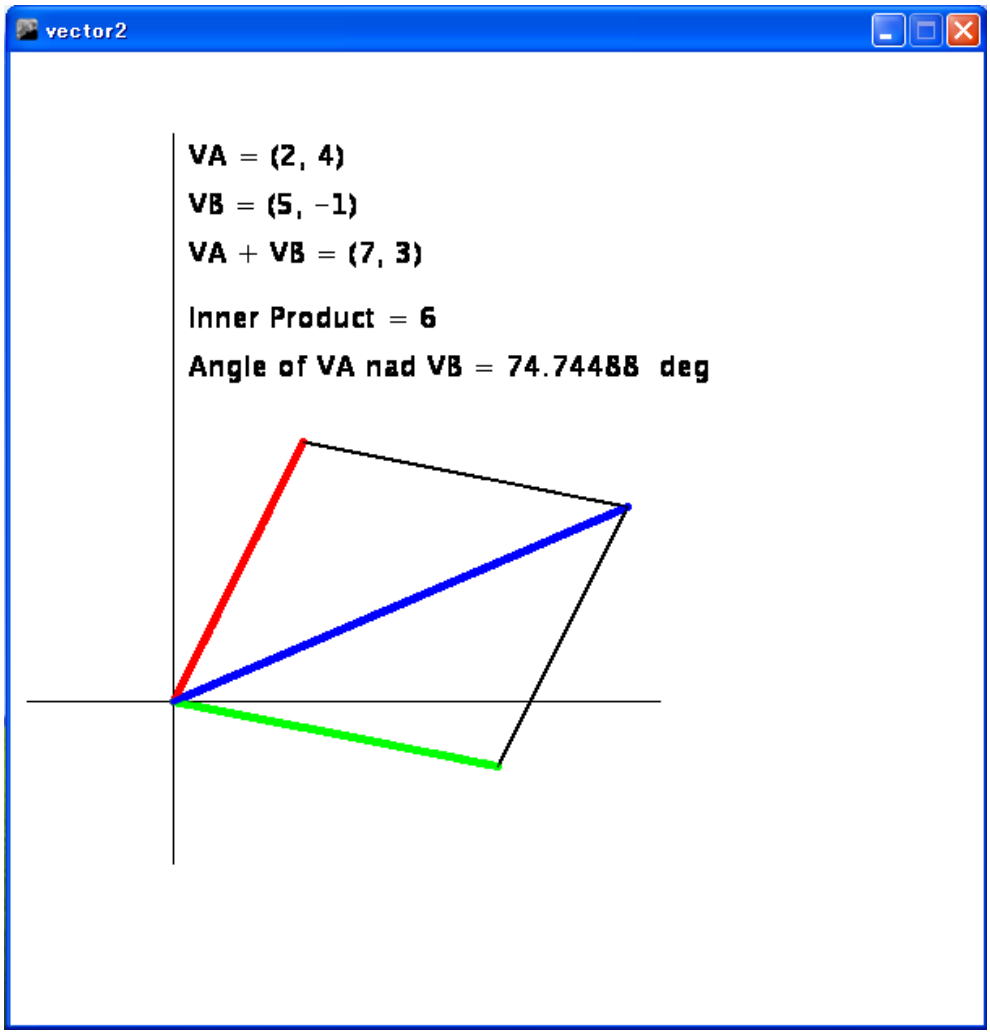
計算式は次のようになるが、実際はarctan2 を使って簡単に行われる。
なお、arctan2などの数学関数の使用方法は上部のツールバー [Help]から[Reference]において、例をあげてくわしく説明されている。

$$c = a \cdot b = |a| \cdot |b| \cdot \cos\theta,$$
$$(cx, cy) = (\vec{a}x \cdot \vec{b}x, ay + by)$$
$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{|a| \cdot |b|} = \frac{ax \cdot bx + ay \cdot by}{\sqrt{ax^2 + ay^2} * \sqrt{bx^2 + by^2}}$$

グラフィックスによる表示と数値計算による計算をおこなった。

Jプログラミングと違うところは、数値計算の結果表示も文字列として、同じグラフィックス画面に表示しなくてはならない。

Processing の実行結果、グラフィックス画面は次のようになる。



６．Processing のグラフィックス画像ファイルの貼付けとマウス操作

Processing の便利な特徴として、簡単に画像ファイルを貼り付け表示することができる。
さらにマウス操作によって、画像ファイルを自由に動かすプログラム「アジサイとルービック」を作ってみた。

```
// ngraph1.pde
// moving rubik with static ajisai

PImage rubik, ajisai;
float rubikX = 0.0f;
float rubikY = 0.0f;
```

```
void setup() {
  size(800, 800);
  rubik = loadImage("rubik.png");
  ajisai = loadImage("ajisai.png");
}

void draw() {
  background(255, 255, 255);
  image(ajisai, 400, 500);

  imageMode(CENTER);
  rubikX = rubikX * 0.9 + mouseX * 0.1;
  rubikY += (mouseY - rubikY) * 0.1f;
  image(rubik, rubikX, rubikY);
}
```