Knoppix/Math 徹底攻略, 有馬の巻 A

Knoppix/Math プロジェクト著

2005.08.12 版

Knoppix/Math book のための第一回目の強化合宿を, 2005 年 8 月 8 日から, 8 月 10 日の期間敢行いたしました. まだ書きなぐり状態の下書で, 途中までしか書いてありませんが, とりあえず “Knoppix/Math 徹底攻略, 有馬の巻 A” として強化合宿の成果を公開します. コメントや質問大歓迎です. 中川さん, ponpoko さん 著の下書も追って公開されるでしょう.

Knoppix/Math/Book プロジェクト

目 次

第1章　KSEG で遊ぶ平面幾何

濱田

このセクションは PDF の後ろのほうに結合されています. そちらを参照して下さい.

1.1　序

1.2　KNOPPIX について

1.3　KSEG について

1.4　KSEG 例題集

1.4.1　サイクロイド

例 1 サイクロイドを kseg で作る作り方.

1.4.2　サイクロイドの数学

1.4.3　アポロニウスの円

例 2 アポロニウスの円を kseg で作る作り方.

1.4.4　アポロニウスの円の数学

関連図書

[1] ここは参考文献.

第2章　Risa/Asir で書く GCD 計算とその応用

高山, 野呂

Risa/Asir は汎用の数式処理システムである．さらにライブラリとしてリンクしたり TCP/IP と OpenXM プロトコルを用いてサーバとして利用したりもできる．また C 言語に良く似た言語でプログラムを書くことも可能である．この章では Risa/Asir を簡単に紹介し, 応用例として実用的にも使うことの可能な GCD の計算プログラムを Risa/Asir で書いてみよう．

Risa/Asir についてより詳しい解説は [4] を見よ．

2.1　Knoppix での Risa/Asir の使い方

Risa/Asir を起動するには KDE のメニューから アプリケーション ⇒ Math ⇒ Asir(OpenXM) で起動する．Asir の一部分は利用許諾の問題から CD に含めていないので利用許諾に同意したあとネットワークからダウンロードする．サイズは 2M 以内なのでブロードバンド環境ならば 1 分以内にダウンロードが終了する．ダウンロードしたものは ホームディレクトリの下に名前.asir-tmp, .TeXmacs, .asirrc でセーブされる．たとえば knoppix ⇒ configure ⇒ 継続的なホームの作成 で USB メモリ等に書き込んでおけば再度ダウンロードする必要はない．

[数字]

は asir のプロンプト (入力催促文字列) である．終了は Risa/Asir のプロンプトに対して `quit;` を入力する．まず Asir を対話型電卓として利用する方法を説明する．

Asir は数の処理のみならず, 多項式の計算もできる．電卓的に使うための要点を説明し例をあげよう．

例 3 `+,-,*,/,^` はそれぞれ足し算, 引き算, かけ算, 割算, 巾乗. たとえば,

```
2*(3+5^4);
```

と入力すると $2(3+5^4)$ の値を計算して戻す．

例 4 `sin(x), cos(x)` はおなじみの三角関数. `@pi` は円周率を表す定数. $\sin(x)$ や $\cos(x)$ の近似値を求めるには

```
deval(sin(3.14));
```

と入力する. `deval` は 64 bit の浮動小数点数により計算する．

例 5

$$\left\{\left(2+\frac{2}{3}\right)4+\frac{1}{3}\right\}+5$$

を Asir で計算してみよう. ; (セミコロン) までを入力してから↵を押すと実行する. セミコロン ↵ の入力がないと実行がはじまらない. セミコロン の代わりに $ (ドル記号) を用いると, 計算した値を印刷しない. Risa/Asir では普通のプログラム言語と同様, 数式の括弧は (,) しか用いない. [,] や {,} は別の意味であり, 特に前者はリストを表す.

```
[0]     ((2+2/3)*4+1/3)+5;  ↵
16
[1]     ((2+2/3)*4+1/3)+5$  ↵
[2]
```

例 6 plot(f); は x の関数 f のグラフを描く x の範囲を指定したいときはたとえば plot(f,[x,0,10]) と入力すると, x は 0 から 10 まで変化する.

```
0
[1]     plot(sin(x));  ↵
  0
[2]     plot(sin(2*x)+0.5*sin(3*x),[x,-10,10]);  ↵
```

例 7 実行中の計算を中断したい時は ctrl+C (C は cancel の C) を入力する. すると

```
interrupt ?(q/t/c/d/u/w/?)
```

と表示されるので u ↵を入力する. 次に

```
Abort this computation? (y or n)
```

と表示されるので y ↵を入力する. (q/t/c/d/u/w/) の各文字の説明は ? を入力すれば読むことができる.

```
[0]     fctr(x^1000-y^1000);  ↵
ctrl+C
        interrupt ?(q/t/c/d/u/w/?)     u ↵
        Abort this computation? (y or n)     y ↵
return to toplevel
[1]
```

2.2 Risa/Asir のプログラミング

Asir は多項式の計算ができる. たとえば

```
fctr(x^10-1);
```

と入力すると x^10-1 の因数分解を行う. 他のプログラム言語と異なり, 小文字ではじまる記号は多

項式の変数である. たとえば x2 と書くと, $x2$ という名前の多項式の変数となる. x かける 2 は x*2 と書く. 大文字で始まる名前が変数となる. 下の例では K が変数である.

Risa/Asir で短いプログラムを書いてみよう. くりかえしは以下のように for 文を用いる.

例 8

```
for (K=1; K<=5; K=K+1) { print(K); };
```

を実行してみなさい. このプログラムは print(K) を K の値を 1 から 5 まで変えながら 5 回実行する.

```
[347]  for (K=1; K<=5; K=K+1) { print(K); };
1
2
3
4
5
[348] 0
[349]
```

例 9 関数 (手続き) は def 文で定義する. 関数の中の変数は自動的に局所変数となり, 外からは参照できない. 1 から N までの数の和を求める関数 main を定義して, 1 から 100 までの数の和を求めてみよう.

```
def main(N) {
  S = 0;
  for (K=1; K<=N; K++) {
      S = S+K;
  }
  return S;
}
main(100);
```

これらの内容を書いてあるファイル a.rr を emacs や ktext などのテキストエディタ作成して, load("./a.rr"); でロード実行すると修正が容易である.

2.3 情報科学からの準備

2.3.1 リスト

数学に集合やベクトルという考え方があるが, リスト構造という情報科学特有の考えはこれらに似ている. リストはいくつかのデータをまとめておくのに有効な仕組みである. Risa/Asir ではリストは [ ] で囲ってあらわす. 前節の関数 asciitostr の戻す値は実はリストであった. リストは次のような特徴がある.

- 先頭に要素を追加できる.

- 先頭の要素を外せる.
- 要素の書き換えはできない.
- 空リストがある.

一見して不自由そうに見えるが, 実はリストは強力で, リストだけでなんでもプログラミングできる. 例えば emacs は LISP と呼ばれるリスト処理言語で記述されていて emacs での 1 文字入力も実はある LISP コマンドに対応している.

1. リストの作り方 (その 1)

```
[0] A = [1,2,3];                    見ての通り
[1,2,3]                             表示が配列と微妙に違う
```

2. リストの作り方 (その 2)

```
[1] B = cons(0,A);                  先頭に要素を追加
[0,1,2,3]
[2] A;                              A は影響を受けない
[1,2,3]
```

3. リストの作り方 (その 3)

```
[3] C = cdr(A);                     cdr = クッダー; 先頭要素を取り外す
[2,3]
[4] A;                              A は影響を受けない
[1,2,3]
```

4. 空リスト

```
[5] A = [];                         [] は空のリストを表す
[]
[6] cons(1,A);
[1]
```

5. 要素取り出し (その 1)

```
[7] car(B);                         car = カー; 先頭要素を取り出す
0
```

6. 要素取り出し (その 2)

```
[8] B[2];                           配列と同様に書ける
2
```

7. 書き換え不可

```
[9] B[2] = 5;                       書き換えはダメ
putarray : invalid assignment
return to toplevel
```

8. リストの結合

```
[10] A = [1,2];
[11] B = [3,4];
[12] C = append(A,B);
```

A, B が結合されて, C には [1,2,3,4] がはいる.
cons の方がメモリの利用効率がよいがわかりやすくするため, 後述の RSA のプログラムでは append を多用している.

例 10 リストを用いると例えば, A を B で割った商と剰余を返す関数を次のように書ける.

プログラム

```
def quo_rem(A,B) {
    Q = idiv(A,B);
    R = A - Q*B;
    return [Q,R];
}
```

実行例

```
[1] QR = quo_rem(123,45);
[2,33]
[2] Q = QR[0];
2
[3] R = QR[1];
33
```

2.4 ユークリッドのアルゴリズム

整数環 $\mathbf{Z}$, 一変数多項式環 $\mathbf{k}[x]$ はともに次の割算定理が成り立つ: R を 整数環または一変数多項式環とする. このとき R の 0 でない任意の元 f, g に対して,

$$f = qg + r, \quad \deg(r) < \deg(g)$$

を満たす R の元 q, r が存在する. ここで $R = \mathbf{Z}$ のとき $\deg(f) = |f|$, $R = \mathbf{k}[x]$ のときは $\deg(f) = f$ の次数 と定義する

ユークリッド整域はこの割算定理の成立を仮定した整域であり, ユークリッド整域で議論を展開しておくことにより, 整数での議論も一変数多項式での議論も共通化が可能である. 計算機科学における, Object 指向, 部品化, 抽象データ型等の概念も, このような現代数学の考え方— 抽象化, 公理化— と同じである. 現代数学では, このような思考の節約は多くの分野で有効であったが, それが数学の全てではない. 同じように計算機科学における Object 指向や抽象データ型の概念 (Java などで実現されている) は, 有効な局面も多くあったが, 万能というわけではないことを注意しておこう.

2.5 単項イデアルと 1 変数連立代数方程式系の解法

“ユークリッド整域” では, 整数のときの互除法アルゴリズムがつかえる. 互除法アルゴリズムを用いることにより, 一変数多項式環のイデアルに関する多くの問題を解くことが可能である.

$f, g \in \mathbf{Q}[x]$ に対して, 一変数の連立代数方程式

$$f(x) = g(x) = 0$$

の共通根をもとめることを考えよう. (複素) 共通根の集合を

$$V(f, g) = \{a \in \mathbf{C} \mid f(a) = g(a) = 0\}$$

と書くことにする. 私達の考えたい問題は, $V(f,g)$ が空かそれとも何個の元からなっているか? 空でないとして, 根の近似値を求めることである.

この問題を見通しよく考えるには, イデアルの考えをもちいるとよい. I を f, g の生成するイデアルとしよう. つまり, $\mathbf{Q}[x]$ の部分集合

$$I = \langle f, g \rangle = \{p(x)f(x) + q(x)g(x) \,|\, p, q \in \mathbf{Q}[x]\}$$

を考える. このとき,

$$V(f,g) = V(I) = \{a \in \mathbf{C} \,|\, h(a) = 0 \text{ for all } \ h \in I\}$$

である.

さて, I は単項生成なので,

$$I = \langle h \rangle$$

となる生成元 h が存在する. $V(I) = V(h)$ であることが容易に分かるので, h が定数なら, $V(I)$ は空集合であり, そうでないときは, 重複度も込みで, $V(I)$ の個数は, h の次数にほかならない. h は f, g の GCD にほかならないことが証明できるので, 結局, 互除法アルゴリズムで h を f, g より計算して, それから, $h = 0$ を数値的にとけば, $V(f,g)$ を決定できることになる.

以上をプログラムすると以下のようになる. 関数 `g_c_d(F,G)` は多項式 `F` と `G` の最大公約多項式 (GCD) を求める. 関数 `division(F,G)` は割算定理をみたす, q, r を求めている. `variety(F,G)` で共通根の計算をおこなう.

次の関数達をすべて含めたファイルが `gcd.rr` である.

```
def in(F) {
  D = deg(F,x);
  C = coef(F,D,x);
  return(C*x^D);
}
```

```
def division(F,G) {
   Q = 0; R = F;
   while ((R != 0) && (deg(R,x) >= deg(G,x))) {
      D = red(in(R)/in(G));
      Q = Q+D;
      R = R-D*G;
   }
   return([Q,R]);
}
```

```
def g_c_d(F,G) {
   if (deg(F,x) > deg(G,x)) {
     S = F; T = G;
   }else {
     S = G; T = F;
   }
   while (T != 0) {
     R = division(S,T)[1];
     S = T;
     T = R;
   }
   return(S);
}
```

```
def variety1(F,G) {
   R = g_c_d(F,G);
   if (deg(R,x) == 0) {
     print("No solution.(variety is empty.)");
     return([]);
   }else{
     Ans = pari(roots,R);
     print("The number of solutions is ",0); print(size(Ans)[0]);
     print("The variety consists of  : ",0); print(Ans);
     return(Ans);
   }
}
end$
```

上のプログラムで利用されている組み込み関数について解説を加えておこう.

1. `deg(F,x)`： 多項式 `F` の変数 `x` についての次数をもどす. たとえば, `deg(x^2+x*y+1,x)` は 2 を戻す.

2. `coef(F,D,x)` : 多項式 `F` の変数 `x` の `D` 次の係数を戻す. すなわち, 多項式 `F` を変数 `x` の 1 変数多項式とみたとき `x^D` の係数を戻す. たとえば `coef(x^2+x*y+2*x+1,1,x)` は `y+2` を戻す.

よりくわしくは, `help` コマンドでマニュアルを参照してほしい.
例. $x^4-1=0$ と $x^6-1=0$ の共通根の集合, $V(x^4-1, x^6-1)$ の計算をしてみよう.

```
[346] load("gcd.rr");
1
[352] variety1(x+1,x-1);
No solution.(variety is empty.)
[]
```

```
[353] variety1(x^4-1,x^6-1);
The number of solutions is 2
The variety consists of  : [ -1.0000000000000000000 1.0000000000000000000 ]
[ -1.0000000000000000000 1.0000000000000000000 ]
[354]
```

あとの節でみるように, ユークリッドの互除法は数学において基本的のみならず, RSA 暗号系の基礎としても利用されており, 現代社会の基盤技術としても重要である. 蛇足ながら, こんな八方美人な数学の話はそうめったにないのも注意しておこう.

問題 1 3 つの多項式の共通零点を求めるプログラムを書きなさい.

問題 2 多項式環における一次不定方程式

$$p(x)f(x) + q(x)g(x) = d(x)$$

の解を一つ求めるアルゴリズムを考え, そのプログラムを書きなさい. ここで, f, g, d が与えられた一変数多項式で, p, q が未知である.

問題 3 来週数学のテスト?! プログラミングなんかしてらんない! ちょっとまった. 数学の教科書をみながら, いろんなプログラミングを考えてみるのはどうでしょう. この節でみたように, たとえばユークリッド環とそのイデアルについてプログラミングをすれば, 対象の理解がぐんとすすみます. 教科書を読んでわからなかったこともわかるようになるかも.

補足: ここでは, いくつかの一変数多項式が与えられたとき, それらが生成するイデアルの生成元が互除法で求められることを見た. そこで求めた生成元は, イデアルの中で 0 を除く最低次数のものであり, ある多項式がそのイデアルに属するかどうかは, 求めた生成元による割算の結果で判定できる. 多変数の場合, 一般にイデアルは単項生成にはならないが, 単項式の中にある種の全順序を入れることで, 剰余が一意的に計算できるような生成系 (グレブナ基底) を考えることができる. グレブナ基底を求めるアルゴリズムとして Buchberger アルゴリズムがあるが, それは互除法の拡張と思ってよい. グレブナ基底は多変数多項式の共通零点を求めるだけでなく, 理論的にも重要な役割を演じる. 詳しくは, **??** 章および [1] または [2] を参照.

Risa/Asir でグレブナ基底を計算するコマンドは, `gr` か `hgr` である. グレブナ基底の計算は, 互除法の拡張であるので, `gr` を用いても GCD を計算できる. `x` の多項式 `F` と `G` の GCD は, 集合 {`F`,`G`} のグレブナ基底であるので, コマンド `gr([F,G],[x],0);` でも計算できる.

2.6 計算効率

前節の関数 `g_c_d` で, $f = (2x^3 + 4x^2 + 3)(3x^3 + 4x^2 + 5)^{10}$, $g = (2x^3 + 4x^2 + 3)(4x^3 + 5x^2 + 6)^{10}$ の GCD を計算してみよう.

```
[151] F=(2*x^3+4*x^2+3)*(3*x^3+4*x^2+5)^10$
[152] G=(2*x^3+4*x^2+3)*(4*x^3+5*x^2+6)^30$
[153] H=g_c_d(F,G)$
6.511sec + gc : 0.06728sec(6.647sec)
```

使用する計算機にもよるが, 数秒程度で巨大な係数を持つ多項式が得られる. 実はこの多項式は $2x^3 + 4x^2 + 3$ の定数倍である. これを組み込み関数 `ptozp(F)` で確かめてみよう. `ptozp(F)` は, `F` に適当な有理数をかけて, 係数を GCD が 1 であるような整数にした多項式を返す関数である.

```
[154] ptozp(H);
2*x^3+4*x^2+3
```

この例からわかるように, 前節の g_c_d では, 互除法の途中および結果の多項式に分母分子が巨大な分数が現れてしまう. 人間と同様, 計算機も分数の計算は苦手である. そこで, 分数の計算が現れないように工夫してみよう. まず, 剰余を定数倍しても, GCD は定数倍の影響を受けるだけということに注意して, 次のような関数を考える.

```
def remainder(F,G) {
   Q = 0; R = F;
   HCG = coef(G,deg(G,x));
   while ((R != 0) && (deg(R,x) >= deg(G,x)))
      R = HCG*R-coef(R,deg(R,x))*x^(deg(R,x)-deg(G,x))*G;
   return R;
}
```

この関数は, 適当な自然数 k に対し

$$\mathrm{lc}(g)^k f = qg + r, \quad \deg(r) < \deg(g)$$

($\mathrm{lc}(g)$ は g の最高次の係数) なる $r \in \mathbf{Z}[x]$ を求めていることになる. この関数で, 前節の division を置き換えてみよう.

```
def g_c_d_1(F,G) {
   if (deg(F,x) > deg(G,x)) {
     S = F; T = G;
   }else {
     S = G; T = F;
   }
   while (T != 0) {
     R = pseudo_remainder(S,T);
     S = T;
     T = R;
   }
   return(S);
}
```

```
[207] g_c_d_1(F,G);
Needed to allocate blacklisted block at 0x988d000
Needed to allocate blacklisted block at 0x9899000
```

どうしたことか, 妙なメッセージは出るものの結果は出そうもない. 実は, pseudo_remainder でかけた $\mathrm{lc}(g)^k$ のせいで, 途中の多項式の係数が大きくなりすぎているのである. そこで, pseudo_remainder の結果を ptozp で簡単化してみよう.

```
def g_c_d_2(F,G) {
   if (deg(F,x) > deg(G,x)) {
     S = F; T = G;
   }else {
     S = G; T = F;
   }
   while (T != 0) {
     R = pseudo_remainder(S,T);
     R = ptozp(R);
     S = T;
     T = R;
   }
   return(S);
}
```

```
[237] g_c_d_2(F,G);
2*x^3+4*x^2+3
0.057sec(0.06886sec)
```

今度はずいぶん速く計算できた. ptozp では, 実際に係数の整数 GCD を計算することで簡単化を行っているが, より詳しく調べると, GCD を計算しなくても, GCD のかなりの部分はあらかじめ知ることができる. この話題にはこれ以上立ち入らない. [3] Section 4.6.1 または [2] 5.4 節 を参照して欲しい.

ここで見たように, 互除法のような単純なアルゴリズムでも, 実現方法によってはずいぶん効率に差が出る場合がある. 特に, 分数が現れないようなアルゴリズムを考えることは重要である.

問題 4 g_c_d も ptozp を用いることで高速化できる. その改良版 g_c_d と g_c_d_2 をさまざまな例で比較してみて, 分数が現れる演算が効率低下を招くことを確認せよ.

2.7 Risa/Asir の Windows 版について

まだ書いてない

関連図書

[1] D.Cox, J.Little, D.O'Shea, *Ideals, Varieties, and Algorithms — An Introduction to Commutative Algebraic Geometry and Commutative Algebra*, 1991, Springer-Verlag.
日本語訳: D. コックス, J. リトル, D. オシー: グレブナ基底と代数多様体入門 (上/下). 落合他訳, シュプリンガー フェアラーク 東京, 2000. ISBN 4-431-70823-5, 4-431-70824-3.
世界的に広く読まれているグレブナ基底の入門書. Buchberger アルゴリズム自体は, 2 章までよめば理解できる. Risa/Asir ドリルの **??** 章 (本章) および **??** 章 (次の章) はコックス達の本をもとにした, グレブナ基底の入門講義等の補足プリントがもとになっている. したがってコックス達の本とともに本章と次の章を読むと理解が深まるであろう. 本章で証明や説明を省略した数学的事実や概念については, コックス達の本の 1 章を参照されたい. 大学理系の教養課程の数学の知識で十分理解可能である.

[2] 野呂: 計算代数入門, Rokko Lectures in Mathematics, 9, 2000. ISBN 4-907719-09-4.
`http://www.math.kobe-u.ac.jp/Asir/ca.pdf` から, PDF ファイルを取得できる.
`http://www.openxm.org` より openxm のソースコードをダウンロードすると, ディレクトリ `OpenXM/doc/compalg` にこの本の TeX ソースがある.

[3] D.E. Knuth: *The Art of Computer Programming*, Vol2. Seminumerical Algorithms, 3rd ed. Addison-Wesley (1998). ISBN 0-201-89684-2.
日本語訳: "準数値算法", サイエンス社.

[4] Risa/Asir ドリル (これは Free Book です)

第3章　Knoppixで解く常微分方程式(Basic, Asir, Yorik)

高山

この章では 10 進 basic (Knoppix/Edu), Risa/Asir, yorick 等を用いて, 常微分方程式の数値解析を試みてみよう. Knoppix/math には多数の数学ソフトウエアが収録されているが, 一行の命令でいろいろなことをやらせることが可能ではあるが, さらにはプログラミングをすることによりよりいろいろなことが可能となる.

常微分方程式の数値解析はプログラミングの入門としても適切な題材であろう.

この章の最後では Knoppix/math に収録されている, `yorik` やフラクタルを見るためのソフトである `xaos` の紹介もする.

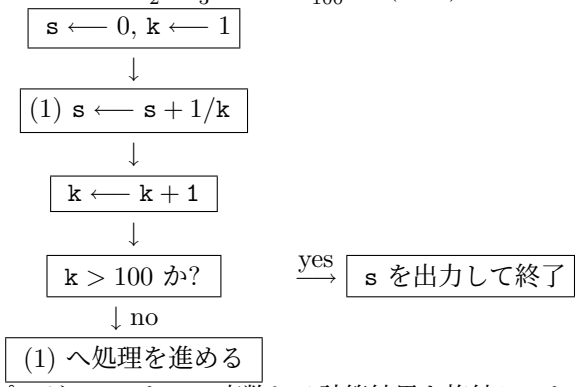
3.1　プログラミング入門

さて常微分方程式を差分法で解く前のトレーニングとして, 数列を求める basic のプログラムを書いてみよう. 高校の数学では数列の一般項を求める訓練を延々をうける. 計算機言語は数列の一般項を求める魔法の言語ではない. 計算機言語でできることはたとえば漸化式で定義された数列の 100 項目を求めるといった計算である. この節では, 数学はよく知ってるが計算機言語をはじめて使うという読者も対象にプログラムを書くイロハからはじめてみる. プログラムの入門者にとり, 10 進 basic (Knoppix/Edu) や Risa/Asir はとてもとっつきやすいプログラム言語であろう.

3.1.1　流れ図 — flow chart

アルゴリズムを図示して説明する方法がいろいろあるが, その最も古いものが “流れ図” (flow chart) である.

次の図は $1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{100}$ の (近似) 値を求める手順を流れ図で書いたのものである.



プログラムにおいて変数とは計算結果を格納しておく箱のようなものである. 格納できる数字の桁数はきまっている. 計算機にはメモリという名前の, コンデンサーとトランジスタを大量に用いた記

憶領域があるが, この変数 (箱のようなもの) はメモリを用いて実現されている.

さて, $\boxed{\mathtt{s} \leftarrow \mathtt{s} + 1/\mathtt{k}}$ の矢印は右辺の計算結果を変数 s に代入せよ, という意味である. はじめて (1) が実行される前は変数 s には 0 がはいっており, k には 1 がはいっている. (1) の右辺の計算が実行されて, その結果の $0 + 1 = 1$ が変数 s に代入されて (1) の実行が終る. 次の $\boxed{\mathtt{k} \longleftarrow \mathtt{k} + 1}$ が実行される前は変数 k には 1 が入っており, 右辺の計算が実行されて, その結果の $0 + 1 = 1$ が変数 k に代入されてこの処理が終る. k は 101 より小さいので (1) に実行が移る.

次に (1) が実行される前は変数 s には 1 がはいっており, k には 2 がはいっている. (1) の右辺の計算が実行されて, その結果の $1 + 1/2 = 1.5$ が変数 s に代入されて (1) の実行が終る. 次の $\boxed{\mathtt{k} \longleftarrow \mathtt{k} + 1}$ が実行される前は変数 k には 2 が入っており, 右辺の計算が実行されて, その結果の $2 + 1 = 3$ が変数 k に代入されてこの処理が終る. k は 101 より小さいので (1) に実行が移る.

次に (1) が実行される前は変数 s には 1.5 がはいっており, k には 3 がはいっている. (1) の右辺の計算が実行されて, $1.5 + 1/3$ の近似値 $1.8333\cdots3$ が変数 s に代入されて (1) の実行が終る. 近似値として小数点以下何桁までとるかは処理系によって決まっている. (このあたりは電卓による計算と同じである.) 次の $\boxed{\mathtt{k} \longleftarrow \mathtt{k} + 1}$ が実行される前は変数 k には 3 が入っており, 右辺の計算が実行されて, その結果の $3 + 1 = 4$ が変数 k に代入されてこの処理が終る. k は 101 より小さいので (1) に実行が移る.

以下この計算が k が 100 になるまで繰り返される.

3.1.2 $\mathtt{s} = \mathtt{s} + 1/\mathtt{k}$ の意味は?

さて前の節の処理を Basic のプログラムとして書くと次のようになる.

```
10 s=0
20 k=1
30 s = s+1/k
40 k=k+1
50 if k < 101 then goto 30
60 print s
70 end
```

Basic では数学と異なり = 記号を右辺計算して左辺に代入するという意味で用いており, 上のプログラムで `s = s+1/k` は $\boxed{\mathtt{s} \leftarrow \mathtt{s} + 1/\mathtt{k}}$ なる意味である.

また, `k=k+1` は $\boxed{\mathtt{k} \leftarrow \mathtt{k} + 1}$ なる意味である.

左辺には変数名しか書けないことに注意しておこう. したがって, たとえば, `k^2+2*k+1 = (k+1)^2` のような式を basic のプログラムとして書くと, エラーにある.

3.2 for ループ

20, 40, 50 の処理は定石的な繰り返し処理である. 定石的な繰り返し処理には専用の命令が用意されている. 今の場合 for ループを用いて次のように書くのが一般的である.

```
10 s=0
20 for k=1 to 100
30   s = s+1/k
40 next k
50 print s
60 end
```

3.2.1 練習

1. $1^2 + 2^2 + \cdots + 10^2$ を計算するアルゴリズムを流れ図で書き, Basic でプログラムしなさい.
2. 漸化式 $f_{k+1} = f_k + 1/f_k$, $f_1 = 1$ で決まる数列で f_{100} を求めるアルゴリズムを流れ図で書き, Basic でプログラムしなさい.
3. 乗法を繰り返して 2^16 を計算するアルゴリズムを流れ図で書き, Basic でプログラムしなさい.
4. 1/10! の (近似) 値を計算するアルゴリズムを流れ図で書き, Basic でプログラムしなさい.
5. $\sum_{k=0}^{1} 01/k!$ ($0! = 1$ ときめる) の (近似) 値を計算するアルゴリズムを流れ図で書き, Basic でプログラムしなさい.
6. ユークリッドの互除法を用いて整数 a と b の最大公約数 gcd(a, b) を計算するアルゴリズムを流れ図で書き, Basic でプログラムしなさい.

3.3 Debug の仕方

3.3.1 print 文をはさむ

3.3.2 自分が計算機となったつもりになって下のメモリに格納されたデータを書き変えていく

漸化式

$$f_{k+2} = f_{k+1} + f_k, \quad f_1 = f_2 = 1$$

できまる数列 (フィボナッチ数列) の 100 項まで計算して表示するプログラム.

```
10 a=1
20 b=1
30 for k=1 to 100
40    print a
50    c = a+b
60    a = b
70    b = c
80 next k
```

アドレス	変数の名前	内容	内容	内容	内容	内容
	k	1	2			
	a	1	1			
	b	1	2			
	c	2	3			

1. “内容” は 50 行目の実行が終了した時の各変数の値.

問題: 上の図の空白をうめよ.
問題: 他のプログラムについても同様な図を作成してプログラムの実行を追いなさい.
参考:

1. アドレス (メモリの番地) は Basic のプログラムを読むときは必要ないが, C のプログラムを読むときは必須. 特にポインターを利用するとき.
2. C のプログラムを読むときは, 変数のサイズ, つまり 2 進数何桁の数を格納できるかにも注意.

3.4　常微分方程式の差分化

微分方程式の近似解を求める漸化式の導出方法（差分法) について説明しよう.

3.4.1　単独方程式の差分化

今 $f(t)$ を未知関数とする微分方程式

$$f'(t) = a(t)f(t) + b(t), \quad f(0) = c$$

を解く問題を例として考える. ここで $a(t)$, $b(t)$ は時刻 t の関数であり, c は定数である. たとえば $a(t) = -1$, $b(t) = |\cos(t)|$, $c = 2$ とすれば上の微分方程式は $f'(t) = -f(t) + |\cos(t)|$, $f(0) = 2$ となる.

$f(t)$ は時刻 t おけるある量, たとえば温度とか細菌の数とか物体の位置や電流の大きさを表すとすれば, 初期条件 $f(0) = c$ での微分方程式を満たす関数 $f(t)$ を求めることは, 時刻 0 でのこれらの値から時刻 t での値を予想することにほかならない.

さて $h = \Delta t$ を微小な時間, たとえば $h = 0.00001$ とする. このとき微分の定義より $f'(t)$ は $(f(t+h) - f(t))/h$ にほぼ等しい. よって

$$f(t+h) \approx f(t) + hf'(t) = f(t) + (\Delta t)f'(t)$$

である. 微分方程式の右辺を用いて $f'(t)$ を書き換えると,

$$f(t+h) \approx f(t) + h(a(t)f(t) + b(t)) \tag{3.1}$$

となる. つまり $f(t)$ の $h = \Delta t$ 秒後の値 $f(t+h)$ は $f(t) + h(a(t)f(t) + b(t))$ に大体等しいということになる. つまり (3.1) は 微小時間 h 秒未来 の f の値を現在の f の値で表す式とみなせる.

(3.1) より $t = 0$ とすれば,

$$f(h) \approx f(0) + h(a(0)f(0) + b(0)) = c + h(a(0)c + b(0))$$

である. これで時刻 $t = h$ での $f(t)$ の近似値が求まった. 次に (3.1) で $t = h$ とすれば,

$$f(h+h) \approx f(h) + h(a(h)f(h) + b(h)),$$

$t = 2h$ とすれば

$$f(2h+h) \approx f(2h) + h(a(2h)f(2h) + b(2h))$$

となり, 時刻 $t = 2h$, $t = 3h$ の $f(t)$ の近似値が次々とさだまっていくこととなる. まとめると漸化式

$$f_{k+1} = f_k + h(a(hk)f_k + b(hk)), \quad f_0 = c \tag{3.2}$$

で数列 f_k を決めていけばそれが時刻 $t = hk$ での $f(t)$ の近似値となるのである.

3.4.2 連立方程式の差分化

問題 5 上の考え方を用いて $y_1(t)$, $y_2(t)$ を未知関数とする次の連立の微分方程式の近似解を求めるプログラムを書きなさい. ((f_k, g_k) の値をプロットしていくプログラムを書きなさい.)

$$y_1' = (2 - y_2)y_1, \quad y_2' = (2y_1 - 3)y_2, \quad y_1(0) = 4, y_2(0) = 1.$$

参考: 種族 2 は種族 1 を食べる. $y_1(t)$ は時刻 t における種族 1 の数. $y_2(t)$ は時刻 t における種族 2 の数. と解釈する場合もある.

答え. h を微小な数とする. $y_1(t) = f(t)$, $y_2(t) = g(t)$ とおこう. このとき微分方程式より

$$\begin{aligned} f(t+h) - f(t) &\approx h(2 - g(t))f(t) \\ g(t+h) - g(t) &\approx h(2f(t) - 3)g(t) \end{aligned}$$

である. よって,

$$\begin{aligned} f(t+h) &\approx f(t) + h(2 - g(t))f(t) \\ g(t+h) &\approx g(t) + h(2f(t) - 3)g(t) \end{aligned}$$

この式の右辺は時刻 t での f, g の値できまる量であり, 左辺は時刻 $t + h$ での f, g の値である. したがって漸化式

$$\begin{aligned} f_{k+1} &= f_k + h(2 - g_k)f_k \\ g_{k+1} &= g_k + h(2f_k - 3)g_k \\ f_0 &= 4, \\ g_0 &= 1 \end{aligned}$$

で数列 f_k, g_k をきめていけば f_k, g_k は 時刻 hk での $f(t)$, $g(t)$ の近似値となる.

```
100 SET WINDOW -1,10,-1,10
110 DRAW axes
120 LET  f=4
130 LET  g=1
140 LET  h=0.0001
150 FOR k=0 TO 200000
160    LET  f2 = f+h*(2-g)*f
170    LET  g2 = g+h*(2*f-3)*g
180    SET COLOR 0
190    PLOT POINTS: f,g
200    SET COLOR 4
210    PLOT POINTS: f2,g2
220    LET  f = f2
230    LET  g = g2
240 NEXT k
250 END
```

http://www.math.kobe-u.ac.jp/ ~ taka/2005/prayp.bas よりダウンロードできる.

問題 6 初期条件 $f_0 = 4, g_0 = 1$ を変更して種族 1, 2 が絶滅したり絶滅寸前になる初期条件を実験的に決めよ.

3.5　カオス的力学系

解の関数が非常に複雑な形を描くような微分方程式があるということは 19 世紀から 20 世紀前半の数学者により理論的にある程度解明されていたが, その理論は難解で科学者一般の認識となることはなかった.

グラフィックを容易に表示できる計算機と差分法による微分方程式の解法がこの状況を一変させることとなる. 1970 年代以降のことである. 解の関数が初期値に敏感に依存し, 非常に複雑な形を描くような微分方程式は カオス的な微分方程式 と呼ばれている. 現在はカオス的な方程式を含むいわゆる 複雑系 の考え方が提唱され多くの科学者の研究対象となっている.

ここではカオス的な微分方程式の代表の一つである ローレンツ方程式 の解のグラフを描くプログラムを実行してその複雑な解を見てみよう.

課題 1

(1) 次のプログラムを入力して実行し, 解のグラフを観察せよ.

(2) 180, 190, 200 行で初期条件を設定している. これらの値を変えて解のグラフがどのように変わるか調べよ. 解のグラフは 8 の字を描く. 8 の下の部分を何度か回ってから 8 の上の部分へ移り上の部分を何度かまわる. これを繰り返す. 回る回数を数えよ. 動きが早すぎるときは 235 行目の wait delay 命令での待ち時間をたとえば

```
wait  delay  0.1
```

とするとよい. 0.1 は 0.1 秒停止することを意味する.

```
100 ! lorentz.bas. Solving p1' = -a p1 + a p2,
110 ! p2' = -p1 p3 + b p1 - p2
120 ! p3' = p1 p2 + c p3
130 SET WINDOW -25,25,-25,25
140 DRAW axes
150 LET  a=10
160 LET  b=20
170 LET  c=2.66
180 LET  p1=0
190 LET  p2 = 3
200 LET  p3 = 0
210 LET  dt = 0.004
220 LET  t = 0
230 FOR t=0 TO 50 STEP dt
235    wait delay 0.01
240    LET  q1 = p1+dt*(-a*p1+a*p2)
250    LET  q2 = p2+dt*(-p1*p3+b*p1-p2)
260    LET  q3 = p3+dt*(p1*p2-c*p3)
262    SET COLOR 0
264    PLOT POINTS: p1,p2
266    SET COLOR 4
270    PLOT POINTS: q1,q2
280    LET  p1 = q1
290    LET  p2 = q2
300    LET  p3 = q3
310 NEXT t
320 END
```

ローレンツ方程式は 3 つの未知関数 $p_1(t), p_2(t), p_3(t)$ についての次の連立微分方程式である. a, b, c には適当な数字をいれる.

$$\begin{aligned} p_1' &= -ap_1 + ap_2, \\ p_2' &= -p_1p_3 + bp_1 - p_2, \\ p_3' &= p_1p_2 + cp_3 \end{aligned}$$

上のプログラムでは $(p_1(t), p_2(t))$ をプロットしている.

`http://www.math.kobe-u.ac.jp/~taka/2004/lorentz2.txt` よりダウンロードできる.

差分化の解説 240 行目を変形すると

$$\frac{\mathtt{q1}-\mathtt{p1}}{\mathtt{dt}} = -\mathtt{a}*\mathtt{p1}+\mathtt{a}*\mathtt{p2}$$

である. dt を微小な数 (上のプログラムでは 0.004) としたとき, 変数 q1 に $p_1(t+\mathtt{dt})$, 変数 p1 に $p_1(t)$, 変数 q2 に $p_2(t+\mathtt{dt})$, 変数 p2 に $p_2(t)$, 変数 q3 に $p_3(t+\mathtt{dt})$, 変数 p3 に $p_3(t)$, が対応する. したがって 240 行目は

$$-\mathtt{a}p_1(t)+\mathtt{a}p_2(t) = \frac{p_1(t+\mathtt{dt})-p_1(t)}{\mathtt{dt}} \simeq p_1'(t)$$

(一つ目の微分方程式) にほかならない. 同様に 250, 260 行目はそれぞれ 2 つ目, 3 つ目の微分方程式にほかならない.

3.6 2 階の微分方程式の差分化

たとえば, 単振動の方程式

$$\frac{d^2}{dt^2}y + y = 0$$

を数値的に解くことを考えてみよう.

物理既修者向け解説:
解く前に, この方程式の物理的意味を復習しておこう. 高校物理の最初の基本公式は

$$\text{質量} \times \text{加速度} = \text{力}$$

なる関係式である. この関係式を Newton の運動方程式という. 物体の時刻 t における位置を $q(t)$ とおくと, 速度は $q'(t)$, 加速度は $q''(t)$ である.
1 次元的に単振動をするバネについた 質量 1 の物体 W を考えよう. 時刻 t における, W の位置を $y(t)$ とすることにする. ただし, バネが自然な長さにあるとき $y=0$ とする. フックの法則によると, 自然な長さから y だけのびた (負のときはちじんだとみなす) とき物体 W にかかる力は $-ky$ である. ここで k はバネできまる定数. ここで $k=1$ と仮定して Newton の運動方程式を適用すると, 単振動の方程式

$$y''+y=0$$

を得る.

$y(0)=1$, $y'(0)=0$ を初期条件として $y(t)=\cos(t)$ がこの方程式の解であるが, この方程式を数値解法で解こう. 数値解法の利点は, cos や sin で解を書けないときでも, 微分方程式の近似解がわかることである. 式 (??) より, h が十分小さいとき,

$$\frac{y(t+h)-2y(t)+y(t-h)}{h^2}+y(t)$$

は大体 0 に等しい. したがって,

$$y(t+h) = 2y(t)-y(t-h)-h^2y(t)$$

が近似的になりたつとしてよいであろう. この式は時刻 $t-h$ と t の y の値で, すこし先の時刻 $t+h$ の y の値を表す式である. また $y'(0)$ は $\frac{y(h)-y(0)}{h}$ にほぼ等しい. $y'(0)=0$ なので, $y(h)=y(0)$ が近似的に成り立つとしてよいであろう. したがって, 漸化式

$$y_{k+2} = 2y_{k+1} - y_k - h^2 h_{k+1},\ y_0 = 1, y_1 = 1 \tag{3.3}$$

を満たす数列を決めることにより, 解の近似をもとめることが可能であると予想できる. つまり y_k は hk 秒での y の値を近似していると予想される. このように解の近似を求める方法を 差分法 とよぶ. 漸化式 (3.3) を元の微分方程式の 差分化, または差分スキームとよぶ.

```
! 振動の方程式 y''+y = 0
LET  x1=1
LET  x2=1
LET  dt=0.01  ! これは十分小さい数なら何でもよい.
! y(k+2), y(k+1), y(k) が x3, x2, x1 に対応.
! h が dt に対応.
FOR t=0 TO 3 STEP dt
   LET x3 = 2*x2-x1-dt*dt*x2
   PRINT t,x1
   PLOT  LINES: t,x1;
   LET  x1=x2
   LET  x2=x3
NEXT t
END
```

上のプログラムで ! で始まる行は注釈行 (コメント行) であり, プログラムの実行には関係ない. ! から始まる注釈は行の途中から書きはじめてもよい. この場合は ! から行末までが注釈となる. 適宜注釈行を書くことにより, 人間がプログラムを読む助けとなる.

さて, これで微分方程式を近似的に解く問題が, 漸化式をみたす数列を求める問題になったのであるが, このような近似解が本当の解に収束するかとか, 全くことなる解しかとらえられない不安定現象が起こる場合があるとかの議論をやらないといけない. これはより上級の話題である.

注意: 漸化式は常に微小な数 h を含む. この h は十分小さければどんな数でも構わない. たとえば 0.01 とか 0.001 など. 近似解が本当の解に収束するという議論では, h を小さくしていけば差分法で求めた近似解が本当の解に収束して行く (どんどん近くなる) ということを証明する.

問題 7

h が十分小さいとき次の近似公式が成り立つ.

$$\begin{aligned} \frac{dy}{dt}(T) &\simeq \frac{y(T+h)-y(T)}{h} \\ \frac{d^2y}{dt^2}(T) &\simeq \frac{y(T+h)-2y(T)+y(T-h)}{h^2} \end{aligned}$$

この近似公式を用いて次の微分方程式を差分化しなさい. (微分方程式の解を近似する数列の漸化式を求めよ.) 数列の漸化式を用いて数列の値を決めて行くプログラムを書きなさい.

$$y'' + 3y = 0, \quad y(0) = 1, y'(0) = 0.$$

さてこのあたりで読者には徹底的に復習をすることをすすめる. (1) 微分方程式 $y''+y=0, y(0)=1, y'(0)=0$ の差分化を独力で導けるだろうか? (2) 上のプログラムを自力で書けるだろうか? (1), (2) ができたら次へ進むとよいであろう. 孔子が論語の中でいっているように, “学びて時にこれを習う. またよろこばしからずや” である.

3. 第 3 回の講義では, 物体の落下を表す微分方程式 $y'' = -9.8$ について学んだ. (a) 次のプログラムを入力, 実行せよ. どのようなグラフとなるか? (b) 物体が速度に比例する抵抗を受ける場合, 方程式は $y'' = -9.8 - ay'$ となる (a は適当な定数). これを解くプログラムに変更せよ.

```
100 ! 方程式 y''=-9.8, y(0)=100, y'(0)=0
110 SET WINDOW -1,5,-10,110
120 DRAW axes
130 LET  y1 = 100
140 LET  y2 = 100
150 LET  dt=0.001
160 ! y(k+1), y(k), y(k-1) が  y3, y2, y1 に対応. h が dt に対応.
170 FOR t=0 TO 5 STEP dt
180   LET  y3 = 2*y2 - y1 -9.8*dt*dt
190   PLOT LINES: t,y1
200   LET  y1 = y2
210   LET  y2 = y3
220 NEXT t
230 END
```

この問題も, 上と同じ方針で解ける. つまり,

$$y''(t) = -9.8 - ay'(t)$$

を とりあえず差分化してみればよい. 差分化の計算を記す.

$$y''(t) \simeq \frac{y(t+h) - 2y(t) + y(t-h)}{h^2},\ y'(t) \simeq \frac{y(t+h) - y(t)}{h},\ y''(t) = -9.8 - ay'(t)$$

より

$$\frac{y(t+h) - 2y(t) + y(t-h)}{h^2} \simeq -9.8 - a\frac{y(t+h) - y(t)}{h}$$

両辺に h^2 を掛けて,

$$y(t+h) - 2y(t) + y(t-h) \simeq -9.8h^2 - ah\,[y(t+h) - y(t)]$$

移項すると

$$(1+ah)y(t+h) \simeq (2+ah)y(t) - y(t-h) - 9.8h^2$$

$y(t+h)$ を y3, y(t) をy2, $y(t-h)$ を y1 と置き換え, $\simeq$ を $=$ とおけば,

$$(1 + a*h)*\mathtt{y3} = (2 + a*h)*\mathtt{y2} - \mathtt{y1} - 9.8*h^2$$

したがって 180 行を次のように書き換えればよい.

```
180 LET y3 = ((2+a*dt)*y2- y1 - 9.8*dt*dt)/(1+a*dt)
```

3.7 Risa/Asir で書くと?

```
load("glib");

def lorentz() {
  glib_window(-25,-25,25,25);
  A=10; B=20; C=2.66;
  P1=0; P2 = 3; P3 = 0;
  Dt = 0.004; T = 0;
  while (T <50) {
      Q1=P1+Dt*(-A*P1+A*P2);
      Q2=P2+Dt*(-P1*P3+B*P1-P2);
      Q3=P3+Dt*(P1*P2-C*P3);
      glib_putpixel(Q1,Q2);
      T=T+Dt;
      P1=Q1; P2=Q2; P3=Q3;
  }
}

end$
```

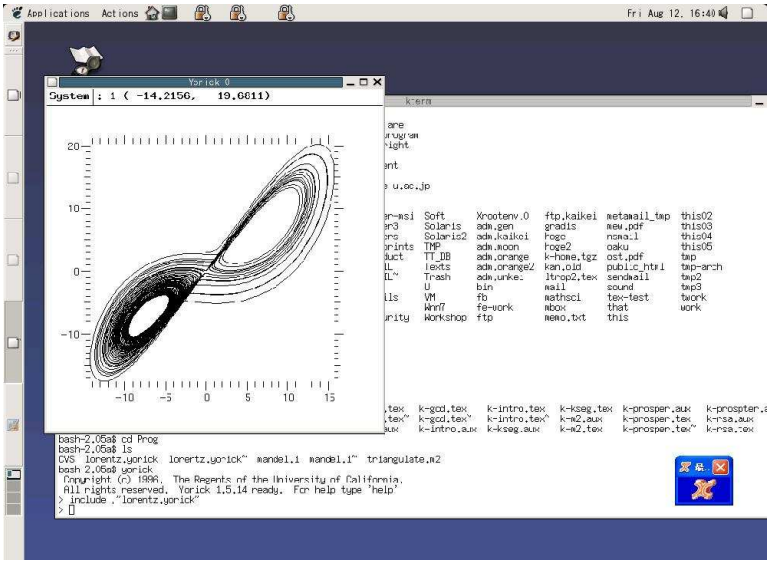
3.8 yorik によるシミュレーション

上と同じプログラムを シミュレーションシステム yorick で書くと次のようになる. Yorick のユーザ言語は Risa/Asir とよく似ているが, yorick では引数の無い関数には （ ） をつけない.

```
// yorick
// http://www.maumae.net/yorick/doc/refcard/index.php
// include,"lorentz.yorick"
func lorentz {
  A=10; B=20; C=2.66;
  P1=0; P2 = 3; P3 = 0;
  Dt = 0.004; T = 0;
  while (T <50) {
      Q1=P1+Dt*(-A*P1+A*P2);
      Q2=P2+Dt*(-P1*P3+B*P1-P2);
      Q3=P3+Dt*(P1*P2-C*P3);
      pldj,P1,P2,Q1,Q2;
      T=T+Dt;
      P1=Q1; P2=Q2; P3=Q3;
  }
}

lorentz;
```

このファイル名を lorentz.i とするとき， `include,"lorentz.i"` で実行できる．



3.9 フラクタルの作成

3.9.1 フラクタルって何?

3.9.2 Risa/Asir による Mandelbrot 集合の描画

```
/* mandelbrot.rr */
load("glib")$
#define MAG 50.0
#define abs(x) (x < 0.0? -x : x)
def mandel() {
  glib_open();

  for (Ar = 0.0; Ar < 5.0 ; Ar += 1.0/MAG) {
for (Ai = 0.0; Ai <5.0; Ai += 1.0/MAG) {
  X = 0.1 ; Y = 0.1;
  for (I=0; I<200; I++) {
D = X*X*X*X+2*Y*Y*X*X+Y*Y*Y*Y;
X2 = ((2.0/3.0)*X*X*X*X*X+(4.0/3.0)*Y*Y*X*X*X-(1.0/3.0)*Ar*X*X
              +((2.0/3.0)*Y*Y*Y*Y-(2.0/3.0)*Ai*Y)*X+(1.0/3.0)*Ar*Y*Y)/D;
Y2 = ((2.0/3.0)*Y*X*X*X*X+((4.0/3.0)*Y*Y*Y-(1.0/3.0)*Ai)*X*X
              +(2.0/3.0)*Ar*Y*X+(2.0/3.0)*Y*Y*Y*Y*Y+(1.0/3.0)*Ai*Y*Y)/D;
if (abs(X2) > 30.0 || abs(Y2) > 30.0) {
  break;
}
if (I == 199) {
  glib_putpixel(Ar*MAG,Ai*MAG);
}
X = X2; Y = Y2;
  }
}
  }
}

print("Type in mandel()")$
end$
```

3.9.3 Yorick による Mandelbrot 集合の描画

```
/*  include,"mandel.i" */
func mandel {
  MAG=50.0;

  for (Ar = 0.0; Ar < 5.0 ; Ar += 1.0/MAG) {
for (Ai = 0.0; Ai <5.0; Ai += 1.0/MAG) {
  X = 0.1 ; Y = 0.1;
  for (I=0; I<200; I++) {
D = X*X*X*X+2*Y*Y*X*X+Y*Y*Y*Y;
X2 = ((2.0/3.0)*X*X*X*X*X+(4.0/3.0)*Y*Y*X*X*X-(1.0/3.0)*Ar*X*X
              +((2.0/3.0)*Y*Y*Y*Y-(2.0/3.0)*Ai*Y)*X+(1.0/3.0)*Ar*Y*Y)/D;
Y2 = ((2.0/3.0)*Y*X*X*X*X+((4.0/3.0)*Y*Y*Y-(1.0/3.0)*Ai)*X*X
              +(2.0/3.0)*Ar*Y*X+(2.0/3.0)*Y*Y*Y*Y*Y+(1.0/3.0)*Ai*Y*Y)/D;
if (abs(X2) > 30.0 || abs(Y2) > 30.0) {
  break;
}
if (I == 199) {
  plm,[[Ar*MAG,Ar*MAG],[Ar*MAG+1,Ar*MAG+1]],
              [[Ai*MAG,Ai*MAG+1],[Ai*MAG,Ai*MAG+1]],color="red";
}
X = X2; Y = Y2;
  }
}
  }
}

mandel;
```

3.9.4 Xaos でみる Mandelbrot 集合

Knoppix/Math の Xaos を用いると前の節で描画した Mandelbrot 集合を自由自在に拡大してみることができる.

3.10 箱と玉の系とソリトン

```
DIM b(20,40)
FOR i=1 TO 40
   LET  b(1,i) = 0
NEXT i
LET  b(1,2) =1
LET  b(1,3) = 1
LET  b(1,4) = 1
LET  b(1,10) =1
LET  b(1,15)=1
SET WINDOW 1,20,1,40
DRAW grid
FOR n=1 TO 20-1
   LET  my = 0
   FOR j=1 TO 40
      IF b(n,j)=1 THEN SET AREA COLOR 1
      IF b(n,j)=0 THEN SET AREA COLOR 0
      PLOT AREA : n,j;n+1,j; n+1,j+1;n,j+1
      IF b(n,j)=1 THEN
         LET  my = my+1
         LET  b(n+1,j) = 0
      ELSEIF b(n,j)=0 AND my > 0 THEN
         LET  my = my-1
         LET  b(n+1,j) = 1
      END IF
   NEXT j
NEXT n
END
```

関連図書

[1] ここは参考文献.

第4章　Risa/Asir で書く RSA 暗号システム

高山, 野呂

Risa/Asir は汎用の数式処理システムである．さらにライブラリとしてリンクしたり TCP/IP と OpenXM プロトコルを用いてサーバとして利用したりもできる．また C 言語に良く似た言語でプログラムを書くことも可能である．この章では Risa/Asir を簡単に紹介し, 応用例として実用的にも使うことの可能な RSA 暗号システムを Risa/Asir で書いてみよう．

Risa/Asir についてより詳しい解説は [1] を見よ．

4.1　Knoppix での Risa/Asir の使い方

Risa/Asir を起動するには KDE のメニューから アプリケーション ⇒ Math ⇒ Asir(OpenXM) で起動する．Asir の一部分は利用許諾の問題から CD に含めていないので利用許諾に同意したあとネットワークからダウンロードする．サイズは 2M 以内なのでブロードバンド環境ならば 1 分以内にダウンロードが終了する．ダウンロードしたものは ホームディレクトリの下に名前.asir-tmp, .TeXmacs, .asirrc でセーブされる．たとえば knoppix ⇒ configure ⇒ 継続的なホームの作成 で USB メモリ等に書き込んでおけば再度ダウンロードする必要はない．

[数字]

は asir のプロンプト (入力催促文字列) である．終了は Risa/Asir のプロンプトに対して `quit;` を入力する．まず Asir を対話型電卓として利用する方法を説明する．

Asir は数の処理のみならず, 多項式の計算もできる．電卓的に使うための要点を説明し例をあげよう．

例 11 `+,-,*,/,^` はそれぞれ足し算, 引き算, かけ算, 割算, 巾乗．たとえば,

```
2*(3+5^4);
```

と入力すると $2(3+5^4)$ の値を計算して戻す．

例 12 `sin(x), cos(x)` はおなじみの三角関数．`@pi` は円周率を表す定数．$\sin(x)$ や $\cos(x)$ の近似値を求めるには

```
deval(sin(3.14));
```

と入力する．`deval` は 64 bit の浮動小数点数により計算する．

例 13

$$\left\{\left(2+\frac{2}{3}\right)4+\frac{1}{3}\right\}+5$$

を Asir で計算してみよう．`;` (セミコロン) までを入力してから↵を押すと実行する．セミコロン ↵ の入力がないと実行がはじまらない．セミコロン の代わりに `$` (ドル記号) を用いると, 計算した

値を印刷しない. Risa/Asir では普通のプログラム言語と同様, 数式の括弧は (,) しか用いない. [,] や {,} は別の意味であり, 特に前者はリストを表す.

```
[0]     ((2+2/3)*4+1/3)+5;  ↵
16
[1]     ((2+2/3)*4+1/3)+5$  ↵
[2]
```

例 14 plot(f); は x の関数 f のグラフを描く x の範囲を指定したいときはたとえば plot(f,[x,0,10]) と入力すると, x は 0 から 10 まで変化する.

```
0
[1]     plot(sin(x));  ↵
  0
[2]     plot(sin(2*x)+0.5*sin(3*x),[x,-10,10]);  ↵
```

例 15 実行中の計算を中断したい時は ctrl+C (C は cancel の C) を入力する. すると

```
interrupt ?(q/t/c/d/u/w/?)
```

と表示されるので u ↵を入力する. 次に

```
Abort this computation? (y or n)
```

と表示されるので y ↵を入力する. (q/t/c/d/u/w/) の各文字の説明は ? を入力すれば読むことができる.

```
[0]     fctr(x^1000-y^1000);  ↵
ctrl+C
        interrupt ?(q/t/c/d/u/w/?)     u ↵
        Abort this computation? (y or n)     y ↵
return to toplevel
[1]
```

4.2 Risa/Asir のプログラミング

Asir は多項式の計算ができる. たとえば

```
fctr(x^10-1);
```

と入力すると x^10-1 の因数分解を行う. 他のプログラム言語と異なり, 小文字ではじまる記号は多項式の変数である. たとえば x2 と書くと, $x2$ という名前の多項式の変数となる. x かける 2 は x*2 と書く. 大文字で始まる名前が変数となる. 下の例では K が変数である.

Risa/Asir で短いプログラムを書いてみよう. くりかえしは以下のように for 文を用いる.

例 16

```
for (K=1; K<=5; K=K+1) { print(K); };
```

を実行してみなさい. このプログラムは print(K) を K の値を 1 から 5 まで変えながら 5 回実行する.

```
[347]  for (K=1; K<=5; K=K+1) { print(K); };
1
2
3
4
5
[348] 0
[349]
```

例 17 関数 (手続き) は def 文で定義する. 関数の中の変数は自動的に局所変数となり, 外からは参照できない. 1 から N までの数の和を求める関数 main を定義して, 1 から 100 までの数の和を求めてみよう.

```
def main(N) {
  S = 0;
  for (K=1; K<=N; K++) {
      S = S+K;
  }
  return S;
}
main(100);
```

これらの内容を書いてあるファイル a.rr を emacs や ktext などのテキストエディタ作成して, load("./a.rr"); でロード実行すると修正が容易である.

4.3 情報科学からの準備

4.3.1 文字コード, アスキーコード

計算機の内部では文章, 音, 画像, プログラム等さまざまな情報が 2 進数に変換されて格納されている. 計算機の記憶装置は 2 進数 8 桁分を基本単位としている. 計算機の扱う情報は 2 進数である. それらにたいする, 四則演算と記憶ですべての処理がすすむ. これを常に念頭において勉強していくのが, 理解の早道であろう.

2 進数で 8 桁の情報を 1 byte とよぶ. メモリ等の記憶装置の各番地に 1 byte の情報を格納でき

る. 2 進数は 0 と 1 の 2 つの記号を用いて数を表現する. 2 進数と 10 進数の対応は以下のとおり.

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8

$a_n a_{n-1} \cdots a_0$, $a_i \in \{0, 1\}$ なる表示の 2 進数を 10 進数であらわすと,

$$\sum_{k=0}^{n} a_k 2^k$$

に等しい. たとえば, 2 進数 1101 は, $2^3 + 2^2 + 2^0$ に等しい.

2 進数を書くのは桁が多くて面倒なので, 代わりに普通 16 進数を使うことが多い. 16 進数 2 桁が 2 進数 8 桁に対応するので換算が簡単である. 16 進数では

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

と 16 個の記号を用いて数をあらわす. A が 10 進数の 10, B が 10 進数の 11, C が 10 進数の 12, D が 10 進数の 13, E が 10 進数の 14, F が 10 進数の 15 に対応する. したがって, 2 進数, 16 進数, 10 進数の対応は以下のようになる.

0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

10000	10	16
10001	11	17
10010	12	18
10011	13	19
10100	14	20
10101	15	21
10110	16	22
10111	17	23
11000	18	24
11001	19	25
11010	1A	26
11011	1B	27
11100	1C	28
11101	1D	29
11110	1E	30
11111	1F	31

16 進数は `0x` や H をつけて表すことも多い. また A から F を小文字で書くことも多い. たとえば `0x1E`, `0x1e`, 1EH, 1eH はすべて 16 進数の 1E を表す. $a_n a_{n-1} \cdots a_0$, $a_i \in \{0, 1, \ldots, F\}$ なる表示の 16 進数を 10 進数であらわすと,

$$\sum_{k=0}^{n} a_k 16^k$$

に等しい. ただし a_i が 0xA のときは 10, a_i が 0xB のときは 11, 等と解釈する. 2 進数と 16 進数の間の変換は容易であろう.

問題 8 次の 16 進数達を 10 進数, 2 進数で書け. 0xFF, 0x100, 0xFFFF, 0x10000, 0x81.

問題 9 16 進数で筆算をやることを試みよ.

さて文字をどのようにして 2 進数として表現するかは, 規格が決められている. アルファベットについてはアスキーコードが標準としてつかわれている. 次の表がその対応表である.

20		40	@	60	‘
21	!	41	A	61	a
22	"	42	B	62	b
23	#	43	C	63	c
24	$	44	D	64	d
25	%	45	E	65	e
26	&	46	F	66	f
27	’	47	G	67	g
28	(	48	H	68	h
29	)	49	I	69	i
2a	*	4a	J	6a	j
2b	+	4b	K	6b	k
2c	,	4c	L	6c	l
2d	-	4d	M	6d	m
2e	.	4e	N	6e	n
2f	/	4f	O	6f	o
30	0	50	P	70	p
31	1	51	Q	71	q
32	2	52	R	72	r
33	3	53	S	73	s
34	4	54	T	74	t
35	5	55	U	75	u
36	6	56	V	76	v
37	7	57	W	77	w
38	8	58	X	78	x
39	9	59	Y	79	y
3a	:	5a	Z	7a	z
3b	;	5b	[	7b	{
3c	<	5c	\	7c	|
3d	=	5d	]	7d	}
3e	>	5e	^	7e	~
3f	?	5f	_	7f	

20H が空白, 20H 未満は制御コードと呼ばれている. たとえば, unix では 0AH が改行を表すために使われている. 制御コードの意味は OS によってことなる. MSDOS や Windows では 0DH, 0AH の 2 バイトが改行を表すために, Macintosh では 0DH が改行を表すために利用されている.

問題 10 (05)　次の文章をアスキーコードで表すとどうなるか？

Do not cry over the spilt milk.

Risa/Asir では組み込み関数 `strtoascii` が用意されており, これを用いるとアスキーコードを, 対応する数字の列に変換できる. 逆の変換をする関数は `asciitostr` である.

```
[1271] strtoascii("ABC abc"); ↵
 [65,66,67,32,97,98,99]
```

上の対応表をみればわかるように文字列 `ABC abc` がアスキーコードの列に変換された. とくに空白文字が 32 であることに注意しよう. 逆変換は次のようにおこなう.

```
[1272] asciitostr([65,66,67,32,97,98,99]); ↵
 ABC abc
```

4.3.2　リスト

数学に集合やベクトルという考え方があるが, リスト構造という情報科学特有の考えはこれらに似ている. リストはいくつかのデータをまとめておくのに有効な仕組みである. Risa/Asir ではリストは [ ] で囲ってあらわす. 前節の関数 `asciitostr` の戻す値は実はリストであった. リストは次のような特徴がある.

- 先頭に要素を追加できる.
- 先頭の要素を外せる.
- 要素の書き換えはできない.
- 空リストがある.

一見して不自由そうに見えるが, 実はリストは強力で, リストだけでなんでもプログラミングできる. 例えば emacs は LISP と呼ばれるリスト処理言語で記述されていて emacs での 1 文字入力も実はある LISP コマンドに対応している.

1. リストの作り方 (その 1)

```
[0] A = [1,2,3];                    見ての通り
[1,2,3]                             表示が配列と微妙に違う
```

2. リストの作り方 (その 2)

```
[1] B = cons(0,A);                  先頭に要素を追加
[0,1,2,3]
[2] A;                              A は影響を受けない
[1,2,3]
```

3. リストの作り方 (その 3)

```
[3] C = cdr(A);
[2,3]
[4] A;
[1,2,3]
```

cdr = クッダー; 先頭要素を取り外す

A は影響を受けない

4. 空リスト

```
[5] A = [];
[]
[6] cons(1,A);
[1]
```

[] は空のリストを表す

5. 要素取り出し (その 1)

```
[7] car(B);
0
```

car = カー; 先頭要素を取り出す

6. 要素取り出し (その 2)

```
[8] B[2];
2
```

配列と同様に書ける

7. 書き換え不可

```
[9] B[2] = 5;
putarray : invalid assignment
return to toplevel
```

書き換えはダメ

8. リストの結合

```
[10] A = [1,2];
[11] B = [3,4];
[12] C = append(A,B);
```

A, B が結合されて, C には [1,2,3,4] がはいる.
cons の方がメモリの利用効率がよいがわかりやすくするため, 後述の RSA のプログラムでは append を多用している.

例 18 リストを用いると例えば, A を B で割った商と剰余を返す関数を次のように書ける.

プログラム

```
def quo_rem(A,B) {
    Q = idiv(A,B);
    R = A - Q*B;
    return [Q,R];
}
```

実行例

```
[1] QR = quo_rem(123,45);
[2,33]
[2] Q = QR[0];
2
[3] R = QR[1];
33
```

4.4　数学からの準備

RSA 暗号系は, 次の定理を基礎としている.

定理 1 G を位数 (要素の個数) が n の群とするとき G の任意の元 a に対して $a^n = e$ である. ここで e は単位元である.

群の定義については, 適当な数学の本を参照されたい.

この定理は可換とは限らない一般の群で成立するが, ここでは可換な場合の証明のみを紹介する. この証明の理解には群の定義を知ってるだけで十分である.

定理 1 の証明: 群 G の n 個の相異なる要素を $g_1, \ldots, g_n$ としよう. このとき, $\{ag_1, \ldots, ag_n\}$ を考えるとこれらもまた, G の n 個の相異なる元の集合となる. なぜなら, たとえば $ag_i = ag_j$ となると, a の逆元を両辺にかけることにより, $g_i = g_j$ になり, 仮定に反するからである.

$\{g_1, \ldots, g_n\}$ と $\{ag_1, \ldots, ag_n\}$ は集合として等しいのであるから,

$$g_1 \cdots g_n = (ag_1) \cdots (ag_n) = a^n (g_1 \cdots g_n)$$

がなりたつ. 両辺に $g_1 \cdots g_n$ の逆元を掛けてやると, $e = a^n$ をえる. 証明おわり.

p を素数としよう. とくにこの定理を, $\mathbf{Z}/p\mathbf{Z}$ の乗法群

$$G = \{1, 2, \ldots, p-1\}$$

に適用すると, 次の定理を得る.

定理 2 p を素数とするとき, p で割れない任意の整数 x について,

$$x^{p-1} = 1 \bmod p$$

となる.

もうすこしくわしくこの定理の説明をしよう. $a \bmod p$ で, a を p でわった余りをあらわすものとする. このとき

$$(a \bmod p)(b \bmod p) \bmod p = ab \bmod p$$

が成立する. 左辺は, a を p でわった余りと b を p でわった余りを掛けたあと, p でわった余りをとることと, ab を p でわった余りをとることは同じである. という意味である. この事実および p が素数のとき, 集合 $G = \{1, 2, \ldots, p-1\}$ の元 $a \in G$, $b \in G$ に対して, $ab \bmod p \in G$ でかけ算を定義することにより, G は位数 $p-1$ の可換な群となることを用いると, 定理 2 の証明ができる. もちろん 1 がこの群の単位元である. G が (可換な) 群であるを示すには, 逆元の存在が非自明である. 次の問題の 1 を示す必要がある.

問題 11　1. p を素数とする. a を 1 以上, $p-1$ 以下の数とするとき,

$$ab = 1 \bmod p$$

となる 1 以上, $p-1$ 以下の数 b が存在する.

2. a, p が互いに素な数なら, $ab = \bmod p$ となる数 b が存在する.

3. b を構成するアルゴリズムを考えよ. その計算量を考察せよ.

ヒント: a と p にユークリッドの互除法を適用せよ. Asir では, 関数 inv を a, p より b を求めるのに利用できる.

```
[346] for (I=1; I<5; I++) print(inv(I,5));
1
3
2
4
```

上の結果をみればわかるように, たしかに $1 \times 1 \bmod 5 = 1$, $2 \times 3 \bmod 5 = 1$, $3 \times 2 \bmod 5 = 1$, $4 \times 4 \bmod 5 = 1$ である.

問題の答え: **答えを書く. 互除法の説明もする.**

4.5 RSA 暗号系の原理

p, q を相異なる素数とし,

$$n = pq, \quad n' = (p-1)(q-1)$$

とおく. e を

$$\gcd(e, n') = 1$$

となる適当な数とする.

$$de = 1 \bmod n'$$

となる数 d をとる. このような d が存在してかつ 互除法アルゴリズムで構成できることは, 問題 11 で考察した.

定理 3 m を n 未満の数とする. このとき $c = m^e \bmod n$ とすると,

$$c^d = m \bmod n$$

が成り立つ.

証明: 定理 2 を $x = m^{q-1} \bmod p$ に対して適用すると,

$$(m^{q-1})^{p-1} = m^{n'} = 1 \bmod p$$

である. 同様の考察を素数 q と m^{p-1} に対しておこなうと,

$$(m^{p-1})^{q-1} = m^{n'} = 1 \bmod q$$

がわかる. $m^{n'} - 1$ は p でも q でも割り切れかつ p と q は相異なる素数であるので $m^{n'} - 1$ は $pq = n$ で割り切れる. よって, $m^{n'} = 1 \bmod n$ が成り立つ.

さて, 証明すべき式は, $(m^e)^d = m \bmod n$ であるが, 仮定よりある整数 f が存在して $ed = 1 + fn'$ が成り立つことおよび $m^{n'} = 1 \bmod n$ を用いると, $(m^e)^d = m^{ed} = m^{1+fn'} = m(m^{n'})^f$ を n で割った余りが m であることがわかる. 証明おわり.

上のような条件をみたす数の組の例としては, たとえば

$$p = 47, q = 79, n = 3713, n' = 3588, e = 37, d = 97$$

がある. 最後の数, d は `inv(37,3588);` で計算すればよい. したがって, 二つの素数 p, q を用意すれば, 簡単に上のような条件をみたす数の組を作れる.

RSA 暗号系では, p, q, d を秘密にし, e, n を公開する. (e, n) を公開鍵, d を秘密鍵と呼ぶ. m (m は n 未満の数) の暗号化は,

$$m^e \bmod n$$

でおこなう. この暗号化されたメッセージの復号化 (もとのメッセージにもどすこと) は, 秘密鍵 d を利用して,

$$m^d \bmod n$$

でおこなう. この計算で正しくメッセージ m が復号できることは, 定理 3 で証明した.

さて, これのどこが暗号なんだろうと思った人もいるかもしれない. e, n が公開されているのなら, n を素因数分解して, p, q を求め, `inv`$(e, (p-1)*(q-1))$ をもとめれば, 秘密鍵 d がわかってしまうではないか! ここで, 素因数分解は最大公約数 (GCD) の計算に比べて, コストのかかる計算だということを思い出してほしい. p, q を十分大きい素数にとると, pq の素因数分解分解の計算は非常に困難になる. したがって p, q の秘密が保たれるのである.

参考: 量子計算機はこの素因数分解を高速にやってしまうということを Shor が示した. これが現在量子計算機がさかんに研究されている, ひとつのきっかけである.

4.6 プログラム

下のプログラムの `encrypt(M)` は文字列 S を RSA 暗号化する. `decrypt(C)` は `encrypt` された結果を元の文字列に戻す. 例を示そう.

```
[356] encrypt("OpenXM");
Block_size = 2
The input message = OpenXM
20336
25966
22605
0

[4113338,3276482,4062967,0]
[357] decrypt(@@);
Block_size = 2
The input message to decrypt
= [4113338,3276482,4062967,0]
20336
25966
22605
0

[OpenXM,[79,112,101,110,88,77]]
```

文字列 “OpenXM” を `encrypt` で暗号化する. 結果は [4113338,3276482,4062967,0] である. これを入力として, `decrypt` を呼び出すと, 文字列 “OpenXM” を復元できる.

`encrypt` はあたえらた文字列をまず アスキーコードの列に変換し, それをブロックに分割してから, 各ブロック m の $m^e \bmod n$ を計算して暗号化する. `20336, 25966, 22605` は各ブロックの m の値である. なお下のプログラムの `PP` が p, `QQ` が q, `EE` が e, `DD` が d(秘密鍵) にそれぞれ対応する.

この実行例では, $p = 1231$, $q = 4567$, $e = 65537$, $d = 3988493$ を利用している.

以下の変数への値の設定プログラムと関数を集めたファイルが rsa.rr である.

```
PP=1231$
QQ=4567$
EE=65537$
DD=3988493$
/*
    PP = 1231, QQ=4567, N=PP*QQ,  N'=(PP-1)*(QQ-1)
    EE = 65537, (gcd(EE, N') = 1),
    DD = 3988493, ( DD*EE = 1 mod N').
 (These values are taken from the exposition on RSA at
  http://www8.big.or.jp/%7E000/CyberSyndrome/rsa/index.html)
 (EE,N) is the public key.
  DD is the private key.  PP, QQ, N' should be confidential
*/
```

```
def naive_encode(S,P,N) {
  /* returns S^P mod N */
  R = 1;
  for (I=0; I<P; I++) {
    R = (R*S) % N;
  }
  return(R);
}
```

```
def encode(X,A,N) {
  R = 1; P = X;
  while (A != 0) {
    if (A % 2) {
      R = R*P % N;
    }
    P = P*P % N;
    A = idiv(A,2);
  }
  return(R);
}
```

```
def encrypt(M) {
   extern EE,PP,QQ;
   E =  EE; N= PP*QQ;
   Block_size = deval(log(N))/deval(log(256));
   Block_size = pari(floor,Block_size);

   print("Block_size = ",0); print(Block_size);
   print("The input message = ",0); print(M);
   M = strtoascii(M);
   L = length(M);
   /* Padding by 0 */
   M = append(M,
        vtol(newvect((idiv(L,Block_size)+1)*Block_size-L)));
   L = length(M);

   C = [ ];  S=0;
   for (I=1; I<=L; I++) {
     S = S*256+M[I-1];
     if (I % Block_size == 0) {
       print(S);
       S = encode(S,E,N);
       C = append(C,[S]);
       S = 0;
     }
   }
   print(" ");
   return(C);
}
```

```
def decrypt(M) {
   extern DD, PP, QQ;
   D = DD; N = PP*QQ;
   Block_size = deval(log(N))/deval(log(256));
   Block_size = pari(floor,Block_size);

   print("Block_size = ",0); print(Block_size);
   print("The input message to decrypt = ",0); print(M);
   L = length(M);

   C = [ ];
   for (I=0; I<L; I++) {
     S = encode(M[I],D,N);
     print(S);
     C1 = [ ];
     for (J=0; J<Block_size; J++) {
       S0 = S % 256;
       S = idiv(S,256);
       if (S0 != 0) {
         C1 = append([S0],C1);
       }
     }
     C = append(C,C1);
   }
   print(" ");
   return([asciitostr(C),C]);
}
end$
```

encode(X,A,N) は, $\mathtt{X}^{\mathtt{A}} \bmod \mathtt{N}$ を計算する関数である. native_encode(X,A,N) は定義どおりにこの計算をする関数である. この関数をためしてみればわかるように, 工夫してこの計算をしないと大変な時間がかかる. A を 2 進展開して計算しているのが, encode(X,A,N) である. 実行時間を比べてみてほしい.

A を 2 進展開し

$$\sum a_i 2^i, \quad (a_i = 0 \text{ or } 1)$$

なる形にあらわすと,

$$\mathtt{X}^{\mathtt{A}} = \prod_{i\,:\,a_i \neq 0} \mathtt{A}^{2^i}$$

とかける. encode(X,A,N) では, A, $\mathtt{A}^2$, $\mathtt{A}^4$, ... を N でわった余りを順番に計算して変数 P にいれている. あとは, 2 進展開を利用して

$$\mathtt{X}^{\mathtt{A}} \bmod \mathtt{N} = \prod_{i\,:\,a_i \neq 0} \mathtt{A}^{2^i} \bmod \mathtt{N}$$

を計算している.

encrypt では, あたえらた文字列をまずアスキーコードに変換して, 変数 M にいれている. Block_size を b とするとき, まず,

$$\mathtt{M}[0]256^{b-1} + \mathtt{M}[1]256^{b-2} + \cdots + \mathtt{M}[\mathtt{b}-1]256^0$$

を変数 S に代入し, この S に対して, $\mathtt{S}^{\mathtt{E}} \bmod \mathtt{N}$ を計算する. この操作を各ブロック毎に繰り返す.

decrypt は encrypt とほぼ同様の操作なので説明を省略する.

さて次の問題として, RSA 暗号化システムのための公開鍵 (n, e) および秘密鍵 d を生成する問題がある. 次のプログラム rsa-keygen.rr は, これらの数を生成し, 変数 EE, DD などに設定する.

```
def rsa_keygen(Seed) {
  extern PP,QQ,EE,DD;
  random(Seed);
  do {
    P = pari(nextprime,Seed);
    Seed = Seed+P;
    Q = pari(nextprime,Seed);
    PP = P;
    QQ = Q;
    Phi = (P-1)*(Q-1);
    E = 65537;
    Seed = Seed+(random()*Q % Seed);
  } while (igcd(E,Phi) != 1);
  EE = E;
  DD =inv(EE,Phi);
  print("Your public key (E,N) is ",0); print([EE,PP*QQ]);
  print("Your private key D is ",0); print(DD);
  return([PP,QQ,EE,DD]);
}
end$
```

次の例は, $2^{128} = 340282366920938463463374607431768211456$ 程度の大きさの素数を 2 個生成して, RSA の公開鍵, 秘密鍵 を作る例である. なお, この程度の大きさの素数の積は最新の理論とシステムを用いると容易に因数分解可能である.

```
[355] load("rsa.rr")$
[356] load("rsa-keygen.rr")$
[359] rsa_keygen(2^128);
Your public key (E,N) is [65537,
231584178474632390847141970017375815766769948276287236111932473531249232711409]
Your private key D is
19961886913057446009652405554498340187104891091301936388575383184168509927206 1

[340282366920938463463374607431768211507,
68056473384187692692674921486353642298 7,
65537,
19961886913057446009652405554498340187104891091301936388575383184168509927206 1]
```

```
[360] encrypt("Risa/Asir");
Block_size = 32
The input message = Risa/Asir
372759688465508844469111436918076915836368359054402083770354411365009352294 40

[146634940900113296504342777649966848592634201106623057430078652022991264082696]
[361] decrypt(@@);
Block_size = 32
The input message to decrypt =
[146634940900113296504342777649966848592634201106623057430078652022991264082696]
372759688465508844469111436918076915836368359054402083770354411365009352294 40

[Risa/Asir,[82,105,115,97,47,65,115,105,114]]
```

高速に安全な公開鍵 (n, e) および秘密鍵 d を生成する問題は, RSA 暗号ファミリを利用するうえでの一つの中心的問題である. たとえば, $p-1$, $q-1$ が小さい素数の積に分解する場合は, 比較的高速な n の素因数分解法が知られている. つまりこのような (p, q) から生成した鍵はこの素因数分解法の攻撃に対して脆弱である. 上の関数 `rsa_keygen` はこのような攻撃に対する脆弱性がないかのチェックをしていない. その他, さまざまな攻撃法に対する, 脆弱性がないかのチェックが必要となる.

Risa/Asir は, 楕円曲線の定義する可換群をもちいる暗号系である, 楕円暗号系に関して, 安全なこれらのパラメータを生成するシステムの基礎部分として実際に利用されている. Risa/Asir に組み込まれている, 大標数有限体の計算機能および高速な 1 変数多項式の計算機能はこれらに必要な機能として開発された.

問題 12 上のプログラムでは, 文章をブロックに分けてから, RSA 暗号化している. 1 byte づつ暗号化すると比較的容易に暗号が解読できる. その方法を考察せよ.

問題 13 公開鍵 $(e, n) = (66649, 24691358025875308641989 47)$ を用いて, 関数 `encrypt` で, ある文字列を変換したら,
`[534331413430079382527551,486218671433135535521840]`
を得た. どのような文字列だったか?

4.7 プログラムの詳しい説明

まだ書いてない

4.8 Risa/Asir の Windows 版について

まだ書いてない

関連図書

[1] Risa/Asir ドリル (これは Free Book です)

第5章　Macaulay2

高山

Macaulay2 は可換環論や D-加群の計算を行う以外に非常に美しいユーザ言語も付属しているシステムである.

5.1　Macaulay2 の基本

まず Macaulya2 を対話型電卓として利用する方法を説明する.

Macaulay は数の処理のみならず, 多項式の計算や環論の不変量の計算もできる. 電卓的に使うための要点を説明し例をあげよう.

例 19 `+,-,*,/,^` はそれぞれ足し算, 引き算, かけ算, 割算, 巾乗. たとえば,

```
2*(3+5^4)
```

と入力すると $2(3+5^4)$ の値を計算して戻す.

例 20 多項式の計算をするにはまず環の定義をする. それから計算する.

```
i3 : R = QQ[t,x,y,z]

o3 = R

o3 : PolynomialRing
```

ここで `R : PolynomialRing` は R が PolynomialRing オブジェクトであることを示している. R ↵と入力すると R と PolynomialRing と戻るのみである. このように Macaulay 2 では 変数名 ↵と入力しても値が戻るとは限らないことに注意. R の中身を詳しく見るには, `describe(R)` または `describe R` コマンドを用いる.

一般に Macaulay 2 では, 変数 ↵ とすると, 値 ： オブジェクトの名前 を表示するが, 値が複雑な場合は適宜簡潔な表現方法が選択される.

```
i6 : f=x-t^2

        2
o6 = - t  + x
o6 : R

i7 : g=y-t^3

        3
o7 = - t  + y
o7 : R

i8 : h=z-t^4

        4
o8 = - t  + z
o8 : R

i9 : f+g

        3    2
o9 = - t  - t  + x + y
o9 : R
```

この例では ...

例 21 さて前の例で定義した多項式 f, g, h から t を消去してみよう. 消去は t 変数の重みを 1 に, 他の変数の重みを 0 としてグレブナ基底を計算することにより実行できる.

```
i12 : R = QQ[t,x,y,z, Weights => {{1,0,0,0}}]

o12 = R

o12 : PolynomialRing

i13 : f=x-t^2; g=y-t^3; h=z-t^4;

i16 : curve=ideal(f,g,h)

                2         3         4
o16 = ideal (- t  + x, - t  + y, - t  + z)

o16 : Ideal of R

i17 : gb(curve)

o17 = | y2-xz x2-z tz-xy ty-x2 tx-y t2-x |

o17 : GroebnerBasis
```

この結果をみればわかるように

$$y^2 - xz, x^2 - z$$

が t を消去して得られる多項式であり, その幾何的意味は ... **まだ書いてない**

さて Macaulay2 ではさまざまな環論の不変量を計算するための関数があらかじめ組み込まれている.

例 22 上の curve の (Krull) 次元を計算せよ.

```
i18 : dim curve
o18 = 1

i19 : degree curve
o19 = 4
```

なお 上の dim curve は dim(curve) と書いてもよい. また 上の degree curve は degree(curve) と書いてもよい.

例 23 モノミアルで生成されるイデアルの極小自由分解の計算をしてみよう. この機能により組み合わせ論と可換環論の交錯する魅力的な分野での数学実験が可能となる.

```
i20 : S=QQ[a,b,c,d];

i21 : I=monomialIdeal(a^2,a*b,b^3,a*c);

o21 : MonomialIdeal of S

i22 : res I

       1      4      4      1
o22 = S  <-- S  <-- S  <-- S  <-- 0

      0      1      2      3      4

o22 : ChainComplex
```

上の $1, 4, 4, 1$ を betti 数とよぶ. betti 数に関する数学の定理... **まだ書いてない**

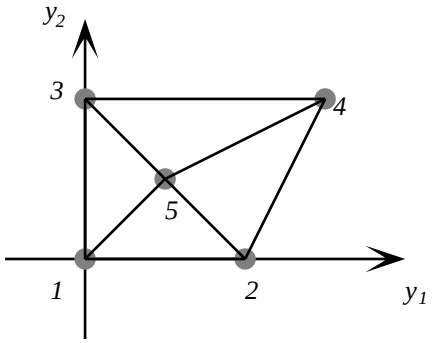
5.2 Macaulay2 例題集

5.2.1 Macaulay2 でのプログラミング

`OpenXM/src/Macaulay2` 提供可能.

5.2.2 Macaulay2 を用いて多面体を三角形に分割する

ここでは図 5.2.2 に表示された 5 点を考える. この 5 点の凸包は四角形 1,2,3,4 であるが, この 5 点とその重み $(1, 1, 1, 1, 0)$ を用いた正則三角形分割を計算してみよう.



```
i1 : load "LLL.m2";
i2 : load "triangulate.m2";
--loaded triangulate.m2

i3 : A={{1,1,1,1,1},{0,2,0,3,1},{0,0,2,2,1}}

o3 = {{1, 1, 1, 1, 1}, {0, 2, 0, 3, 1}, {0, 0, 2, 2, 1}}

o3 : List

i4 : I = toricIdeal A

                    2   3 2      4
o4 = ideal (b*c - e , a d  - b*e )

o4 : Ideal of R
```

変数 a,b,c,d,e がそれぞれ点 1,2,3,4,5 に対応している. 環 R は関数 toricIdeal の中で自動的に定義される.

次のコードは M.Stillman, B.Sturmfels, R.Thomas **??** による点集合で定義される affine toric 多様体を計算するプログラムである.

```
-- M2 book page 181.
toBinomial = (b,R) -> (
  top := 1_R; bottom := 1_R;
  scan(#b, i -> if b_i > 0 then top = top * R_i^(b_i)
                else if b_i < 0 then bottom = bottom * R_i^(-b_i));
  top - bottom);

toricIdeal = (A) -> (
  n := #(A_0);
  R = QQ[vars(0..n-1), Degrees=>transpose A, MonomialSize=>16];
  B := transpose LLL syz matrix A;
  J := ideal apply(entries B, b-> toBinomial(b,R));
  scan(gens ring J, f -> J = saturate(J,f));
  J );
```

このコードの解説はまだ書いてない.

```
i6 : R2=QQ[a,b,c,d,e,Weights=>{{1,1,1,1,0}}]

o6 = R2
o6 : PolynomialRing

i10 : F=map(R2,R)

o10 = map(R2,R,{a, b, c, d, e})
o10 : RingMap R2 <--- R

i11 : I2 = F(I)                    -- I2 は新しい順序の入った ring でのイデアル.

                    2   3 2      4
o11 = ideal (b*c - e , a d  - b*e )

o11 : Ideal of R2

i17 : M=monomialIdeal I2             -- M に I2 の initial ideal を.

                          3 2
o17 = monomialIdeal (b*c, a d )

o17 : MonomialIdeal of R2


i19 : MM=radical M                   -- MM は M の radical

o19 = monomialIdeal (b*c, a*d)

o19 : MonomialIdeal of R2

i20 : primaryDecomposition MM  --MM を準素イデアル分解して三角形分割を得る.

o20 = {monomialIdeal (a, b), monomialIdeal (a, c), monomialIdeal (b, d),
  monomialIdeal (c, d)}

o20 : List
```

さて, (a,b) の補集合は (c,d,e), 頂点のインデックスでは $(3,4,5)$,
さて, (a,c) の補集合は (b,d,e), 頂点のインデックスでは $(2,4,5)$,
さて, (b,d) の補集合は (a,c,e), 頂点のインデックスでは $(1,3,5)$,
さて, (c,d) の補集合は (a,b,e), 頂点のインデックスでは $(1,2,5)$ である. 頂点のインデックスが図 5.2.2 の三角形の頂点と見事に対応していることが観察できるだろう.

多面体の幾何と可換環論—数学的な背景

まだ書いてない

5.3 グレブナ基底関連システムのコマンド対応表

Knoppix/Math にはグレブナ基底や可換環論のためのシステムが数通りはいっている．これらの間のコマンドを比較してみよう．

5.3.1 環の定義

たとえば $\mathbf{Q}[x,y,z]$ を x の重みが 10, y の重みが 5, z の重みが 0 で順序づけたモノミアル順序の環.

Macaulay2 では

```
QQ[x,y,z,Weights==>{{10,5,0}}]
```

Singular では

Risa/Asir では

```
[1153] Opt=[["v",[x,y,z]], ["order",[[x,10,y,5]]]];
 [[v,[x,y,z]], [order,[[x,10,y,5]]]];
[1154]  dp_gr_main([x^2+y^2+z^2-1,x*y*z-1,x*y+y*z+z*x-3] | option_list=Opt);
```

とグレブナ基底計算等をおこなうと自動的に現在の環が設定され, 分散表現多項式がこの環に属することとなる.

Kan/sm1 では

```
[(x,y,z) ring_of_polynomials [[(x) 10 (y) 5 (z) 0]] weight_vector 0]
 define_ring
```

5.3.2 normal form の計算

Macaulay2 では　%　, Risa/Asir では **true_nf** , Kan/sm1 では **reduction** , Singular では ...
まだ詳しく書いてない

関連図書

[1] David Eisenbud, Daniel R.Grayson, Michael Stillman, Bernd Sturmfels 編, Computations in Algebraic Geometry with Macaulay 2, Springer, 2002.

第6章　微分作用素環での計算

高山, 小原

Knoppix/Math には微分作用素環での計算をするソフトウエアが各種入っている.

Risa/Asir ファミリでは bfct, yang, dp_gr_main コマンド等があり, Macaulay2 や kan/sm1 には専門のパッケージがついている.

6.1　解空間の次元の計算

6.2　偏微分方程式系に潜む常微分方程式を探す

6.3　b-関数の世界

6.4　コホモロジ群の計算

第7章　Knoppix/Math, OpenXMで分散計算

高山

OpenXM は数学に関する通信をやるための基礎技術である. OpenXM は数学に関する通信をするためのプロトコルを提案している (OpenXM-RFC's). OpenXM では実際にいろいろな数学ソフトウエアをつないでパッケージを作成する実験をやっており, そのパッケージを配布している. Knoppix/Math にはこの OpenXM パッケージが搭載されている. Risa/Asir を一台の計算機で動かす時もこの OpenXM を基礎として様々なコンポーネントを動作させているが, OpenXM はネットワーク透過的なプロトコルであり, 複数台の計算機を協調的に動作させることも可能である.

7.1　sshの準備

2 台の Knoppix/Math を動かした計算機で OpenXM による通信と分散計算をためしてみよう. 準備として ssh でお互いに自由自在に相互接続できるようにしないといけない.

一台目の計算機を knoppix-main, 二台目の計算機を knoppix-2-go (knoppix 2 号) と呼ぶことにする. knoppix-main も knoppix-2-go もブロードバンドルーター等から IP アドレスを自動的に取得している (DHCP によるアドレスの自動取得) ものと仮定する. IP アドレスを調べるにま `ifconfig` コマンドを用いる (後述).

仮定 1

計算機名	IP アドレス
knoppix-main	192.168.1.1
knoppix-2-go	192.168.1.2

IP アドレスの値は環境によりかわるので注意.

仮定 2

以下 ユーザ knoppix のパスワードは `abcd30xyz`, ssh のパスフレーズも `abcd30xyz` とする.

仮定 3

knoppix-main, knoppix-2-go 共に asir のネットワークダウンロードは完了している.

knoppix-2-go 側の設定.

1. ペンギン ⇒ services ⇒ Start SSH server で sshd をスタートする. パスワードを `abcd30xyz` に設定.
2. Shell を起動. `mkdir ~/.ssh ; chmod 700 .ssh` コマンドで ssh の公開鍵を格納するフォルダを作る.

knoppix-main 側の設定.

1. Shell を起動. `ssh-keygen -t dsa` コマンドでパスフレーズを設定. `abcd30xyz` を入力.
2. ファイル `.ssh/id_dsa` と `.ssh/id_ida.pub` が生成される. それぞれ公開鍵と秘密鍵である. 公開鍵を knoppix-2-go に転送して knoppix-main から knoppix-2-go へ ssh で接続できるようにする. それには Shell で, scp コマンドでコピーする. `scp ~/.ssh/id_dsa.pub 192.168.1.2:.ssh/authorized_key` パスワードを聞かれるので, `abcd30xyz` を入力.

さて以上で準備が完了した. usb メモリ等にホームディレクトリを作ってホームを保存している場合は次回はそれを利用すれば十分である.

さて, 接続できるかテストしてみよう.

knoppix-main 側より.

```
ssh-agent bash
ssh-add
   ここでパスフレーズを聞かれるので abcd30xyz を入れる.
ssh 192.168.1.2
無事 knoppix-2-go へ接続できたら, ifconfig を入力してみよう.
  192.168.1.2 が出力されたら, OK である.
接続を切断するには exit を入力する.
```

いよいよ asir と OpenXM による分散計算である.

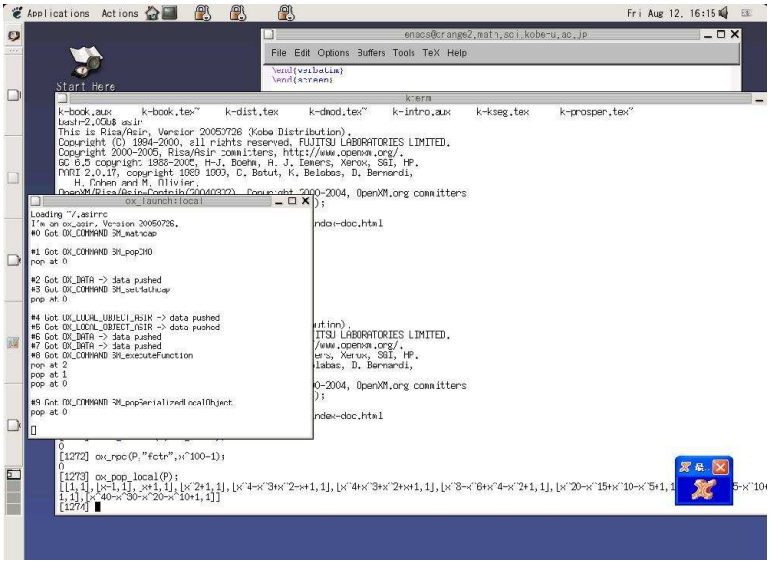
knoppix-main 側より.

```
ssh-agent bash
ssh-add
   ここでパスフレーズを聞かれるので abcd30xyz を入れる.
export ASIRHOSTNAME="192.168.1.1"
asir で asir を起動.
ox_launch("192.168.1.2","/usr/local/OpenXM/lib/asir","ox_asir");
```

無事 192.168.1.2 側 (knoppix-2-go 側) の asir が起動すると, 画面のようになる. rpc (Remote Procedure Call) をためしてみよう.

7.2 計算例

```
P=ox_launch("192.168.1.2","/usr/local/OpenXM/lib/asir","ox_asir");
ox_rpc_cmo(P,"fctr",x^100-1);
ox_pop_cmo(P);
```



別のマシンで因数分解が実行されてるが, どうやって信じてもらおうか.

```
ox_execute_string(P,"system(\"sudo halt\");");
```

とやると, knoppix-2-go が shutdown されるはずで, これで信じてもらおうか?

まだ途中.

第8章　Prosper 解説, 有馬温泉の巻

小原

この章は PDF ファイルとして後ろに結合してある.

8.1　Prosper で講演スライドを作成. overlay しない場合

8.2　Prosper で講演スライドを作成. overlay をする場合

8.3　Pstrick の利用方法

ToConsider:
2. メニュー等の参照方法
3. スタイルは 例 (例題), 問題, 定理 等でいいか?
4. この題名はまじめなタイトルではない.
5. サンプルプログラムの背景の数学, サンプルプログラムの解説にてっするか?
6. プログラムはしない. one line command の解説のみか?
7. Knoppix と他のコンピュータとのファイルのやりとり

KSEG で遊ぶ平面幾何

濱田 龍義

2005 年 8 月 6 日

1 序

ここでは、Ilya Baran 氏が作成したソフトウェア “KSEG” の紹介を行ないます。“KSEG” は、“Interactive Geometry” と呼ばれる種類のソフトウェアです。Interactive Geometry という言葉を日本語に訳すと「対話式の幾何学」という意味でしょうか？ コンパスや定規の代わりにコンピュータと対話をしながら図形を描くソフトウェアのことです。単にコンパスと定規の代わりならば、コンピュータを使う意味はありません。KSEG を使うと、図形の性質を保ったまま、変形、回転、移動を行なえます。また、点の軌跡を描く機能があるので、様々な平面曲線を描くことができます。

2 KNOPPIX について

皆さんの前にあるコンピュータでは、KNOPPIX/Math というシステムが動いています。初めて名前を聞く方がほとんどだと思いますが、KNOPPIX は、Linux と呼ばれるシステムの一種です。Windows が動いているコンピュータに KNOPPIX の CD を入れて再起動するだけで、すぐに Linux システムを使えるようになります。元々、KNOPPIX はドイツの Klaus Knopper さんが開発しました。独立行政法人産業技術総合研究所（以下、産総研）で日本語化を行なっています。皆さんの前にある KNOPPIX/Math は産総研の須崎有康氏、飯島賢吾氏と福岡大学の濱田との共同研究成果です。

KNOPPIX/Math は KSEG の他にも、数学に関係するソフトウェアを大量に収録しています。実習に使った KNOPPIX の CD は持ち帰っていただいて構いませんので、家にコンピュータがある人は、是非試してみてください。解説文書はデスクトップの「数学デモ」というフォルダに収録されています。

3 KNOPPIX/Math の使い方

収録されているソフトウェアを起動するためには、メニューから起動する方法が簡単でしょう。
画面左下の左から 2 番目の $\sqrt{x}$ というアイコンをマウスでクリックしてください。

図 1 KDE パネル

この中から起動したいソフトウェアを選択して、マウスをクリックすれば、数学に関連するソフトウェアが起動します。メニューはアルファベット順に並んでいます。

4 KSEG について

KSEG を起動しましょう。先ほどの $\sqrt{x}$ メニューの中から KSEG(KSEG) を選んで、マウスでクリックしてください。下の図のようなウィンドウが表示されます。

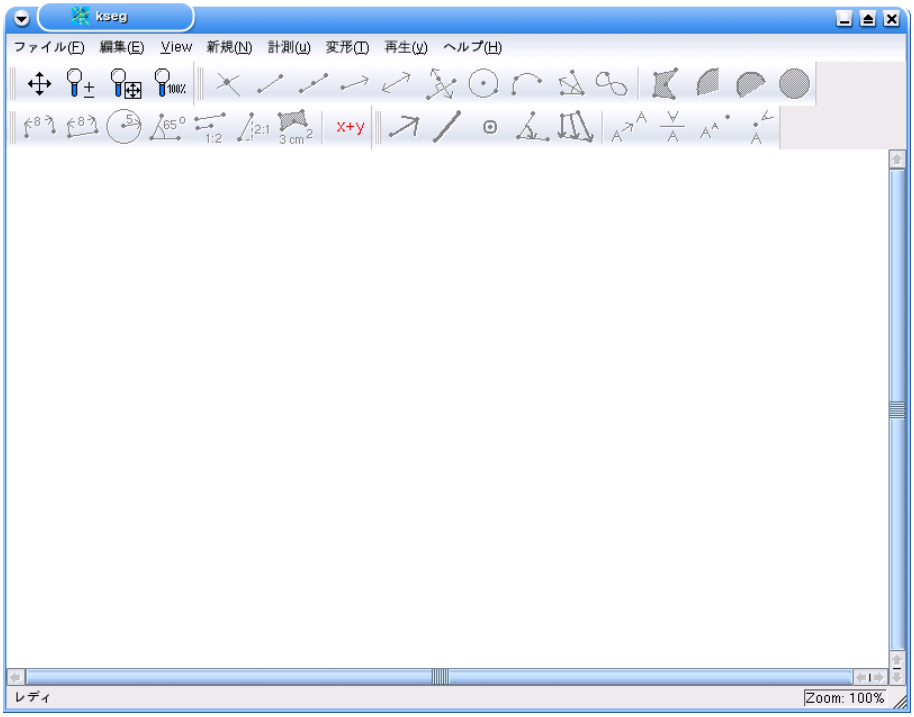


図 2 KSEG 画面

ウィンドウ上部にメニューが配置され、その下にはボタン型のアイコンがあります。ボタンの絵を見れば、おおよその見当はつくのではないでしょうか？ヘルプをクリックすると日本語に翻訳された解説を読むこともできます。詳しい使い方を知りたい時は、このヘルプファイルを読むと良いでしょう。

4.1 KSEG の使い方

基本的な使い方については、次の 3 つのことだけ押えておけば十分です。

1. 右クリックで点を描画
2. 左クリックで点や線、円を選択（矩形選択や shift キーを用いた複数選択も可能）
3. 図形作成に必要な点や線を選択後、メニューもしくはボタンで図形を作成

例えば、一般に平面内で線分を決定するためには 2 点が必要です。従って、KSEG で線分を描く時は、次のような手順を踏みます。

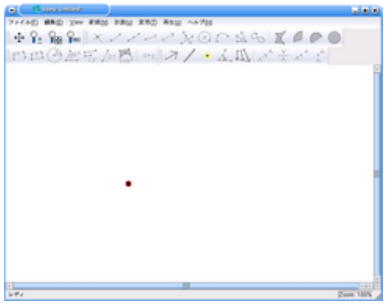


図 3　適当な場所で右クリック

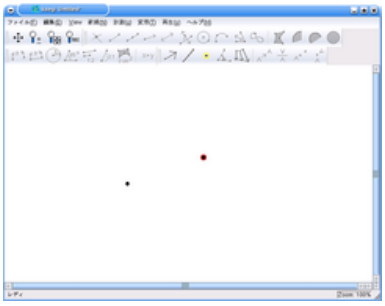


図 4　さらに別な場所で右クリック

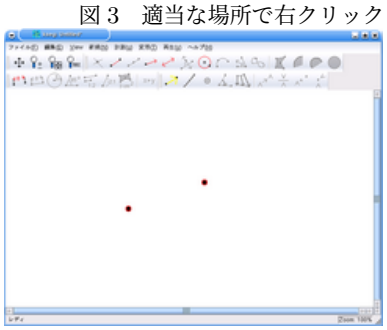


図 5　Shift キーを押しながら 2 点を選択

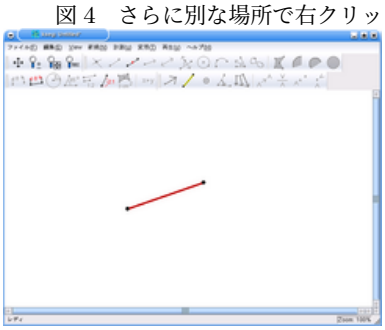


図 6　メニューから 「新規」 → 「線分」

同様に「線分」の代わりに「直線」や「半直線」を選択することもできます。もちろん、ボタンで指定しても構いません。

また、KSEG で円を描くためには、次のような手順を踏みます。

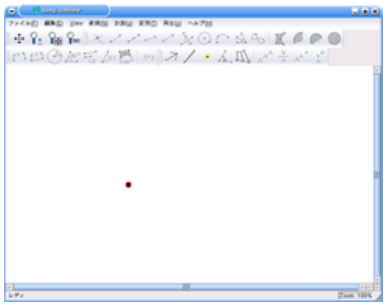


図 7　適当な場所で右クリック

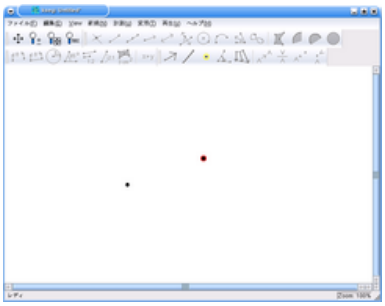


図 8　さらに別な場所で右クリック

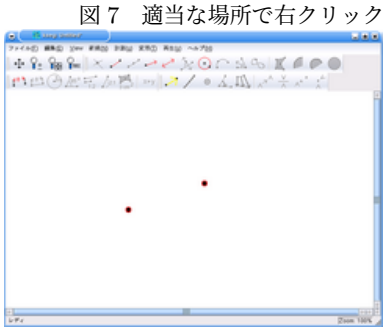


図 9　Shift キーを押しながら 2 点を選択

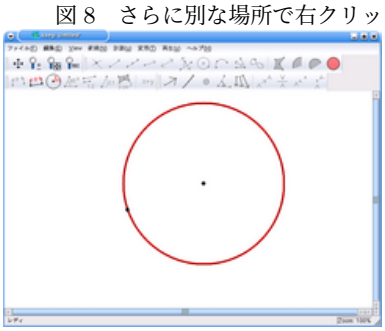


図 10　メニューから 「新規」 → 「円」

最初に選択した点を中心にして、もう一つの点を通る円が描かれます。

おそらく、ボタンに描いてある絵をみれば、どのような作図ができるか、予想できるのではないでしょうか？

4.2 三角形を描く

三角形を描くときは、マウスによる矩形選択を利用すると便利です。

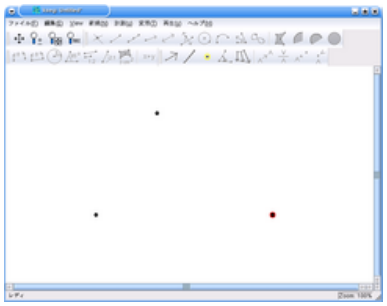


図 11　三角形を描くように 3 点を描く

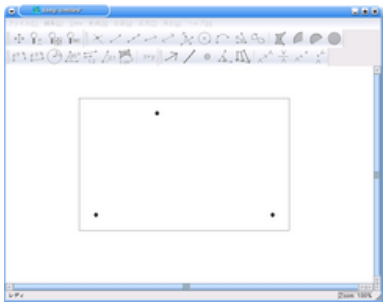


図 12　3 点を囲むようにマウスをドラッグ

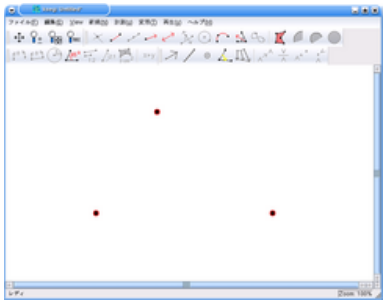


図 13　3 点が選択されました。

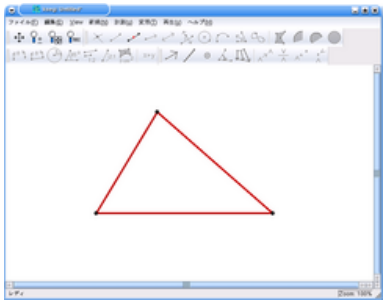


図 14　メニューから「新規」 → 「線分」

4.2.1 三角形の五心

三角形は内心、外心、垂心、重心、傍心をもちます。これらをあわせて五心とも呼びます。

内心　三角形の 3 つの内角の二等分線は 1 点で交わる。この点を内心と呼ぶ。

外心　三角形の 3 辺の垂直二等分線は 1 点で交わる。この点を外心と呼ぶ。

垂心　三角形の 3 つの頂点からそれぞれの対辺に引いた垂線は 1 点で交わる。この点を垂心と呼ぶ。

重心　三角形の頂点とその対辺の中点を結ぶ 3 つの線分は 1 点で交わる。この点を重心と呼ぶ。

傍心　三角形の 1 つの内角と他の 2 つの外角の二等分線は 1 点で交わる。この点を傍心と呼ぶ。三角形に傍心は 3 つある。

KSEG では線分の長さを計ったり、計算を行なうこともできます。線分の比を計算することで、五心の性質を確認することもできます。ここでは、重心の描き方を解説します。残りの内心、外心、垂心、傍心については、自分で描き方を考えてみてください。

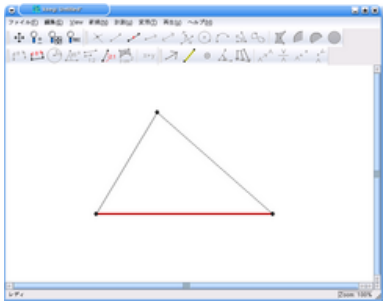


図 15　三角形の辺の一つをマウスでクリック

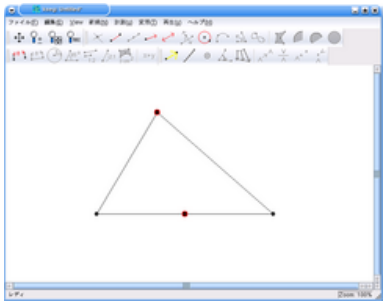


図 17　三角形の頂点の一つと、その対辺の中点を Shift キーを押しながらマウスでクリック

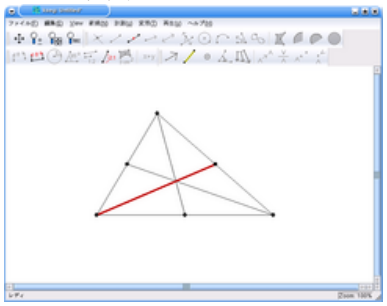


図 19　他の 2 個の三角形の頂点に対しても同様の操作

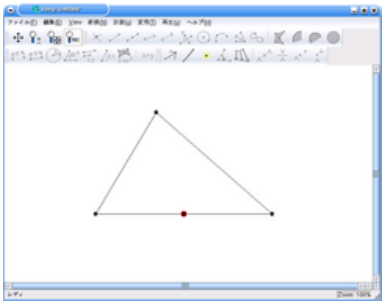


図 16　メニューから「新規」 → 「中点」

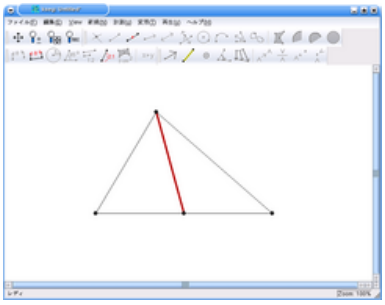


図 18　メニューから「新規」 → 「線分」

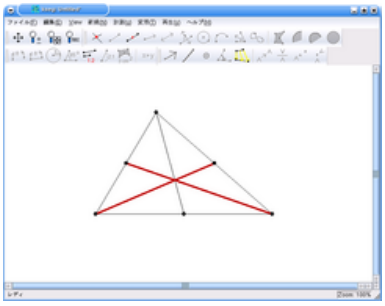


図 20　三角形の中に書かれた線分を 2 本選択

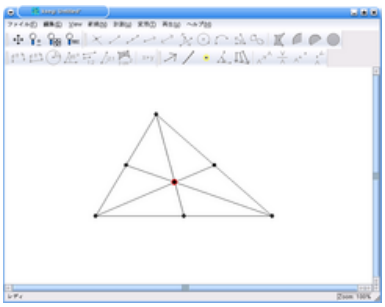


図 21　メニューから「新規」→ 「交点」

4.3 曲線を描く

KSEG は点の軌跡を描くことができます。

ここでは、放物線 (例:$y = x^2$) や、正弦曲線 (例:$y = \sin x$) を描きます。まず、最初に x 軸、y 軸の描き方を述べておきます。

4.3.1 水平線と垂直線を描く

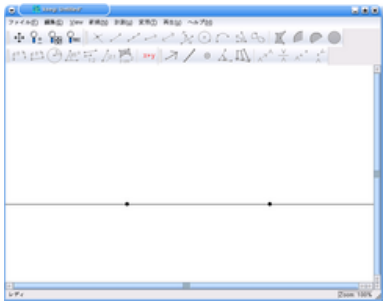


図 22　2 点を水平な位置に描き、直線を描く

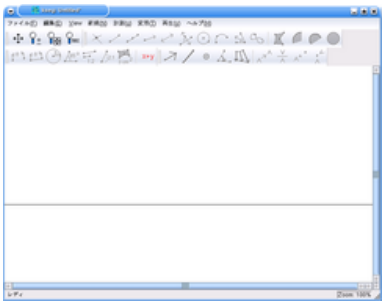


図 23　2 点だけ選択後、「編集」→「オブジェクトを隠す」

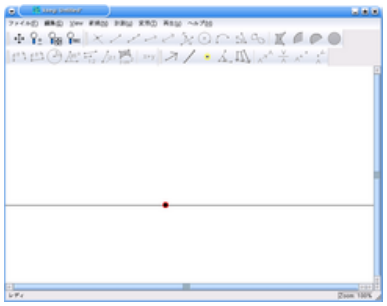


図 24　水平な直線上の好きな場所に点を描く

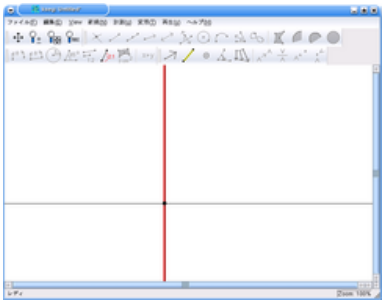


図 25　点と直線を選択し、「新規」→「垂線」

4.3.2 x 軸上を動く駆動点を描く

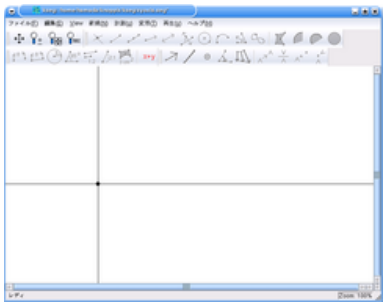


図 26　xy 座標を描く

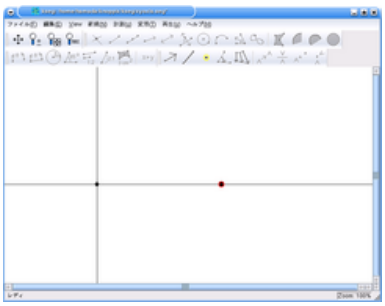


図 27　x 軸上に点を描く

点は x 軸上を移動するでしょうか。点をマウスでドラッグして、移動してみてください。

4.4　放物線を描く

水平線を x 軸、垂直線を y 軸に見立てて、交点を原点と思います。放物線は図形的には “定直線 l と、l 上にない定点 F から等しい距離にある点の軌跡” として定義されます。ここでは、原点を通る放物線を描きます。

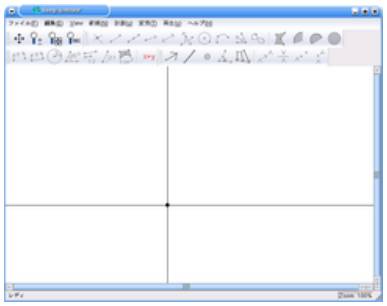


図 28　xy 座標を描く

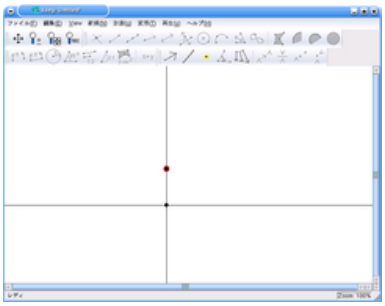


図 29　y 軸上に点を描く。これが焦点

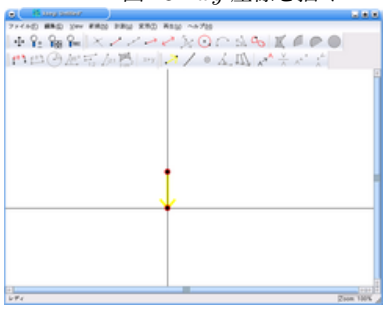


図 30　2 点を上から順番に選択して「変形」 → 「ベクトルを選択」

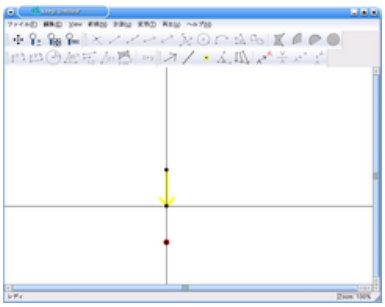


図 31　原点だけを選択して、「変形」 → 「変換」

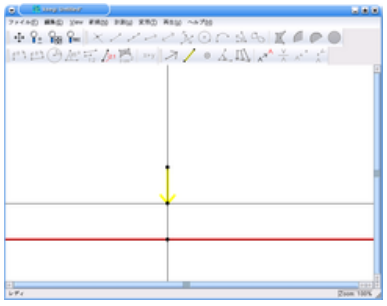


図 32　y 軸と y 軸上の負の部分にある点を選択して、「新規」 → 「垂線」

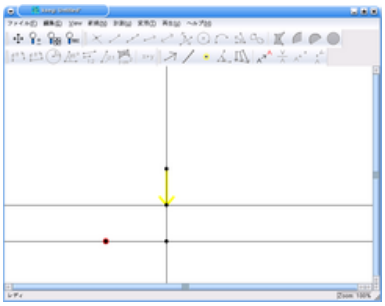


図 33　x 軸に平行な直線上に駆動点を作成

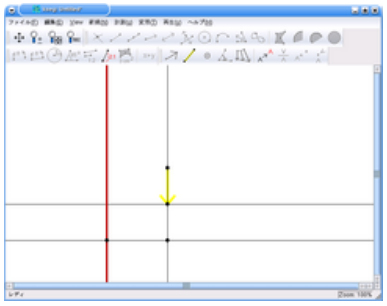


図 34　x 軸に平行な直線と駆動点を選択して、「新規」 → 「垂線」

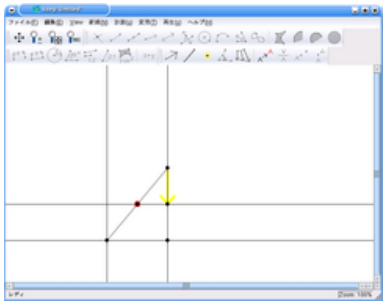


図 36　線分を選択して、「新規」 → 「中点」

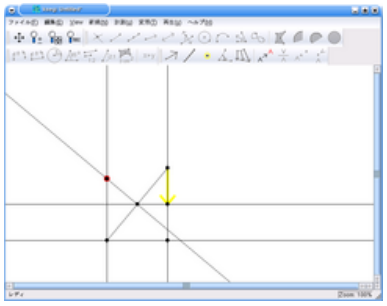


図 38　2 直線の交点を選択

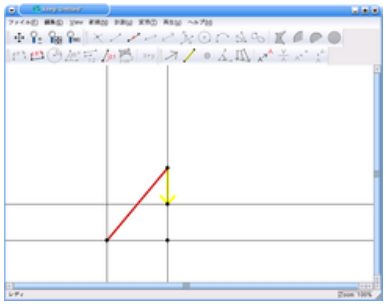


図 35　焦点と駆動点を選択して、「新規」 → 「線分」

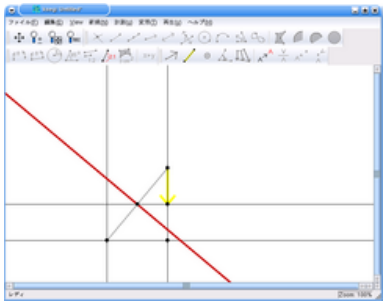


図 37　線分と中点を選択して、「新規」 → 「垂線」

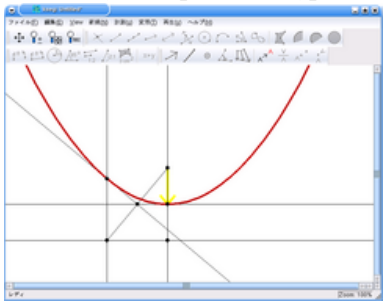


図 39　Shift キーを押しながら駆動点も選択して、「新規」 → 「軌跡」

4.5　正弦曲線を描く

放物線を描く方法の中で、「変形」という機能を用いました。「変換」の他にも、「鏡映」、「倍率」、「回転」などの操作が可能です。ここでは、「回転」という機能を用いて、正弦曲線を描きます。[*1]

[*1] ここで紹介した方法では、y 軸に関して対称なグラフ $y = \sin|x|$ が描かれます。駆動点の動く範囲を $x > 0$ に限定すれば、皆さんの見慣れたグラフが得られます。

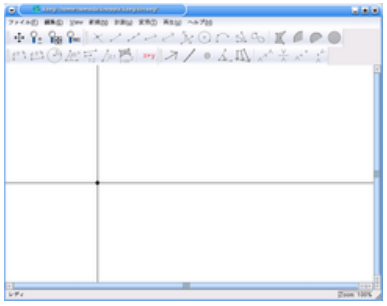


図 40　xy 座標を描く

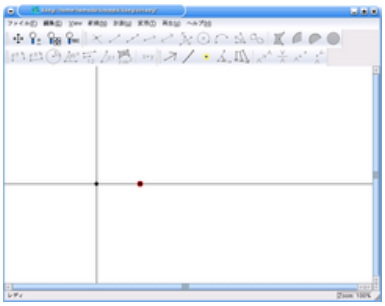


図 41　x 軸上に点を描く

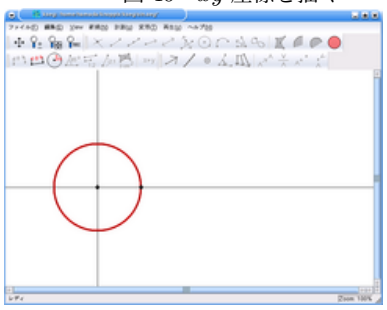


図 42　原点を中心として x 軸上の点を通る円を描く

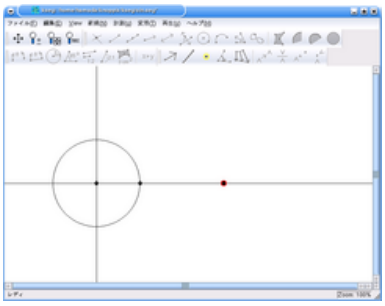


図 43　x 軸上にもう一つ点を描く。これを駆動点とする。

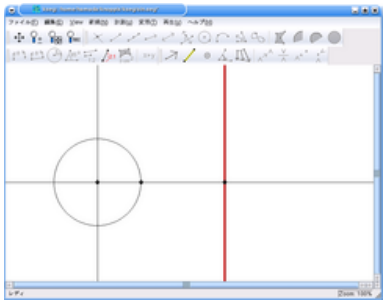


図 44　新たに作成した点と x 軸を選択して「新規」→「垂線」

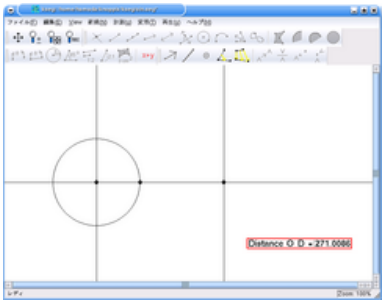


図 45　原点と x 軸上の駆動点を選択して「計測」→「距離」

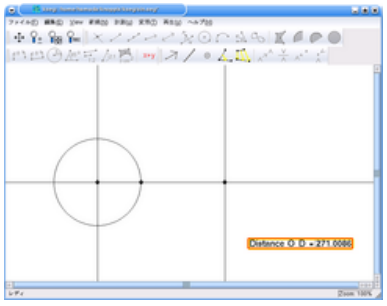


図 46　計測された距離を選択して、「変形」→「角度を選択」

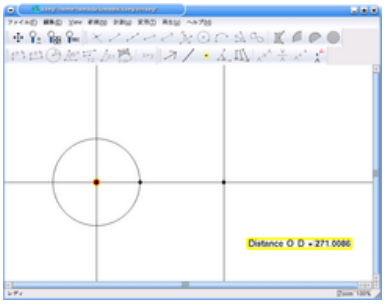


図 47　原点を選択して、「変形」→「中心を選択」

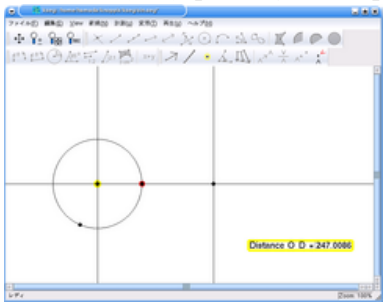


図 48　x 軸と円との交点を選択して、「変形」→「回転」

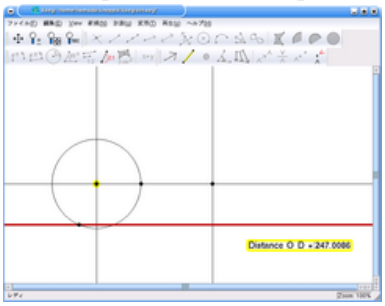


図 49　円上の動点と y 軸を選択して、「新規」→「垂線」

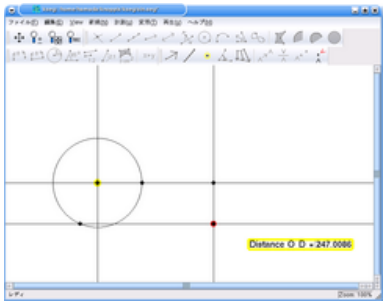


図 50　2 垂線を選択して、「新規」 → 「交点」

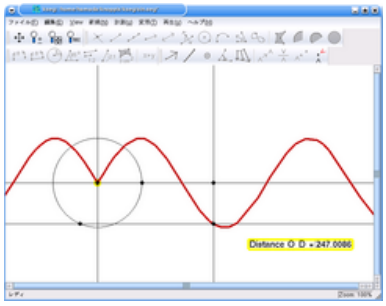


図 51　駆動点と交点を選択して、「新規」 → 「軌跡」

ここまでに、紹介してきた機能は KSEG の持つ機能の一部だけですが、十分に楽しめるのではないかと思います。いくつか課題を出しておきましょう。解答は一つとは限りません。

1. KSEG を用いて放物線を描く別の方法は？
2. KSEG を用いて楕円を描く方法は？
3. KSEG を用いて双曲線を描く方法は？
4. KSEG を用いて余弦曲線 $y = \cos x$ を描く方法は？
5. KSEG を用いて正接曲線 $y = \tan x$ を描く方法は？

福岡大学理学部応用数学教室では、「数学」と「情報」の教員免許を取得できます。教員免許を取得するための必修科目として「数学科教育法」という科目があります。以下に紹介するのは、「数学科教育法 I」におけるレポート課題です。全て、現在、応用数学科に在籍している学生が考えた問題です。

コンパスと定規のみをもちいて、次の作図問題の 2 題以上に答えよ。（作図不能なものもあるかもしれない。）

1. 任意に与えられた三角形に相似であって、面積が 2 倍となる三角形を作図せよ。
2. 円の面積を 3 等分する直線を引きなさい。
3. 等しい半径をもつ三つの円が与えられた円に内接し、これらは互いに外接する。この 3 つの円を作図せよ。
4. $\angle AOB$ の二等分線を OC とし、OC 上に点 P をとる。点 P を通り OA と OB に接する円を 2 つ描け。
5. 一辺が 1 の正方形と同じ面積の正五角形を作図せよ。
6. 任意に与えられた四角形の面積と同じ面積をもつ三角形の作図法を述べよ。

参考文献

[1] シンデレラ 幾何学のためのグラフィックス J. リヒター–ゲバート, U.H. コルテンカンプ著, 阿原 一志訳, ISBN4-431-70966-5

[2] シンデレラで学ぶ平面幾何, 阿原 一志著, ISBN4-431-71120-1

[3] Wikipedia, `http://ja.wikipedia.org/wiki/`

[4] 数学科教育法 I, `http://www.sm.fukuoka-u.ac.jp/~hideki/kyouiku.htm`

prosper 解説 — 有馬温泉の巻 —

おはらかつよし

平成 17 年 8 月 12 日

prosper は, 講演用のスライドをつくるための LaTeX クラスである. 講演用であるから, まず見栄えに注意が払われている. また, アニメーションもできるし, PStrick を使えば, 絵の中に TeX で書かれた数式を簡単に入れられるのも有難い. 出力形式は pdf であり, 電子文書の世界標準であるから, 講演の現場には, 特別な環境は必要ない. Adobe Reader の載った Windows 機とデータプロジェクタがあればよい. platex を使うことで, 当然日本語の利用もできる.

スライド (example.tex) を pdf ファイルに変換するためには, TeX ソースから dvi ファイル, dvi ファイルから PostScript ファイル, PostScript ファイルから pdf ファイルと三回にわけて変換を行う. TeX から dvi ファイルに変換するにはいつも通りに platex コマンドを利用する. dvi ファイルから PostScript ファイルへは, dvips を, PostScript から pdf ファイルへは ps2pdf を利用する. ps2pdf は Ghostscript に付属するツールである. まとめると, コマンドラインから次のように打てばよい.

```
knoppix@ttyp1[~]$ platex example.tex
knoppix@ttyp1[~]$ dvips example.dvi
knoppix@ttyp1[~]$ ps2pdf example.ps
```

$$\text{example.tex} \overset{\text{platex}}{\longrightarrow} \text{example.dvi} \overset{\text{dvips}}{\longrightarrow} \text{example.ps} \overset{\text{ps2pdf}}{\longrightarrow} \text{example.pdf}$$

Knoppix/Math には, dvi ファイルから pdf ファイルに直接に変換するプログラムとして dvipdfm が収録されているが, prosper を使用している場合には, 非常に残念なことに dvipdfm は使えない. これは prosper の実装方法に起因する問題である.

作成された pdf ファイルは, Knoppix/Math に付属の xpdf で見ることができる. 日本語のフォントがギザギザがあって読みにくいのは, pdf ファイルを生成するときに, ライセンスの関係で, アウトラインフォントがビットマップフォントに変換されて pdf ファイルの中に埋め込まれているからであり, xpdf が原因ではない. よって, Adobe Reader を用いてもギザギザに見える.

debian パッケージ cmap-adobe-japan1, cmap-adobe-japan2 を用いれば解決する (はずだ) が, CD-ROM から起動している場合には, いまのところ対策のとりようがない. どうしても気に入らない場合には, ハードディスクに Knoppix/Math をインストールすることで解決することができる. (この部分, 後で調べる)

prosper の生成する pdf ファイルは各ページの題目をキーとするブックマークが自動的に作成される. xpdf ではブックマークを見ることができないが, Adobe Reader では見ることができる. 講演には Adobe Reader を用いるべきである. 日本語を含む TeX ソースを用いて pdf ファイルを作成する場合には, 日本語のブックマークが文字化けするという問題がある. これはブックマークをつくるためには特別な形式の文字列を Unicode[1] で埋め込まなければならないためである. この問題に対応するには, http://www.rmatsumoto.org/tex-ps-pdf/hyperref.ja.html から http://www.rmatsumoto.org/tex-ps-pdf/convert-euc.txt という perl スクリプトをダウンロードして, tex のソースと同じディレクトリに置く. そして, 次の手順で pdf ファイルを作成すればよい.

```
knoppix@ttyp1[~]$ platex example.tex
knoppix@ttyp1[~]$ dvips -f example.dvi| perl ./convert-euc.txt > example.ps
knoppix@ttyp1[~]$ ps2pdf example.ps
```

prosper を用いたスライドの簡単な実例を示そう.

```
\documentclass[pdf,slideColor,colorBG,darkblue]{prosper}

\title{1 変数多項式の GCD}
\author{Author}
\email{knoppix@example.org}
\institution{Arima Spa}

\begin{document}

\maketitle

\begin{slide}{1 変数多項式の GCD}
\begin{itemize}
\item 1 変数多項式環 ${\mathbf k}[x]$ では割算ができる.
\item {\color{yellow}\bf 割算定理.} $0$ でない任意の
$f,g \in {\mathbf k}[x]$ に対して,
\[
f = {\color{red}q}g + {\color{red}r}, \qquad \deg(r) < \deg(g)
```

[1] 文字コードの国際規格の一つ.

```
\]
を満たす $q, r \in {\mathbf k}[x]$ が存在する.
\item したがって, ${\mathbf Z}$ の場合と同様にユークリッドアルゴリズムに
よって GCD が計算できる.
\end{itemize}
\end{slide}

\end{document}
```

例の最初の行では文書クラスに prosper を指定している. prosper を使う以上, これは必須である. 次に文書クラスに適切なオプションをつける必要がある. この例では, pdf,slideColor,colorBG,darkblue がオプションである. オプションのうち darkblue がスライドのスタイルである. スタイルは主に背景画像や各ページの余白の大きさなどに関係する. Knoppix/Math に含まれているスタイルは次の通り. alcatel, alienglow, autumn, azure, blends, capsules, contemporain, corners, darkblue, default, frames, fyma, gyom, lignesbleues, mancini, nuancegris, prettybox, rico, serpaggi, thomasd, troispoints, whitecross, winter, wj.

スライドの背景に正しく画像を表示するには, 残りのオプションも指定しなければならないので注意すること.

スタイル darkblue の場合, /usr/share/texmf/tex/latex/prosper/PPRdarkblue.sty というファイルが実体であり, 単に, このファイルが参照されているだけである. よって簡単にスタイルが作成できることがわかる. 中にはフリーでスタイルを配布しているサイトもある.

次に通常の TeX ソースと同様に必要ならばプリアンブルで, パッケージを指定する. `maketitle` は 1 ページ目にスライドの表紙を出力する. `maketitle` がない場合は本文からはじまる.

スライドの各ページは, slide 環境として実現される. slide 環境は 1 つの引数をとり, そのページの副題になる. また副題はそのまま, ブックマークとして扱われる. 箇条書きにしたい場合は, itemize 環境が使える.

このソースを変換したものが, 次の図である.

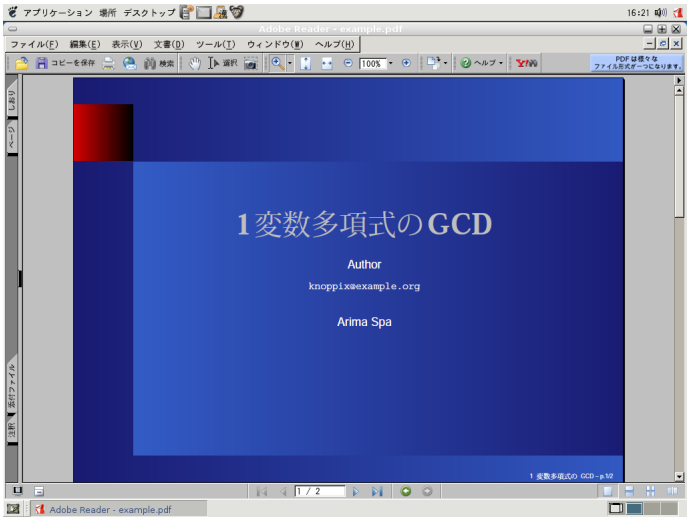


図 1: Example 1. 表紙

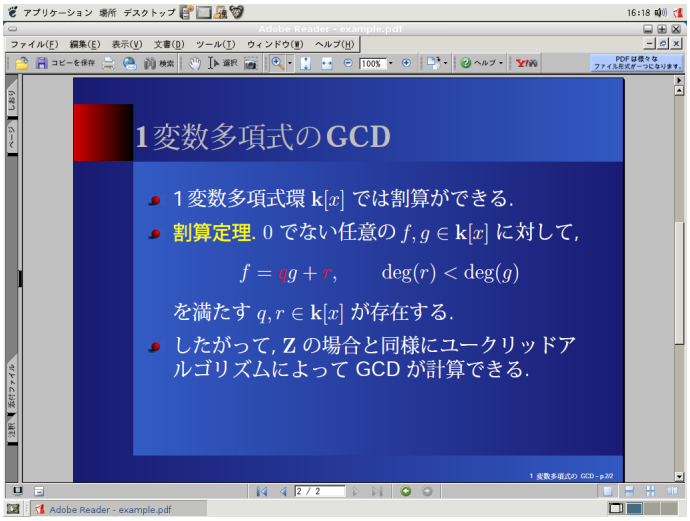


図 2: Example 1. 本文

次に overlays を用いたスライドを作ってみよう.

講演では, 話のすすむまでは, 聴衆にはスライドの一部を隠したいことがある. そのような場合には, prosper では, スライドの overlays という概念で対応している. 例えば,

```
\overlays{3}{
\begin{slide}

\end{slide}
}
```

となっている場合, スライドが 3 段階に変化することを意味する. そのうち

```
\fromSlide*{2}{
...
}
```

で囲まれた部分は 2 段階目以降で現れる部分を意味する. 同様に

```
\onlySlide*{2}{
\ncline[linewidth=2pt]{<-}{Engine}{Kernel}
\Aput{\yellow ReturnPacket[\$Aborted]}
}
```

と書くと, スライドの 2 段階目のみで現れる部分になる.

overlays で便利な箇条書きの形式として itemstep 環境がある. この環境では, 各条がスライドの各段階で一つづつ現れるので, タイプ量が減って便利である.

```
\overlays{5}{
\begin{slide}{割り込みの後始末}
\begin{itemstep}
\item MenuPacket[1,"Interrupt> "] を受け取れば計算が中断されている
\item MLPutString("$\backslash$n")
\item MenuPacket[0,"Interrupt> "] を受け取る
\item MLPutString("a")
\item TextPacket["..."] を受け取る
\end{itemstep}
\end{slide}
}
```

(overlays のサンプル)

(overlays のスクリーンショット)

次にスライドに図を入れることを考えよう. TeX の graphicx パッケージは, dvi ファイルの中に図を埋め込むわけではなく, dviware に special 命令を渡すだけである. 我々は, dvips を使わなければならないから, dvipdfmx のように jpeg や png の図を入れるわけにはいかない. ユーザーがあらかじめ eps 形式に図を変換しておく必要がある. スライドに eps 形式の図を入れたい場合は, 通常の TeX と同様に, `includegraphics` を用いる. そのときに注意しなければならないことは, 何も指定しなければ, 反時計回りに 90 度回転されて図が挿入されていることである. これは, prosper クラスの実装に起因する問題であるので, ユーザが注意して

```
\includegraphics[scale=0.3,angle=-90]{img.eps}
```

のように angle を指定するか, rotatebox を使う必要がある.

また, スライドと連携してアニメーションする図を入れたい場合, 数式を含む絵を入れる場合には, LaTeX の PStricks と連携させることも可能である. 例で示そう.

(PStricks 解説)