KNOPPIX/Math と方程式

高山信毅

2007.05.21

はじめに

著者は方程式を理解することと解くこと, とくに計算機を活用して方程式を理解したり解く事に興味がある. 方程式を解く事への興味にたいして, 数学に興味がある人なら誰でも共感してもらえると思う.

方程式はいろいろなものがあるが, 筆者はいろんな偶然がかさなり, 微分方程式とくに超幾何方程式系を親しく研究することとなった. 超幾何方程式系の研究の仕方にも, 狭く深くから広くなんでもまでいろいろあり, 筆者は “広くなんでも” 型である (付録 A).

超幾何方程式系は数学のさまざまな分野に関係がある. このノートでは超幾何方程式系を解く事, 理解する事, さらには数学者, 数学教育関係者 (情報も), 数学に関係する学生としての毎日の活動に関連した話題をえらび, 主に KNOPPIX/Math に収録されたソフトウエアを紹介をするのを目的とする.

このノート (第 2 部) を執筆するにあたり, どのような形式とするとか随分悩んだが, 一見雑多にみえる話題の集まりも, “計算機を活用して, 方程式を理解することと解くこと” という大目標の下に統一を試みた.

最後にこの本の内容の一部は Free Book “Risa/Asir ドリル 2007” (野呂, 高山) よりの引用である. 引用をこころよく承諾されまた種々のアドバイスをいただいた野呂氏に感謝したい. また講義に出席した学生諸君からはいろいろなフィードバックをもらったり筆者のしらないことを教えてもらった. 筆者の実験的な講義につきあっていただいた, 学生諸君にも感謝したい.

準備

KNOPPIX/Math を利用するにあたり以下の準備は完了しているものとこの本では仮定する. 準備の内容については, 第一部 “KNOPPIX/Math 超入門”, “KNOPPIX/Math 超入門ムービー編” およびフリーブックの “超入門 cfep/asir”, “Asir ドリル” を参照してほしい.

1. KNOPPIX/Math の基本操作ができる.
2. unix シェルの `ls`, `cd`, `cp`, `man` の基本 4 命令をきちんとマスターしている.
3. emacs を使って文書を作成できる. TEX の文書の作成が unix シェルからできる. unix シェルの `platex`, `xdvi`, `dvipdfm` の TEX 基本 3 命令をきちんとマスターしている.
4. 計算機の基本的な作動原理を知っている. 高校数学の教科書程度のプログラム言語の知識に加えてプログラミングの 5 大基本概念変数, 繰り返し, 手続き (関数), 引数, 局所変数を理解している. 10 進ベーシックに習熟していたり, C 言語を本格的に勉強していたり, Risa/Asir ドリルの前半を読破していれば十分である.
5. 拡張子の意味をわかっている. ファイルはバイト列だということを理解している.

個人メモ: この本関連で集めた文献については `@s/2007/02/knx-2007-02`.

Knoppix/Math のデモ.

`xmaxima`

```
plot3d(x^2-y^2,[x,-2,2],[x,2,2]);
factor(x^10-1);
integrate(log(x),x);
```

以上のデモを TeXmacs から試すには, 挿入, セッション メニューを用いる. TeXmacs 上の `asir` も試してみよう.

```
A=newmat(3,3,[[1,1,1],[x,y,z],[x^2,y^2,z^2]]);
poly_factor(matrix_inverse(A));
```

`gnuplot` による Bessel 関数のグラフは次のようになる.

```
plot besj1(x);
```

次は Knoppix/Math の収録のいろんなソフトと JavaView を用いて筆者が研究の成果として作成した, 2 変数ベッセル関数のグラフである.

`http://www.math.kobe-u.ac.jp/HOME/taka/test-bess2m-jv.html`

TEX 関連のソフトも充実している.

Knoppix/Math 収録のいろんなソフトは数学の定理を見つけるための実験装置としても活躍している.

定理 (MMW): $\mathcal{A}$-超幾何方程式系 $H_A(\beta)$ の解空間の次元が任意のパラメータ β にたいして A の正規化体積に等しくなる必要十分条件は 行列 A に付随する affine toric ideal が Cohen-Macaulay であることである.

最近証明されたこの定理に至るまでの道のりでは数多くの数学ソフトが利用されてきている.

目 次

第 1 章　線型代数 **7**
1.1 電卓としての利用, 線型代数 7
1.1.1 行列の計算 7
1.1.2 TEX のソースの生成 10
1.2 教科書の問題を解いてみよう 11
1.2.1 簡約な行列 11
1.2.2 連立方程式を解く 11
1.2.3 簡約な行列の計算と応用 12
1.2.4 Macaulay 14
1.2.5 Singular 14
1.2.6 sgn の計算 16
1.2.7 固有値, 固有ベクトルの計算問題 16
1.2.8 Jordan 標準形の計算 18

第 2 章　微分積分 **23**
2.1 電卓としての利用, 微分積分編 23
2.1.1 微分の計算 23
2.1.2 Taylor 展開の問題 24
2.1.3 合成関数の計算問題 24
2.1.4 不定積分, 微分方程式の記号解の計算 26
2.2 グラフ作成 29
2.2.1 スライドの作成 32
2.3 プログラミング入門 33
2.3.1 流れ図 — flow chart 33
2.3.2 s = s + 1/k の意味は? 34
2.3.3 for ループ 34
2.3.4 練習 34
2.3.5 Debug の仕方 35
2.3.6 他の言語での for と if 35
2.4 数列の計算, maxima, asir 編 37
2.5 2 変数関数の最大値最小値問題 41

第 3 章　微分方程式の近似解法 **45**
3.1 差分法 45
3.1.1 単独方程式の差分化 45
3.1.2 連立方程式の差分化 47

3.1.3 カオス的力学系 . . . 48
3.1.4 2 階の微分方程式の差分化 . . . 49
3.1.5 Risa/Asir で書くと? . . . 52
3.1.6 yorik によるシミュレーション . . . 52
3.1.7 定常状態に対応する代数方程式系 . . . 53
3.1.8 Cavity flow . . . 59
3.2 フラクタルとソリトン: ちょっと寄り道 . . . 59
3.2.1 フラクタルって何? . . . 60
3.2.2 Risa/Asir による Mandelbrot 型集合の描画 . . . 60
3.2.3 Yorick による Mandelbrot 型集合の描画 . . . 61
3.2.4 C と X11 による Mandelbrot 集合の描画 . . . 61
3.2.5 Xaos でみる Mandelbrot 集合 . . . 62
3.2.6 箱と玉の系とソリトン . . . 63
3.3 級数解, Fourier 展開 . . . 64
3.3.1 べき級数解 . . . 64
3.3.2 Maple . . . 64
3.4 Wave ファイル — C 言語による処理 . . . 67
3.4.1 はじめのプログラム . . . 68

第 4 章 グレブナ基底と代数方程式系 75
4.1 1 変数の方程式 . . . 75
4.2 Risa/Asir で書く GCD 計算とその応用 . . . 75
4.2.1 Knoppix での Risa/Asir の使い方 . . . 75
4.2.2 Risa/Asir のプログラミング . . . 77
4.2.3 情報科学からの準備 . . . 78
4.2.4 ユークリッドのアルゴリズム . . . 79
4.2.5 単項イデアルと 1 変数連立代数方程式系の解法 . . . 80
4.2.6 計算効率 . . . 83
4.3 グレブナ基底 . . . 86
4.3.1 initial monomial, initial term など数学的な記号の解説 . . . 86
4.3.2 Initial term の取り出し . . . 86
4.3.3 多項式の内部表現と initial term の取り出しの計算効率 . . . 87
4.4 割算アルゴリズム . . . 90
4.4.1 グレブナ基底 . . . 91
4.5 グレブナ基底と多変数連立代数方程式系の解法 . . . 95
4.6 微分方程式の差分化より得る連立代数方程式の解法例 . . . 98
4.6.1 Lorentz モデル . . . 98
4.6.2 １次元の反応拡散方程式 . . . 99
4.6.3 １次元, 2 成分の反応拡散方程式 . . . 101

第 5 章 グレブナ扇と Tropical Geometry 107
5.1 単独方程式の Tropical zero set . . . 107
5.2 多面体の幾何とグレブナ扇 . . . 109
5.3 tropical geometry . . . 112

5.4 Toric variety . . . 114
5.4.1 Macaulay2 の基本 . . . 114
5.4.2 Macaulay2 を用いて多面体を三角形に分割する . . . 117
5.4.3 グレブナ基底関連システムのコマンド対応表 . . . 120
5.4.4 環の定義 . . . 120
5.4.5 4ti2 . . . 121
5.5 Feasible set の上の乱歩 . . . 122
5.5.1 Toric variety のグレブナ基底の応用 . . . 122
5.5.2 Codon の代数的統計学 . . . 122

第 6 章 代数方程式の数値解法 127
6.1 Newton 法 . . . 127
6.2 ホモトピー法による 1 変数代数方程式の求解 . . . 127
6.3 Numerical Homotopy 法 . . . 130
6.4 SOS による代数方程式の解法 . . . 131
6.4.1 SeDuMi と SOS tools . . . 134

第 7 章 積分とコホモロジ論 137
7.1 超幾何積分の基礎 . . . 137
7.2 D 加群の積分とそのアルゴリズム . . . 137
7.3 対数的コホモロジ群の計算 . . . 138

第 8 章 $\mathcal{A}$ 超幾何方程式系 143
8.1 方程式系の定義 . . . 143
8.2 $\mathcal{A}$ 超幾何方程式系と代数方程式系 . . . 143
8.3 $\mathcal{A}$-超幾何偏微分方程式系 . . . 143
8.4 超幾何微分方程式から導かれた常微分方程式 . . . 144
8.5 級数解の計算アルゴリズム . . . 146
8.6 連立線形偏微分方程式系を数値計算で解くには . . . 146
8.7 modified $\mathcal{A}$-超幾何方程式系についての計算実験 . . . 146

第 9 章 Knoppix/Math について 147
9.1 概観 . . . 147
9.1.1 math-polyglot . . . 147
9.1.2 研究集会対応フォルダについて . . . 147
9.1.3 Free Documents . . . 147
9.2 n 計算機言語対応表 . . . 147
9.3 開発者になる方法 . . . 147
9.4 今後の課題など . . . 147

第 10 章 数学公式集 149
10.1 公式集としての数学ソフト . . . 149
10.2 公式集から数学支援へ . . . 149

付 録 A 超幾何方程式, 関数について **151**
A.1 初等的性質 151
A.2 代数方程式の根と超幾何微分方程式 151
A.3 $\mathcal{A}$-超幾何偏微分方程式系 151
A.4 超幾何微分方程式から導かれた常微分方程式 152

付 録 B あとがき **155**

付 録 C 講義のための補足ノート—神戸大学大学院毎週開講 **157**
C.1 2007-04-13 157
C.1.1 概要と数学の題材 157
C.1.2 実習課題 157
C.1.3 Knoppix/Math の入手方法/実行環境の構築方法 157
C.1.4 Knoppix/Math のブート (起動方法) の仕方 158
C.1.5 その他 158
C.2 2007-04-20 159
C.3 2007-04-27 161
C.3.1 線形代数編 161
C.3.2 課題 161
C.3.3 質問: VMware/Knoppix/Math をインストールしたが動作がおそい 161
C.3.4 質問: `kterm` を MacOS X で起動できませんでした 161
C.3.5 数字の大きさ 162
C.4 2007-05-11 163
C.5 2007-05-18 165
C.6 2007-05-25 167

付 録 D 講義のための補足ノート—東京大学大学院集中講義 **169**
D.1 2007-05-28 169
D.1.1 概要と数学の題材 169
D.1.2 実習課題 169
D.1.3 Knoppix/Math の入手方法/実行環境の構築方法 169
D.1.4 Knoppix/Math のブート (起動方法) の仕方 170
D.1.5 講義のみの資料 (PDF) はパスワード保護されています. 170

第1章　線型代数

この章では KNOPPIX/Math の入門として, 線型代数の計算問題を KNOPPIX/Math を援用してどのように解くか, 作成するか? という問題を考える.

さらにプリントとして配るために TEX ファイルを自動生成する例題も掲載する. KNOPPIX/Math に含まれる種々のシステムのプログラム言語としての側面, 入出力機能も同時に明らかになる.

この本の読者としては線型代数, 微分積分を一度は勉強した人を想定している. 数の計算練習なしに, 電卓をいきなり利用するのが有害であることはすでに定説であると理解している. 数の感覚が全くつかないからである. 線形代数や微分積分も同様で, いきなり数学ソフトで計算をするのは一般にはおすすめできない. はじめは, 紙と鉛筆と工夫で計算すべきである. しかし電卓と同様十分習熟したら積極的にこのようなソフトウエアを利用すべきであろう. しかし経験では別の利用の仕方もある. 紙と鉛筆と工夫で計算するのと併用して利用する方法である. 計算の前に手も足もでない状態のときは, 数学ソフトに計算させてみて, それと比べながら自分でも計算してみて理解を深めていく方法である.

さて残念ながら, 現在の数学ソフトは答えを導出する過程の説明をしてくれない. これは将来の課題である.

1.1　電卓としての利用, 線型代数

ここでは筆者が教科書として利用している “三宅敏恒, 入門線形代数” [5] 等も材料にして KNOPPIX/Math を電卓的につかってみよう.

1.1.1　行列の計算

例 1.1

行列 $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ に対して, A^2, A^{100} を計算しなさい. A の逆行列を計算しなさい.

答は

$$A^2 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix},\ A^{100} = \begin{pmatrix} 218922995834555169026 & 354224848179261915075 \\ 354224848179261915075 & 573147844013817084101 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix}$$

である.

Maxima

Maxima は強力な汎用数式処理システムである. Maxima を使う利点は数字の計算だけでなく, 記号を成分とする行列も扱えることであろう. まずはこれを用いて計算してみよう.

unix シェルより次のように入力して maxima を起動する.

```
maxima
```

Maxima を起動したら,

```
A:matrix([0,1],[1,1]);
A.A;
A^^100;
A^^-1;
```

と入力する. Maxima を終了して unix シェルのプロンプト (入力促進記号) に戻るには ctrl+D または quit(); を入力する. このように maxima を利用するのを, “コマンドラインで利用する” とよぶ. 数学ソフトウエアは unix 生まれのものが多いため “コマンドラインで利用する” のが基本となっている場合が多い. 入力は enter キーをおすまでの行毎に処理される. 行の修正は横方向矢印記号や delete キーで可能である.

コマンドラインから利用する場合は, 入力するコマンドをあらかじめファイルとして用意しておくのも得策である. TEX のソースファイルを準備するのと同じ要領で入力するコマンドをテキストファイル test.txt として用意しておく. maxima に読み込み実行させるには次のように入力すればよい.

```
batch("test.txt");
```

Maxima についての詳しい情報は KNOPPIX/Math に収録のフリー文書として収録されている [2], [3] を参照.

“コマンドラインで利用する” より便利に利用するための工夫がこらされたものが, ノートブックとか GUI インタフェースである. こちらはコマンドラインほどの歴史はないので, システム毎にインタフェースは異なるし, このようなインタフェースを用意してないシステムもある.

Maxima の場合は xmaxima を起動すると GUI インタフェースが起動する. 入力の仕方は同様である. その他の GUI インターフェースとして以下のものがある.

1. texmacs を起動し insert, session, maxima で maxima を起動. enter で計算開始. shift+enter で計算を開始せずに次の行を続けて入力.
2. sage_maxima で maxima を sage-notebook から起動.

これらのインタフェースについては後でインタフェース毎に詳しく説明する.

コマンドラインからの利用法にある程度習熟しておくのは有益である.

例 1.2 $A\begin{pmatrix}1\\2\end{pmatrix}$

答は $\begin{pmatrix}2\\3\end{pmatrix}$.

```
A:matrix([0,1],[1,1]);
B:matrix([1],[2]);
A.B;
```

例 1.3 $\begin{pmatrix} 1 & 1 & 1 \\ x & y & z \\ x^2 & y^2 & z^2 \end{pmatrix}$ の逆行列.

答は

$$\begin{pmatrix} \frac{(z)(y)}{(x-z)(x-y)} & \frac{(-1)(y+z)}{(x-z)(x-y)} & \frac{1}{(x-z)(x-y)} \\ \frac{(-1)(z)(x)}{(y-z)(x-y)} & \frac{(x+z)}{(y-z)(x-y)} & \frac{-1}{(y-z)(x-y)} \\ \frac{(y)(x)}{(y-z)(x-z)} & \frac{(-1)(x+y)}{(y-z)(x-z)} & \frac{1}{(y-z)(x-z)} \end{pmatrix}$$

```
A:matrix([1,1,1],[x,y,z],[x^2,y^2,z^2]);
factor(A^^-1);
```

octave

Octave は MATLAB(商用) に似た数値計算用のシステムである. Octave を使う利点は高速な数値計算と豊富な数値計算ライブラリである.

例 1.1 の問題を octive でやるには次のように入力する.

```
a=[0,1; 1,1];
a*a
a^100
a^(-1)
```

octave では行列の各行を ; で区切る. たとえば $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ を入力するには `[1,2,3 ; 4,5 6]` と入力する.

100 乗の計算の出力をみればわかるように, octave では計算を浮動小数点数でおこなう.

Asir

国産の asir をとりあげよう. Asir を行列計算に使う利点は広く使われている (この例ではでてこないが) C 風の言語が使われている点であろう.

例 1.1 の問題を asir でやるには次のように入力する.

```
A=newmat(2,2,[[0,1],[1,1]]);
A*A;
A^100;
invmat(A);
```

`Asir` では変数名はすべて大文字で始まる. 小文字で始まる変数は多項式の不定元として扱われる.

行列は関数 `newmat` で生成する. たとえば $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ を入力するには `newmat(2,3,[[1,2,3],[4,5,6]]);` と入力する.

参考: asir は texmacs から利用するのも可能である. いままでの asir のそっけない出力しかない人には新鮮でしょう. (insert, session, OpenXM を選択して asir を起動)

Mathematica と Maple

Mathematica と Maple は有名な商用の数学ソフトである. 両者とも数式処理システムを起源とするが, 現在ではさまざまな数学アルゴリズムを組み込んだ総合的システムである.

フリーソフト中心の Knoppix/math には勿論含まれていないが, 比較のためにこれらのソフトも取り上げておこう.

Mathematica の場合.

```
a={{0,1},{1,1}}
a . a
MatrixPower[a,100]
MatrixPower[a,-1]
```

Mathematica では行列は記号 {, } で生成する. たとえば $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ を入力するには {{1,2,3},{4,5,6}} と入力する.

Maple の場合.

```
with(linalg);
a:=matrix( [[0,1],[1,1]] );
evalm(a &* a);
evalm(a^100);
inverse(a);
```

Maple では関数 matrix か array で行列を生成する.

evalm 関数 (eval matrix の略) では, 行列を含んだ表現を評価する.

&* は非可換な積をあらわす記号である. 多くの項を掛ける場合 Maple が行列計算を始める前に式を簡単化する可能性があるので* よりも, この記号を使う方が安全である.

xmaple は maple の GUI 版であり, 各種の便利な機能がある.

その他

math-polyglot/src/matrix-power を参照. 大規模な線型代数計算には C, C++ のライブラリである Linbox や Bras 等を使うといいかも.

1.1.2 TEX のソースの生成

ちなみにこの章の TEX のソースは Mathematica で

```
a={{1,1,1},{x,y,z},{x^2,y^2,z^2}}
 TeXForm[Factor[Inverse[a]]]
```

をもとに作成したり, asir で

```
A=newmat(3,3,[[1,1,1],[x,y,z],[x^2,y^2,z^2]]);
print_tex_form(poly_factor(matrix_inverse(A)));
```

として作成.

1.2 教科書の問題を解いてみよう

1.2.1 簡約な行列

簡約な行列と基本変形については [5, p.23] を参照してほしい. 簡約な行列を多変数の 1 次式の集まりとみなす場合もある. たとえば

$$\begin{pmatrix} 0 & \underline{1} & 0 & 0 & -5 & 3 \\ 0 & 0 & 0 & \underline{1} & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

は

$$\underline{x_2} - 5x_5 + 3x_6, \quad \underline{x_4} + x_6$$

とみなす.

行をみたとき, 初めての 0 でない要素をその行の主成分という. 0 しか要素にない行には主成分がない. Grobner 基底の理論では対応する 1 次式の主成分に対応する monomial を Leading monomial といったり, initial monomial という.

簡約な行列の定義.

1. 0 しか要素にない行は一番下にあつめる.
2. 行ベクトルの主成分は 1.
3. 各行の主成分は下へ行くほど右にある.
4. 各行の主成分の上と下の成分はすべて 0.

具体例でみるとわかりやすい.

行列は基本変形で簡約な行列にできる.

1.2.2 連立方程式を解く

例 1.4 [5, 例題 2.3.2] を解いてみよう.

問題

$$A = \begin{pmatrix} 1 & -2 & 0 & 3 & 0 \\ 1 & -2 & 1 & 2 & 1 \\ 2 & -4 & 1 & 5 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 2 \\ 5 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$$

と置くとき, $Ax = b$ の解 x を計算せよ.

答は c_1, c_2 を任意の数として,

$$x_1 = 2 + 2c_1 - 3c_2, x_2 = c_1, x_3 = -1 + c_2, x_4 = c_2$$

このような問題を解くプログラムを書くこと, つまり Gauss 消去アルゴリズムの実装はプログラミングのよい練習問題である. この本で取り上げているソフトウエアにはこのアルゴリズムがすでに実装されている. Gauss 消去アルゴリズムの実装も一度は自分で試みてから, このようなソフトを用いるのが肝要であろう. これは計算練習を沢山した人が電卓を使いこなせる, ということと同じである.

Maxima

```
eq1:[x1-2*x2+3*x4-2,x1-2*x2+x3+2*x4+x5-2,
     2*x1-4*x2+x3+5*x4+2*x5-5];
s:solve(eq1,[x1,x2,x3,x4,x5]);
/* Call, for example, subst(3,%r1,s)
   to replace %r1 by 3 in s */
```

solve 関数を用いて連立方程式を解ける. 結果にあらわれる% で始まる変数は任意定数である. 任意定数を数字等で置き換えたいときは, subst 関数を用いる.
関数 linsolv を用いると線型方程式系をより高速に解く.

出力結果

```
Maxima restarted.
(%i1)
(%o1) [3 x4 - 2 x2 + x1 - 2, x5 + 2 x4 + x3 - 2 x2 + x1 - 2,
    2 x5 + 5 x4 + x3 - 4 x2 + 2 x1 - 5]
(%i2)
(%o2) [[x1 = - 3 %r2 + 2 %r1 + 2, x2 = %r1, x3 = %r2 - 1, x4 = %r2, x5 = 1]]
(%i3)
```

Asir

```
load("gr")$
A=newmat(3,5,
  [[1,-2,0,3,0],
   [1,-2,1,2,1],
   [2,-4,1,5,2]]);
X=newvect(5,[x1,x2,x3,x4,x5]);
B=newvect(3,[2,2,5]);
E=A*X-B;
gr(vtol(E),vtol(X),2);
end$
```

gr は簡約グレブナ基底を計算する関数であるが, 線型式系に適用した場合は, [5] の “簡約な行列” (p.23, 2.2 簡約な行列) にほかならない.

出力結果

```
[-x5+1,-x4+x3+1,-3*x4+2*x2-x1+2]
```

x1, ..., x5 の変数順序なので, この変数順序で一番大きい変数がその他の変数で書ける. -x4+x3+1 では x4 を任意定数として, x3 を書く. -3*x4+2*x2-x1+2 では x4, x2 を任意定数にとり, x1 を表す.

1.2.3　簡約な行列の計算と応用

簡約な行列 [5, p.23] は rank の計算, 連立方程式の解法, Ker, Im の計算などさまざまな応用がある.

例 1.5 ([5, p.25] より) 次の行列を簡約な行列に変形せよ.

$$A = \begin{pmatrix} 0 & 0 & 0 & 2 & 3 & 2 \\ 0 & 3 & 6 & -9 & -4 & 7 \\ 0 & 2 & 4 & -6 & -4 & 2 \end{pmatrix}$$

答:

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

簡約化のプログラムを書くのは, よい演習問題であるが, 辞書式順序についてのグレブナ基底計算アルゴリズムを線型方程式に適用すると, 簡約化の計算ができる. グレブナ基底については Chapter 4 で詳細に説明するが, ここではグレブナ基底計算への翻訳方法と結果の読み取り方を説明しよう.

問題 1.1 例 1.4 を簡約な行列への変形機能 (Gröbner 基底の計算機能) を用いて解きなさい.

Maxima

ノート: maxima では affine パッケージに グレブナ基底の計算関数が用意されているが, 手元の knoppix/math では `load("affine");` が失敗する.

しかしながら `echelon` なる関数が用意されており, 引数として行列を与えると簡約に近い形の行列を戻す.

Asir

入力

```
A=newmat(3,6,
 [[0,0,0,2,3,2],
  [0,3,6,-9,-4,7],
  [0,2,4,-6,-4,2]]);
G=A*newvect(6,[x1,x2,x3,x4,x5,x6]);
G=vtol(G);
gr(G,[x1,x2,x3,x4,x5,x6],2);
```

出力

```
[ 0 0 0 2 3 2 ]
[ 0 3 6 -9 -4 7 ]
[ 0 2 4 -6 -4 2 ]

[2*x6+3*x5+2*x4,7*x6-4*x5-9*x4+6*x3+3*x2,
 2*x6-4*x5-6*x4+4*x3+2*x2]

[2*x6+x5,-2*x6+x4,-x6+2*x3+x2]
```

出力の最後の式が簡約グレブナ基底である. ただし, LM (後述) の係数は 1 に正規化してない. 簡約グレブナ基底が, 答の簡約行列に対応していることを確かめよ. 簡約グレブナ基底には 0 のみからなる行に対応する式はないので, 簡約グレブナ基底の個数, この例の場合では 3 が与えられた行列の rank である. [5, p.26] をみよ.

各式で `x1`, ..., `x6` の順番で一番始めに出てくる項を leading monomial (LM) と呼ぶ. 結果の `2*x6+x5` の LM は `x5` である. `-2*x6+x4` の LM は `x4` である. `-x6+2*x3+x2` の LM は `x2` である.

Asir と共に Gröbner basis 計算で定評のある Macaulay とか Singular でも同じ計算をやってみよう. Singular や Macaulay は, 可換環論, 代数幾何用の専門家システムでもある. コマンドに登場するイデアル等の用語については後の方の章でくわしく説明する.

1.2.4 Macaulay

Macaulay を起動するには 3 つ方法がある.

1. unix シェルより `M2` と入力.
2. unix シェルより `emacs -e M2` と入力. `emacs` の中で Singular を利用できて便利である.
3. KDE の $\sqrt{x}$ メニューより起動.

```
QQ[a,b,c,d,e,f,MonomialOrder=>Lex]
I=ideal(2*d+3*e+2*f, 3*b+6*c-9*d-4*e+7*f, 2*b+4*c-6*d-4*e+2*f)
gens gb I
```

M2 は完全に簡約な形を戻す.

1.2.5 Singular

Singular を起動するには 3 つ方法がある.

1. unix シェルより `Singular` と入力.
2. unix シェルより `ESingular` と入力. `emacs` の中で Singular を利用できて便利である.
3. KDE の $\sqrt{x}$ メニューより起動.

```
ring R=0,(a,b,c,d,e,f),lp;
ideal I=2d+3e+2f, 3b+6c-9d-4e+7f, 2b+4c-6d-4e+2f;
ideal J=std(I);
J;
```

注意: Singular の std コマンドは完全に簡約な形を戻すわけではない. 主係数は 1 とは限らない. 主係数の上下が 0 が 0 になっているとは限らない. 完全に簡約な形を計算するにはプログラムを書く必要がある.

例 1.6 [5, p.76]
次の列ベクトルの一次独立な最大個数 r と r 個の一次独立なベクトルを一組求め, 次に他のベクトルをこれらの一次結合で表せ.

$$a_1 = \begin{pmatrix} 1 \\ 1 \\ 3 \\ 0 \end{pmatrix}, \quad a_2 = \begin{pmatrix} 1 \\ 2 \\ 0 \\ -1 \end{pmatrix}, \quad a_3 = \begin{pmatrix} 1 \\ 3 \\ -3 \\ -2 \end{pmatrix}$$

$$a_4 = \begin{pmatrix} -2 \\ -4 \\ 1 \\ -1 \end{pmatrix}, \quad a_5 = \begin{pmatrix} -1 \\ -4 \\ 7 \\ 0 \end{pmatrix}$$

答. 上の縦ベクトル $a_1, \ldots, a_5$ をならべてつくった行列を A とおく. x を長さ 5 の未知数のベクトルとするとき, B を A から基本変形で導出された行列とするとき, x が $Ax = 0$ をみたすことと

$Bx = 0$ をみたすことは同値である. つまり B の列ベクトル達の線型関係式は A の列ベクトル達の線型関係式であり, 逆もまたなりたつ.

B としてとくに簡約な行列をとる.

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

B の形から b_1, b_2, b_4 が一次独立となることがわかる. 上の同値性より a_1, a_2, a_4 が一次独立となることがわかる. B より b_3, b_5 つまり a_3, a_5 は $-a_5 - a_4 = 0, , -a_5 + 2a_3 + a_2 = 0, -2a_5 + a_3 - a_1 = 0$ と a_1, a_2, a_4 を用いて書ける.

Asir

入力

```
def hm(F) {
  V=[x1,x2,x3,x4,x5];
  F2=dp_ptod(F,V);
  return dp_dtop(dp_hm(F2),V);
}
A=newmat(4,5,
 [[1,1,1,-2,-1],
  [1,2,3,-4,-4],
  [3,0,-3,1,7],
  [0,-1,-2,-1,0]]);
G=A*newvect(5,[x1,x2,x3,x4,x5]);
G=vtol(G);
Gb=gr(G,[x1,x2,x3,x4,x5],2);
map(hm,Gb);

end$
```

出力

```
[-x5-x4,-x5+2*x3+x2,-2*x5+x3-x1]
[-x4,x2,-x1]
```

簡約グレブナ基底の個数が一次独立なベクトルの個数である. この例では $r = 3$. 一次独立なものは leading monomial に対応するベクトルである.

a_3, a_5 は $-a_5 - a_4 = 0, , -a_5 + 2a_3 + a_2 = 0, -2a_5 + a_3 - a_1 = 0$ と a_1, a_2, a_4 を用いて書ける. この関係式は簡約グレブナ基底の元にほかならない.

他のシステムについては上の例題と同様なので省略する.

例 1.7 [5, p.90] より

$$T = \begin{pmatrix} 2 & -1 & 1 & 5 & 0 \\ 1 & 3 & 4 & -1 & 7 \\ 1 & 0 & 1 & 2 & 1 \end{pmatrix}$$

とおくとき, $\mathrm{Ker}\,(T)$ および $\mathrm{Im}\,(T)$ の基底を求めよ.

T を簡約した行列を B とすると, $Bx = 0$ と $Tx = 0$ は同値である. したがって T のたてベクトルの間の関係式　(syzygy) は, B のたてベクトルの間の関係式　(syzygy) とおなじ. (ひとつ前の例題と同じである.) よって, 簡約グレブナ基底の LM に対応するベクトル達が $\mathrm{Im}\,(T)$ の基底である.

$\mathrm{Ker}\,(T)$ (T のカーネル) の計算は連立方程式 $Tx = 0$ の一般解, つまり $Bx = 0$ を解く事にほかならない.

プログラム例はまだ.

問題 1.2　1. 長谷川, 線形代数, p.36 [4] より. $\begin{pmatrix} 1 & t \\ 0 & 1 \end{pmatrix}$ による, 正方形の像を描画するプログラムを書きなさい.

2. JavaView をインストールし, 平行四面体を描画しなさい. 平行四面体は立方体の非退化な行列による像である. [4, p.111] 参考プログラムは　`taka_jv.rr` ,　`povray.rr`　であるが, 詳細は **??** 節で扱う.

1.2.6　sgn の計算

例 1.8 [5, p.41] 次の置換　σ を互換の積に分解して符号を求めよ.

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 6 & 8 & 2 & 1 & 4 & 9 & 3 & 5 \end{pmatrix}$$

σ は次の互換の積に分解することが, たとえば 1 を出発点とする軌道の計算, 2 を出発点とする軌道の計算, 3 を出発点とする軌道の計算からわかる.

$$\sigma = (3,8)(2,6,4)(1,7,9,5)$$

Gap

GAP は一般的な数学システムであるがとくに群論に強い. Manual p.404 をみてみよう. `SignPerm` は置換の符号を計算する関数である.

```
SignPerm((3,8)(2,6,4)(1,7,9,5))
```

1.2.7　固有値, 固有ベクトルの計算問題

固有値の計算の応用問題として次の問題を考えてみよう.

Mathematica

商用ソフトなので Knoppix/math には含まれていないが, Mathematica で問題の計算をやってみよう. 入力ファイル `test.txt` は以下のとおり.

```
m=Table[0,{i,1,3},{j,1,3}];
For[i=1,i<=3,i++,
  For [j=1, j<=3, j++,
    m[[i,j]] = Random[Integer,{0,99}];
  ]
];

Print[m];

a=Transpose[m]*m;
Print[Eigenvalues[a]];
Print[N[Eigenvalues[a]]];
Print[Eigenvectors[a]];
Print[N[Eigenvectors[a]]];
```

出力は次のようになる. 長い式は省略している.

```
In[1]:= <<test.txt
{{24, 44, 95}, {97, 86, 78}, {95, 4, 82}}
 14696                          341672323
{----- + ------------------------------------------------------- +
   3                                                          1/3
         3 (-3130752399922 + 10557 I Sqrt[269942943790185567])

                                                             1/3
     (-3130752399922 + 10557 I Sqrt[269942943790185567])
>    ---------------------------------------------------,
                                3

--- Other eigenvalues are snipped.

--- Eigenvectors are snipped.

                         -16                           -17
{{0.826089 + 1.00424 10    I, 0.551749 + 6.60821 10    I, 1.},

                           -16                           -17
>   {-1.49952 - 4.03463 10    I, 0.432685 + 1.05105 10    I, 1.},

                          -15                           -13
>   {0.100494 + 4.7727 10    I, -1.96288 - 1.05991 10    I, 1.}}
```

なお Mathematica の linux 版を購入するとハードディスクにインストールした knoppix/math (Debian GNU linux にほかならない) の上で Mathematica が動作する.

注意: 実対称行列の固有値はすべて実数である. ($Ax = cx$ (c は固有値) とおく. $A\bar{x} = \bar{c}\bar{x}$ なので, $\bar{c}\bar{x}^T x = (A\bar{x})^T x = \bar{x}^T Ax = c\bar{x}^T x$ より $\bar{c} = c$ である.) しかしながら上の答えは虚数部分を含む. このように代数的計算は厳密であるが, (1) 簡単化は一意的でない (2) 近似数値計算で誤差が入り込む. この二つが虚数部分を含んだ解を戻す原因である. 数式処理システムの答えといえども無条件に信用するのは禁物である.

なお, 出力のはじめの部分は (多分 Cardano の公式による) 厳密解であるが, これも imaginary part をもつ解にみえる. しかし変形することにより, 実数となる. この部分の事情については, 高木貞治 代数学講義 [1] の第 6 章 3 次および 4 次方程式に解説があるので参照してほしい.

R

R は統計解析のための数学ソフトウエアである.

```
m<-array(c(1,2,3,4,5));
barplot(m);
```

R での変数への代入コマンドは <- である. R においては各種の array (配列) を生成, 操作する. この例は 1 次元の配列 [1,2,3,4,5] を生成する例である. `barplot` は棒グラフを作成する.

```
m<-array(0,c(3,3));
for (i in 1:3) {
  for (j in 1:3) {
    m[i,j] <- floor(runif(1)*100);
  }
}
print(m);
# unif, norm, hyper, ...
# m<-array(runif(9),c(3,3))

a<-aperm(m,c(2,1)) %*% m;
# Computing eigenvalues
#  and eigenvectors of a
ev<-eigen(a);
print(ev$values);
print(ev$vec);

q();
```

R においては与えられた分布にしたがい乱数を発生させる関数が各種用意されている. unif は一様乱数を生成する関数である. この関数は区間 $[0, 1]$ の乱数を発生するので, 100 倍して切り捨てをすることにより, 0 から 100 の間の整数が生成される.

あらかじめテキストエディタで入力したコマンドは R < ソースファイル名　で実行させてもよいが, なお R 内部からのファイルの読み込み実行コマンドは source である.

さて少々高度な話題をとりあげよう

下のプログラムは, 要素が独立, 平均 0, 分散が 1/2 の場合の Winger 半円則 [7, p.42]) が成り立つことを計算機実験するプログラムである.

```
size<-100;
m<-array(0,c(size,size));
for (i in 1:size) {
  for (j in i:size) {
    m[i,j] <- rnorm(mean=0.0, sd=0.5, n=1);
    m[j,i] <- m[i,j];
  }
}
print(m);

# Computing eigenvalues
ev<-eigen(m);
print(ev$values);
e<-ev$values;
hist(e);

#google: グ ラ フィ ック ス 参 考 実 例
集 www.okada.jp.org/Rwiki
```

行列の成分がガウス分布にしたがう実対称行列を考える. その固有値の頻度のグラフをつくるとほぼ円形となるのが, 有名な Winger 則であり, random matrix の理論で有名な定理である. これを R で実験してみよう. rnorm は正規分布 (ガウス分布）にしたがって乱数を生成する関数である.

問題 1.3 筆者は Winger 半円則の証明を知らない. 関連の研究も知らない. でも計算機で実験してみて興味がわいてきた. 参考文献等を読み, わかりやすい解説, 証明等をレポートとして提出していただけるとうれしい.

方程式系を固有値問題に帰着して解くアルゴリズムはグレブナ基底のよく知られた応用である. 詳しくはグレブナ基底の章を.

1.2.8　Jordan 標準形の計算

Asir

代数拡大もやりながら, Jordan 標準形を計算する.

http://www.math.kobe-u.ac.jp/HOME/taka/2007/knx/noro-matrix_ja.txt

関連図書

[1] 高木貞治, 代数学講義

[2] 中川義行, Maxima 入門ノート 1.2.1, 2006.
KNOPPIX/Math, 2006,
Desktop/knoppix-math-ja/PDF/maxima-note.pdf

[3] Maxima マニュアル改訂版, 2006.
KNOPPIX/Math, 2006,
Desktop/knoppix-math-ja/PDF/MaximaManualBeta.pdf

[4] 長谷川浩司, 線形代数, 日本評論社, 2004.

[5] 三宅敏恒, 入門線形代数, 培風館, 1991.

[6] 横田 博史, はじめての Maxima, 2006, 工学社. Maxima の使いかたのみならず, maxima の計算原理も解説している本. Knoppix/math の話題も豊富.

[7] 長尾太郎, ランダム行列の基礎, 東京大学出版会, 2005

第2章　微分積分

この章では KNOPPIX/Math の入門として, 微分積分の計算問題を KNOPPIX/Math を援用してどのように解くか, という問題を考える. Maxima を主に利用するが, 適宜他のシステムへの入力も紹介する.

序文で, 繰り返しなどのプログラミングの基礎をマスターしていることがこの本を読む予備知識と述べたが復習もかねて, プログラミングの基礎も復習する.

2.1　電卓としての利用, 微分積分編

数学関連事項はテキスト以外に板書でも説明. ビデオ: `lecture-2007/knx-05-10`

2.1.1　微分の計算

例 2.1 [4, p.58] $\sin x$ の n 階の微分を計算しなさい.

一般の n での計算は公式集等が組み込まれたシステムでないとできない. n に具体的な数字をいれた場合の計算は簡単なので, いろいろな n で計算してみて結果を予想する.

$\sin(x)$ の 3 階の微分の計算.

システム名	入力
maxima	`diff(sin(x),x,3);`
asir	`diff(sin(x),[x,x,x]);`
axiom	`D(sin(x),[x],[3])`
maple	`diff(sin(x),x$3);`
Mathematica	`D[Sin[x],{x,3}]`

例 2.2 $\tan^{-1}(x)$ の n 階の微分の $x=0$ での値 c_n を求めなさい.

$\tan^{-1}(x)$ の 3 階の微分の $x=0$ での値.

システム名	入力
maxima	`subst(0,x,diff(atan(x),x,3));`
asir	`subst(diff(atan(x),[x,x,x]),x,0);`
axiom	`subst(D(atan(x),x,3),[x=3])`
maple	`subs(x=0, diff(arctan(x),x$3));`
Mathematica	`D[ArcTan[x],{x,3}] /. {x->0}`

一般の n での計算はむつかしい. n に具体的な数字をいれた場合の計算は簡単なので, いろいろな n で計算してみて結果を予想する. 階乗に関係する数が出現すると思う.

この問題の解は

$$c_n = (-1)^{(n-1)/2}(n-1)!\ (n \text{ が奇数}), c_n = 0\ (n \text{ が偶数})$$

である. この結果は $\tan^{-1}(x)$ のテイラー展開を求めるのに必要である. 証明は少々工夫を必要とする. 微分積分の教科書や演習書をみてみよう.

ちなみにいろんなシステムでの書き方を比較してある rosetta.pdf は便利. `http://wiki.axiom-developer.org` よりリンクあり.

2.1.2 Taylor 展開の問題

例 2.3 $\tan^{-1}(x)$ の原点での Taylor 展開を計算しなさい.

Maxima

```
ts:taylor(atan(x),x,0,10);
float(4*subst(1,x,ts));
```

2 行目は Taylor 展開を x^{10} の項まで計算する. 2 行めはその結果を用いて $4 * \tan^{-1}(1)$ の近似計算をしてしている. $4 * \tan^{-1}(1) = \pi$ なのであるが, あまり近似はよくない. Taylor 展開の次数をもっとふやして, たとえ 100 にしても 3.14 には程遠い. Taylor 展開の収束は原点近傍では早いが (どういう意味?), 外ではおそい. 一方マチンの公式は原点近くの $\tan^{-1}$ の展開を用いる (wikipedia).

システム開発者向けコメント.
上で `ts` の Lisp での内部構造を見るには `:lisp $ts;` と入力する. マニュアルの Taylor 展開の章を参照.

2.1.3 合成関数の計算問題

合成関数は上で紹介した置換の関数や, 関数のすなおな合成をやればよい. ここでは応用として次の問題を考えよう.

$D = \{x \,|\, 0 \le x \le 1\} = [0, 1]$ に対して, 写像 f を

$$f(x) = \{\, 2\, x, \quad 0 \le x \le \frac{1}{2} 2 - 2x, \quad \frac{1}{2} \le x \le 1$$

で定義する. f をテント写像とよぶ.

高木関数 $T(x)$ は

$$T(x) = \sum_{n=1}^{\infty} \frac{f_n(x)}{2^n}$$

で定義される. ここで $f_n(x)$ は f (テント写像) を n 階合成した関数である.

例 2.4 テント写像の合成関数の値, および, そのグラフを書きなさい.

Maxima

テント関数を maxima で定義するには

```
f(x):= if x < 0.5 then 2*x else 2-2*x;
```

とすればよい.

$f(x)$ の合成関数は $f(f(x))$, $f(f(f(x)))$ などと計算していけばよい. 値を求めるには, たとえば `f(f(f(0.1)));`

Maxima には `plot2d` というグラフを描く関数がある. たとえば

```
plot2d(sin(x),[x,0,10])
```

とすると sin のグラフが書ける. `plot2d` 関数は残念ながらこのような $f(x)$ は受け付けてくれないようである. もっと長いプログラムを書いてみないといけないようだ. うまい方法があるかどうかは講義中の課題.

問題 2.1 高木関数のグラフを書きなさい.

報告課題: 高木関数は連続だが微分可能な点は存在しないことが知られている. 証明しなさい.

Mathematica

Mathematica は見事につぎのようなグラフを書く.

```
f[x_]=If[x<0.5,2*x,2-2*x];
Plot[f[f[x]],{x,0,1}]
Plot[f[f[f[x]]],{x,0,1}]
```

asir

(このプログラムは少々上級. あとまわし.)

```
def tent(X) {
  if (X < 1/2) return 2*X;
  else return 2-2*X;
}

def t2() {
  gnuplot.plotDots(0,0);
  A=[];
  for (X=0.0; X<=1.0; X += 0.02) {
     A=cons([X,tent(tent(X))],A);
  }
  A = reverse(A);
  print(A);
  gnuplot.plotDots(A,"lines");
}

def tn(N) {
  gnuplot.plotDots(0,0);
  A=[];
  for (X=0.0; X<=1.0; X += 0.01) {
    Y = tent(X);
    for (I=0; I<N-1; I++) {
      Y = tent(Y);
    }
    A=cons([X,Y],A);
  }
  A = reverse(A);
  print(A);
  gnuplot.plotDots(A,"lines");
}

end$
```

tn(5) あたりはうそのグラフになってる．このような方法の限界である．論理的に関数を合成しないといけない．

2.1.4　不定積分, 微分方程式の記号解の計算

不定積分の計算はいわゆる Risch のアルゴリズムが基礎となっている．Maxima は 1970 年代に Risch アルゴリズムの研究が全盛であった時代に開発されたため不定積分の能力は充実している [7, 94–96]. 最新の研究については M.Bronstein による [6] を読むことをお薦めする．Bronstein の最新の研究成果は axiom およびその仲間の aldor, Σ^{it} などに実装されている．

WikiPedia の Risch algorithm `http://en.wikipedia.org/wiki/Risch_algorithm` の項から Bronstein による symbolic integration tutorial がリンクされている．

コラム: 不定積分は現代数学にとって興味深い課題か? やっぱり微分ガロア理論とのつながりで興味がうまれるのか?

不定積分

とりあえず maxima でいろんな不定積分をためしてみよう．

```
assume(not equal(n,-1));
integrate(x^n,x);
```

```
integrate(1/sqrt(1-x^2),x);
integrate(1/(1+x^3),x);
integrate(1/sqrt((1-x)*(1+x*z)),x);
/* restart. z+1 nonzero; z positive;  z-1 nonzero; */
```

nonzero とか質問されたら, nonzero; などと入力する. セミコロンと ENTER キーをおすのをお忘れなく.

専門家向け: $\sin^{-1}(x)$ は超幾何積分の簡単な場合とも思える...

教科書の問題をかたっぱしからやってみるのも楽しいかも. ただしやりかたを説明してくれないのが残念. [4] の p.101 演習問題 4, 6 を試す.

```
integrate((x+1)/(x*(x-8)^(1/3)),x);
integrate((1/x)*sqrt((x+4)/(1-x)),x);
integrate(1/(x*sqrt(x^2+x+1)),x);
integrate(1/(x*sqrt(2+x-x^2)),x);
integrate(sin(x)^4*cos(x)^3,x);
integrate(1/cos(x),x);
integrate(cos(x)/(sin(x)+cos(x)),x);
integrate(6/(4*cos(x)^2+sin(x)^2),x);
```

Axiom への入力方法は maxima とほぼ同様である. 終了は)quit と入力してから y を入力する. ファイルの読み込みは)read ファイル名 である. ; を入力すると結果は表示されない.

```
integrate(1/sqrt((1-x)*(1+x*z)),x)

integrate((x+1)/(x*(x-8)^(1/3)),x)
integrate((1/x)*sqrt((x+4)/(1-x)),x)
integrate(1/(x*sqrt(x^2+x+1)),x)
integrate(1/(x*sqrt(2+x-x^2)),x)
integrate(sin(x)^4*cos(x)^3,x)
integrate(1/cos(x),x)
integrate(cos(x)/(sin(x)+cos(x)),x)
integrate(6/(4*cos(x)^2+sin(x)^2),x)
```

微分方程式

微分方程式についてはまた次の節で詳説するが, とりあえず maxima による記号解法をためしてみよう. 利用しているアルゴリズムについては ode 部分の開発にも参加した渡辺の本 [8] が貴重な文献である. これは 1970 年代の最新の成果である. 特殊関数でとければグラフもかける.

なお, あとの章でふれる級数解については Maple のパッケージ DEtools[formal_sol] (Mark Von Hoeij が開発した) が充実している. ([9] も参照. この本に書かれているプログラム群は配布可能なパッケージとして完成していないのが残念である.)

なお 微分ガロア群の計算, 有理解の計算, 代数解の計算, Liouvillian 解 (雑にいえば初等関数と $\int$ と exp(x) で書ける解) の計算アルゴリズムの研究は macsyma が開発されていた, 1970 以来研究が

おおいに進展しており, 成果は axiom(むしろ Σ^{it}, Maple 等のパッケージに反映されている (ようだ).

強制振動の解

```
atvalue(x(t),t=0,1);
atvalue(diff(x(t),t),t=0,0);
dsol:desolve(diff(x(t),t,2)+x(t) = sin(t),x(t));

/* We need to get the righthand side of =
   or use "part" function (p.118) */
sol2a:substpart("+",dsol,0);
sol2b:subst(0,x(t),sol2a);
plot2d(sol2b,[t,0,30]);

sol:ode2('diff(x,t,2)+x = sin(t),x,t);
sol2:subst(3,%k2,subst(2,%k1,sol));
```

Duffing 方程式の解. (強制振動の解. 外力ありだとカオスでもでてくる.)

```
eq:'diff(x,t,2)+x+e*x^3=0;
ode2(eq,x,t);
```

Logistic equation

```
ode2('diff(x,t)-x*(1-x),x,t)=0;
%% It answers in an implicit form.
```

本来の Logistic 方程式には二つのパラメータ k と A ある.

$$x' = kx(A-x), \quad x(0) = x_0$$

$$x(t) = \frac{A}{1 + (A/x_0 - 1)\exp(-kAt)}$$

が一般解である [11, p.26].

問題 [11] の 第 3 話にでている成長曲線にあらわれるようなデータを R の関数で解析しなさい.

連立の微分方程式.

```
atvalue(x(t),t=0,1);
atvalue(y(t),t=0,0);
sol:desolve([diff(x(t),t)= -(1/2)*x(t)+(1/100)*y(t),
             diff(y(t),t)=  (1/200)*x(t)-(1/3)*y(t)], [x(t),y(t)]);


plot2d([parametric,part(sol[1],2),part(sol[2],2)],[t,0,10]);
```

簡単な reaction-diffusion 方程式の定常解だが, 出力される解は陰形式でかつ, 超幾何不定積分. (todo: これで OK?)

```
eq:(1/200)*'diff(x,t,2)+x*(1-x)=0;
ode2(eq,x,t);
```

Maple

```
#  p.312 (Waterloo Maple Inc)
# in Computer algebra handbook, Springer.
# read ‘maple-de.txt‘;
with(DEtools):
eqn:=diff(y(x),x)=-y(x)-x^2;
dsolve(eqn,y(x));

dsolve({eqn,y(0)=0},y(x));

phaseportrait(eqn,y(x), x=-1..2.5,
  [[y(0)=0],[y(0)=1],[y(0)=-1]],
  title=‘Asymptotic solution‘, colour=magenta,
  linecolor=[red,blue,green]);
```

微分方程式と付随する特殊関数

Bessel 関数や Airy 関数などの特殊関数も maxima には組み込まれている. Airy 関数のグラフ.

```
plot2d(airy(x),[x,-10,10]);
```

$$Ai(0) = 3^{1/6}\Gamma(1/3)/(2\pi)$$

をたしかめて見る. 値があうとうれしい...

Airy の微分方程式, Airy 関数の積分表示については `http://functions.wolfram.com` も参照してみよう.

数学関連事項はテキスト以外に板書でも説明. ビデオ: `lecture/knx-05-11`

問題: Airy 関数のグラフを複素数引数でかけるか?

問題: Bessel 関数を用いた円の Laplacian の固有関数の表示.

問題: 特殊関数を用いた Schrödinger 方程式の特殊解.

研究問題: 虹と Airy 関数.

研究問題: Airy 関数の一般化としての $[1, 2, 3, \ldots, n]$ 超幾何関数と方程式.

2.2 グラフ作成

例 2.5 *[4, p.74]* $r = a\sin(2\theta)$ の描く図形を描画しなさい. ここで a は定数.

これを 1 コマンドで書くシステム知らず...

Maxima

超幾何関数のグラフを描いてみよう.

```
plot2d(x*hgfred([1/2,1],[3/2],-x^2),[x,-2,2]);
plot2d(-x*hgfred([1,1],[2],x),[x,-10,0.9]);
plot2d(hgfred([1/2,1/2],[1],x),[x,-10,0.9]);
```

Maxima では $F(1/2, 1/2, 1, x)$ の描画はできないようだ.

Mathematica

Mathematica は超幾何関数の数値計算に強い. (Maple もそうらしい)

```
Plot[x*HypergeometricPFQ[{1/2,1},{3/2},-x^2],{x,-2,2}]
Plot[-x*HypergeometricPFQ[{1,1},{2},x],{x,-10,0.9}]
Plot[HypergeometricPFQ[{1/2,1/2},{1},x],{x,-10,0.9}]

(* Use N[] to get numerical values *)
(* cf. hypergeom([a,b],[c],x) in Maple *)
```

$F(1/2, 1/2, 1, x)$ の描画もきちんとできる.

Maxima

どんな形が想像できる?

```
plot3d( sin(sqrt(x^2+y^2))/sqrt(x^2+y^2),[x,-15,15],[y,-15,15]);
```

いくつか定番の数学的図形.

Mobius の輪.

```
plot3d([cos(x)*(3+y*cos(x/2)),
        sin(x)*(3+y*cos(x/2)),
            y*sin(x/2)],[x,-%pi,%pi],[y,-1,1]);
```

Torus を描く.

```
plot3d([cos(y)*(10.0+6*cos(x)),
        sin(y)*(10.0+6*cos(x)),
        -6*sin(x)],
       [x,0,2*%pi], [ y,0,2*%pi]);
```

Todo: だれでもわかる cycloid. それから, 中川 p.49 (クロソイド), p.113 (ヘンネベルグの極小曲面) を描いてみる.

dynagraph

```
http://www.math.umbc.edu/~rouben/dynagraph
```

```
plot3d(x^2-y^2,x=-2..2,y=-2..2);
```

File, save image file メニューから画像の保存もできます.

Torus を描く.

```
plot3d([cos(y)*(10.0+6*cos(x)),
        sin(y)*(10.0+6*cos(x)),
        -6*sin(x)],
       x=0..6.28, y=0..6.28);
```

ファイルの読み込みは Maple と同じで `read ‘dyna-torus.txt‘;`

Torus は数学では基本図形. (楕円曲線. 平面を格子で割る. Homology 群. ...). 綺麗にかけると, 頭にこれらの理論がうかんできて, やっぱりうれしいのは数学科だから?

問題: 複素楕円曲線が退化して genus 0 の曲線になるアニメを作成せよ.

Mogan のリーマン幾何学 (beginner's guide)[10] の最初の節に煙突に鉄板を巻き付ける問題が提示されている. 巻き付ける曲線の式は

$$(a\cos(t), a\sin(t), ht/2\pi)$$

であり, a, h に応じてどのように巻き付ける鉄板を切るのかは曲率の問題である. それはさておいて, 巻き付ける曲線を描いてみよう.

```
tubeplot([cos(t),sin(t),t],t=0..10);
spacecurve([cos(t),sin(t),t],t=0..10);
```

tube の太さは `tuberadius` パラメータで変更できる.

Knoppix/math には極小曲面 (せっけん膜曲面) を描く強力なソフトウエアが含まれているが (CMCLab), これについては CD の文書フォルダの 小林君の解説が詳しい. 極小曲面の研究には 3 回の流行があり, 現在は第 3 期目らしい. 第 3 期は計算機による数値計算で曲面を描きその結果から定理をいろいろと考えるという研究方法が主流となったらしい. ちなみに, 極小曲面の偏微分方程式は数値的に解くのも大変に難しいので, いろいろと特別な手法 (Poincare-Birkoff-Witt 法) などが開発されている.

Todo: 昨年の wiki も参考に. (CompOne06 および TryGraphics06)

gnuplot

Maxima は実は gnuplot をよびだしてグラフの描画をしている. asir も同じで,

```
gnutplot.gnuplot("plot sin(x)");
```

などと gnuplot の命令を直接実行できたりもする. くわしくは asir-contrib のマニュアルを参照 (http://www.math.kobe-u.ac.jp/OpenXM/Current/doc/index-doc-ja.html)

Windows 版 gnuplot についてのノート.
Q. 起動時のフォントはあまりに小さくて読みにくい.
A. gnuplot の window で右クリックするとフォントの選択メニュ- "Choose Font" メニューがでるので, これでフォントを大きくする. 次回の起動でも有効にするには "Update .... wgnuplot.init" もお忘れなく.

例 2.6 $z = x^2 - y^2$ の $(x, y) = (1, 1)$, $(x, y) = (1, -1)$, $(x, y) = (-1, 1)$ での接平面を求め, それを図示しなさい.

Maxima が自動生成する gnuplot 用の入力 `maxout.gnuplot` も活用できる.

JavaView

(Todo: asir-contrib/taka_jv.rr について sample に解説を加筆.)
簡単な多面体を書かせてみよう. データは obj 形式で与える.
基本となるデータは 2 次元多面体である. 多面体の集まりとして曲面を描画する.

1. v で頂点の座標を指定.
2. f で頂点の番号を指定.

3. `f 1/2 2/1 3/3` とか `f 1/2/1 2/1/2 3/3/2` などと書いてあってもこれは分数ではない. `/` のあとの数字は texture vertex number と vertex normal number である.

obj 形式でのデータの与え方についてくわしくは, `http://www.javaview.de/guide/formats/Format_Obj.html` を参照.

次の例 (jv-simp.obj) は 0, e_1, e_2, e_3 を頂点とする単体を描く.

```
v 0 0 0
v 1 0 0
v 0 1 0
v 0 0 1
f 1 2 4
f 2 3 4
f 1 3 4
```

(実行の仕方は講義メモ 2007-05-18 の方を参照)

2 変数ベッセル関数のグラフの例は takayama のホームページより.

Java と JavaView のライセンス.

OpenGL

math-polyglot を参照.

2.2.1　スライドの作成

スライドには画像をいれたいもの. さまざまな画像の取り込みのヒント.
スクリーンコピーの取り方.

1. Knoppix/math: KDE にたしかスクリーンコピーのコマンドがあったような.
2. Windows: alt + print screen すると画面が clipboard に入るので, 適当な画像エディタに張り付ける. たとえば mspaint (アクセサリ, ペイント) 等.
3. Mac: shift + コマンドマーク (四角いやつ) + 4 を押した後, マウスで範囲を指定すると自動的に png ファイルがデスクトップにセーブされる.
4. unix 一般. `xwd`

ImageMagic の convert コマンドで画像形式を変換可能.

```
convert   pic1.png  pic1.jpg
```

`jpeg2ps` で ps にすればあとは TEX で取り込める.
スライド作成のその他については, ... を参照.

2.3 プログラミング入門

さて次の節で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.

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より複雑な数学実験が可能となる. 序文ではプログラミングの初歩は理解してると仮定していると述べたが, この章では復習もかねて, 数列の漸化式を計算するプログラミングから始める.

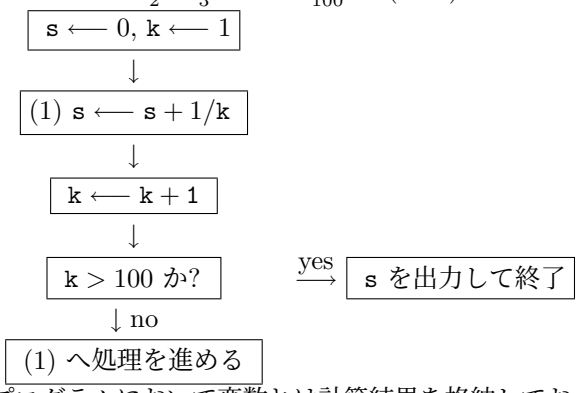
高校の数学では数列の一般項を求める訓練を延々をうける. 計算機言語は数列の一般項を求める魔法の言語ではない. 計算機言語でできることはたとえば漸化式で定義された数列の 100 項目を求めるといった計算である. この節では, 数学はよく知ってるが計算機言語をはじめて使うという読者も対象にプログラムを書くイロハからはじめてみる. プログラムの入門者にとり, 10 進 basic (Knoppix/Edu) や maxima, Risa/Asir はとてもとっつきやすいプログラム言語であろう.

10 進 basic が含まれていない knoppix/math の場合は 10 進 basic をインストールする. 10 進 basic の配布サイトは `http://hp.vector.co.jp/authors/VA008683/`
Knoppix/math の場合は Linux 版 をダウンロード.

2.3.1 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, `s ← s + 1/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 `k ⟵ 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 `k ⟵ 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 k ⟵ 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.

2.3.2　s = s + 1/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 s ← s + 1/k なる意味である.

また, k=k+1 は k ← 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.

2.3.3　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
```

2.3.4　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.

2.3.5 Debug の仕方

print 文をはさむ

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.

2.3.6 他の言語での for と if

例 2.7 (Newton 法による $\sqrt{2}$ の計算) 漸化式 $f_{n+1} = f_n/2 + 1/f_n$, $f_0 = 2$ で定義される数列 $f_1, ..., f_9$ を計算しなさい. f_i の i が奇数の時および f_9 を表示せよ.

C

```
#include <stdio.h>
main() {
  int i;
  double f;
  f=1;
  for (i=1; i<10; i++) {
    f = f/2.0 + 1/f;
    if (i % 2 == 0) {
      printf("%f\n",f);
    }
  }
  printf("%f\n",f);
}
```

```
gcc c-for.c ; ./a.out
```

asir

```
def main() {
  F=1;
  for (I=1; I<10; I++) {
    F = F/2 + 1/F;
    if (I % 2 == 0) {
      print(F);
    }
  }
  print(F);
  print(eval(F*exp(0)));
}
main()$
end$
```

```
load("./asir-for.txt");
```

Maxima

```
f:1;
for i:1 step 1 thru 9 do (
    f : f/2 + 1/f,
    if remainder(i,2) = 0 then print(f)
);
print(f);
print(float(f));
```

```
batch("maxima-for.txt");
```

Todo: 改行, then のあとに複文をどう書くか?

Mathematica

```
f=1;
For[i=1, i<10, i++,
  f = f/2 + 1/f;
  If[TrueQ[Mod[i,2] == 0],
    Print[f]
  ]
];
Print[N[f,30]];
```

```
<<math-for.txt;
```

R

```
f<-1;
for (i in 1:9) {
    f<-f/2 + 1/f;
    if (i %% 2 == 0) {
      print(f);
    }
}
print(f);
```

```
source("r-for.txt");
```

残りは Todo;

Octave

Macaulay2

Singular

Java

Maple

kan/sm1

kan/k0

2.4 数列の計算, maxima, asir 編

例 2.8 [4, p.6] に関連した問題.
漸化式 $a_1 = 2,\ a_{n+1} = \frac{1}{2}(a_n + 1),\ n \geq 0$ で定義される数列を考える. $a_1, \ldots, a_{100}$ を計算せよ. $\lim_{n\to 0} a_n$ の値を予想せよ.

Maxima

```
A:2;
for I:1 step 1 thru 100 do (
  print(A),
  A:(A+1)/2
);


ftest():=block([A,I],
  A:2,
  for I:1 step 1 thru 100 do (
    print(float(A)),
    A:(A+1)/2
  ),
  return(float(A))
);
```

“関数” 定義, 局所変数等は是非覚えておかないといけない基礎的な考え方．超入門参照．Todo: 加筆.

Asir

プログラム例 (漸化式)

```
A = 2;
for (I=1; I<=100; I++) {
  print(A);
  A = (A+1)/2;
}

end$
```

このプログラムを実行してみると $a_{100} = 633825300114114700748351602689/633825300114114700748351602688$ と出力されて結果の予想がつきにくい．このような時は a_n の値をグラフにしてみるとか, 次のプログラムのように近似値を表示させるのも一つの方法である．
注意:　`A=(A+1)/2`　の意味がはっきりわからないときは, “超入門 cfep/asir” [1] をかならず参照.
プログラム例 (漸化式 2)

```
A = 2;
for (I=1; I<=100; I++) {
  print(deval(A));
  A = (A+1)/2;
}

end$
```

この問題を K 君は次のように解いた.

$a_{n+1} = \frac{1}{2}(a_n + 1)$ を変形して, $a_{n+1} - a_n = \frac{1}{2}(a_n - a_{n-1})$．$b_n = a_n - a_{n-1}$ とおくと, $b_2 = 3/2 - 2 = 1/2$, $b_n = -(1/2)^{n-1}$ となる.

いま

$$\begin{aligned} a_n &= (a_n - a_{n-1}) + (a_{n-1} - a_{n-2}) + \cdots + (a_2 - a_1) + a_1 \\ &= \sum_{k=2}^{n} b_k + a_1 \\ &= -\frac{1-(1/2)^{n-1}}{1-1/2}\frac{1}{2} + 2 \end{aligned}$$

となるので, 次のプログラムが解答である.

```
for (I=1; I<=3; I++) {
   A = 2-(1/2)*((1-(1/2)^(I-1))/(1-1/2));
   print(A);
}

end$
```

このプログラム例について次のような意見がある.

意見 1: この方法はこの形の漸化式にしか使えない特別な方法であり, プログラム例 (漸化式 1) のような一般性がない. つまり, プログラム例 (漸化式 1) のようなプログラムなら, たとえば与えられた漸化式が $a_{n+1} = 3a_n + n^5$ のようにかわっても, `A = (A+1)/2` の部分を

```
A = 3*A+I^5
```

と書き換えるだけで使える. よってこのようなプログラムはよくない.

意見 2: 数学的な考察をしてからプログラムを書くのはすばらしい. プログラムも簡潔だ.

意見 3: `(1/2)^I` を何度も計算するのは無駄.

どの意見もごもっともである. よくできたパッケージや組み込み関数では, 特別な場合に通用する方法, 一般的な方法がくみあわせてあって, 最適なものを選ぶようになっている.

疑問: a_n の一般項を求めるパッケージはないの?

商用システムの maple には `rsolve` という関数がある. たとえばこの問題は次のように解ける.

```
> rsolve(f(n+1)=(f(n)+1)/2, f(k));
                                        k         k
                              f(0) (1/2)  - (1/2)  + 1
```

専門家向き 1. (Sloane's encyclopedia of integer sequence)
`http://www.research.att.com/ njas/sequences/` にたとえば `1 1 2 3 5 8` と入力してみよう. Fibonacci 数列という答えがもどり,

```
((1+sqrt(5))^n-(1-sqrt(5))^n)/(2^n*sqrt(5))
```

なる一般項も返事してくれる. この問題のような分数の列には対応してない.

専門家向き 2. `http://algo.inria.fr` から配布している, `gfun` は数列を調べるための maple パッケージ.

例 2.9 [4, p.9 例題 7] に関連した問題.
漸化式 $a_1 = 1$, $a_{n+1} = \sqrt{a_n + 1}$, $n \geq 1$ で定義される数列を考える. $a_1, \ldots, a_{100}$ の近似値を計算せよ. $\lim_{n\to 0} a_n$ の値を予想せよ.

極限を求める数学での答え:

a_n は 有界で単調増加である ([4, p.9] 参照). “有界で単調増加な数列には極限が存在する” のであるから, $\lim_{n\to\infty} a_n = a$ が存在する. 極限の存在が保証されたのであるから, $a_{n+1} = \sqrt{a_n + 1}$ の両辺の極限をとり, $a = \sqrt{a+1}$ がわかる. よって, $a^2 = a+1$. これを解いて, $a = \frac{1\pm\sqrt{5}}{2}$. いま $1 = a_1 < a$ なので, $a = \frac{1\pm\sqrt{5}}{2} \simeq 1.61803$.

Asir

プログラム例 (漸化式)

```
A = 1;
for (I=1; I<=100; I++) {
  print(A);
  A = (A+1)^(1/2);
  A = deval(A);
}

end$
```

専門家向け話題.
Q.

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

を証明したり, 計算するアルゴリズム, システムはないの?
A. “A=B” という本が基本的参考書.

例 2.10 $4 * \tan^{-1}(1)$ およびマチンの公式を用いて π の近似計算を試みよ. 計算速度と精度を比較せよ.

例 2.11 超幾何関数は $\sin^{-1}(x)$ の親玉みたいなもんだから (積分表示をみよ), π が超幾何関数の組合せでかけてもおかしくない. ただしマチンの公式みたいに簡単でない.

$$\frac{1}{\pi} = 12 \sum_{n=0}^{\infty} f(n)$$

```
def f(N) {
  return
  (-1)^N*(( 545140134*N + 13591409)/(640320^(3*N+3/2)))*
     fac(6*N)/( fac(3*N)*fac(N)^3);
}
```

この公式は Chudonovsky が導出した超幾何級数を用いた $1/\pi$ の公式である. これを用いて $1/\pi$ の近似計算 (桁数が多いやつ) を試みよ.

R のプログラミング

最後になるが, 講義のフィードバックでは, 統計システム R の人気が高い. プログラムの例として, 1 次元の乱歩のプログラム (kadota 作) を掲載しておこう.

```
sample<-50000;
for(i in 1:4){
   m <- floor(runif(sample)*100);
   if(m[1]%%2 == 0) { m[1] <- 1; }
   else { m[1] <- -1; }
   for(j in 2:sample){
       if(m[j]%%2 == 0) { m[j] <- m[j-1]+1; }
       else { m[j] <- m[j-1]-1; }
   }
   plot(m, xlim=c(0,sample), ylim=c(-500, 500), type="l");
   par(new=T);
}
abline(0, 0, col="red");
```

(門田作. 2007-05-10). 乱数による一次元の運動のシミュレーション. ベクトル m の大きさは 50000 `m[j]` は `j` 回目の試行による動き. $+1$ か -1 の値が入っている.
結果のグラフより, 一旦上にうごいていくと上に動く傾向が続くこと, 一旦下へ動くと下へ動く傾向が続くことが見えてくる. (逆正弦法則).

統計システム R は `c` 関数で生成する通常のベクトルや `array` 関数で生成する通常の配列以外に, factor (因子), data frame(データフレーム) といった統計用の特別のデータ型が用意されている. また普通のプログラム言語でいう `map` 関数相当の機能も充実している. `R-intro-jp.170.pdf` が基本文献だが, この文献のわかりにくい点を解説しながら, これらの普通の言語との違いを重点的に例を挙げて説明したい (todo).

関数 (手続き) の定義

困難は分割せよ
`mathpolyglot/src/lang-func` を参照.

2.5 2 変数関数の最大値最小値問題

2 次元グラフの作成はまず 1 次元の切口を沢山書くこと.
@s/2007/03/knx* より.
Six-hump camel back function.

$$x_1^2(4 - 2.1x_1^2 + x_1^4/3) + x_1x_2 + x_2^2(-4 + 4x_2^2)$$

Bruce force method. $f_x = 0, f_y = 0, f = 0$ を解いて全部の点を調べる.

関連図書

[1] 高山信毅, 超入門 cfep/asir,
http://www.math.kobe-u.ac.jp/HOME/taka/2005/cfep/pdf/next2.pdf

[2] 中川義行, Maxima 入門ノート 1.2.1, 2006.
KNOPPIX/Math, 2006,
Desktop/knoppix-math-ja/PDF/maxima-note.pdf

[3] Maxima マニュアル改訂版, 2006.
KNOPPIX/Math, 2006,
Desktop/knoppix-math-ja/PDF/MaximaManualBeta.pdf

[4] 林平馬, 岩下孝, 浦上賀久子, 今田恒久, 佐藤良二, 微分積分学序論, 学術図書出版社.

[5] 三宅敏恒, 微分積分

[6] M.Bronstein, *Symbolic Integration I*, Algorithms and Computation in Mathematics, Springer, 1997.

[7] J.Grabmeier, E.Kaltofen, V.Weispfenning, editors. Computer Algebra Handbook, Springer, 2001.

[8] 渡辺隼郎, 常微分方程式の数式処理, 教育出版, 1974

[9] 河野実彦, 常微分方程式と数式処理, 森北出版, 1998

[10] Frank Mogan, リーマン幾何学 (beginner's guide).

[11] 佐藤ふさお, 自然の数理と社会の数理, 1984, 日本評論社.

第3章　微分方程式の近似解法

微分方程式の近似解法により多くの代数方程式系の例題を得る事ができる. 代数方程式系の議論にはいる前に微分方程式の近似解法の議論をおこなおう.

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.

差分法は数列の漸化式の計算にほかならない. 定常状態の解を求めることは, 連立代数方程式系の解を求めることにほかならない. 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.

3.1　差分法

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.

3.1.1　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.

3.1.2 連立方程式の差分化

問題 3.1 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 3.2 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.

3.1.3　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.

さて, Lorentz 方程式に定常状態はあるか? という問題を考えてみよう. 定常解とは, 時間変化のない微分方程式の解である. つまり $y' = f(y)$ が与えられた方程式として,

$$f(y) = 0 \tag{3.3}$$

の解である. Lorentz 方程式の場合, 未知変数 p_1, p_2, p_3 に対する連立代数方程式

$$\begin{aligned} 0 &= -ap_1 + ap_2, \\ 0 &= -p_1p_3 + bp_1 - p_2, \\ 0 &= p_1p_2 + cp_3 \end{aligned}$$

の解が 定常状態 である. その中でさらに安定な定常状態も興味深い.

解があるか, あるとして有限か, 無限か? パラメータにどう依存するのか, これらの疑問に答えるにはこのサイズの問題であればグレブナ基底の方法が一番であろう. これについては, 4.6.1 のプログラムで議論する.

3.1.4 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.

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,

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

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

$$y_{k+2} = 2y_{k+1} - y_k - h^2 h_{k+1},\ y_0 = 1, y_1 = 1 \tag{3.4}$$

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 (3.4) 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 3.3

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,

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

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)
```

3.1.5　Risa/Asir で書くと?

```
load("glib3");

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$
```

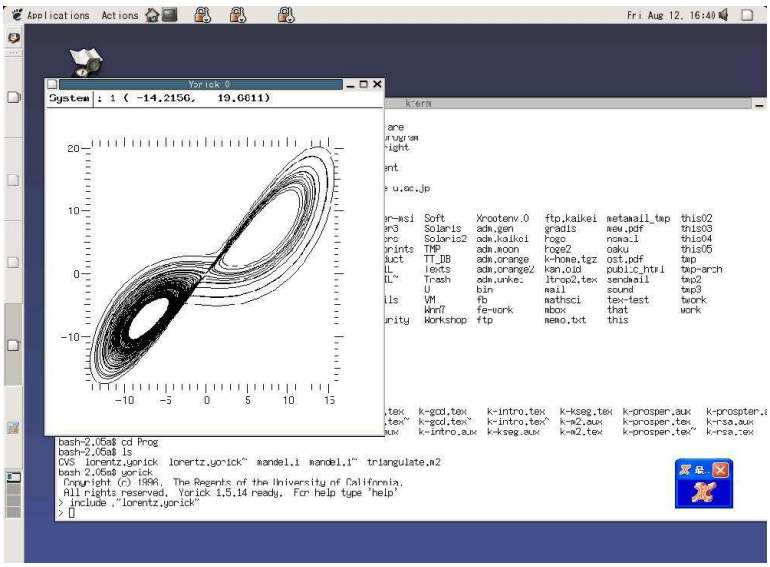
3.1.6　yorik によるシミュレーション

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．



3.1.7 定常状態に対応する代数方程式系

式 () は定常状態の満たすべき代数方程式系である．偏微分方程式を考えると定常状態のみたすべき方程式系もまた微分方程式系となる．それを差分化することにより連立代数方程式系を得る．これを Knoppix/math を利用して解くのはわれわれの興味の一つである．

たとえば次の反応拡散方程式を考えよう．

$$\frac{\partial u}{\partial t} = \epsilon \frac{\partial^2 u}{\partial x^2} + u(1-u), \quad u(0) = u(1) = 0 \text{ 境界条件} \tag{3.5}$$

ϵ はパラメータであり, その大きさにより解 $u(t,x)$ の様子はいろいろと変化する．ちなみにこの問題はきわめて古典的な問題で, 理論的に詳しく解析されている．理論的に詳しく解析されている問題は新しい計算手法の研究を始めるとき (答えがわかってるという意味で）きわめて有益である．Todo: sos より参考文献をコピー．

Mathemaitca には偏微分方程式の数値解析を自動でやる関数も用意されている．簡単な問題は OK だが, かならずしも正しい答えを戻すとは限らないので注意．

```
(*
http://documents.wolfram.com/v5/Built-inFunctions/NumericalComputation/EquationSolving/NDSolve.ja.html

http://documents.wolfram.com/v5/Built-inFunctions/NumericalComputation/EquationSolving/NDSolve.html

http://documents.wolfram.com/mathematica/Built-inFunctions/NumericalComputation/EquationSolving/FurtherExamples/NDSolve.h

*)

(* p.931 *)
wave[]:=Module[{ans},
  ans=NDSolve[{ D[y[x,t],t,t] == D[y[x,t],x,x],
                y[x,0] == Exp[-x^2],
                Derivative[0,1][y][x,0] == 0,
                y[-5,t] == y[5,t]},
                y, {x,-5,5}, {t,0,5}];
  Plot3D[Evaluate[y[x,t] /. First[ans]],
         {x,-5,5}, {t,0,5}, PlotPoints->30]
]

nlwave[]:=Module[{ans},
  ans=NDSolve[{ D[y[x,t],t,t] == D[y[x,t],x,x] + y[x,t]^2,
                y[x,0] == y[x,1],
                y[0,t] == 0,
                y[1,t] == 0},
                y, {x,0,1}, {t,0,1}];
  Plot3D[Evaluate[y[x,t] /. First[ans]],
         {x,0,1}, {t,0,1}, PlotPoints->30]
]

nlwave2[]:=Module[{ans},
  ans=NDSolve[{ D[y[x,t],t,t] == D[y[x,t],x,x] + y[x,t]^2,
                y[x,0] == y[x,1],
                Derivative[0,1][y][x,0] == Derivative[0,1][y][x,1],
                y[0,t] == 0,
                y[1,t] == 0},
                y, {x,0,1}, {t,0,1}];
  Plot3D[Evaluate[y[x,t] /. First[ans]],
         {x,0,1}, {t,0,1}, PlotPoints->30]
]

nlwave0[]:=Module[{ans},
  ans=NDSolve[{ D[y[x,t],t,t] == D[y[x,t],x,x] + y[x,t]^2,
                y[x,0] == Exp[-(x-1/2)^2],
                Derivative[0,1][y][x,0] == 0,
                y[0,t] == 0,
                y[1,t] == 0},
                y, {x,0,1}, {t,0,1}];
  Plot3D[Evaluate[y[x,t] /. First[ans]],
         {x,0,1}, {t,0,1}, PlotPoints->30]
]
```

さてここでは, この様子を観察する C 言語のプログラムを掲載しておこう. Knoppix/Math は C 言語の学習/開発環境としてもすばらしい. (**複雑なプログラムぎらいの Knoppix/Math ユーザは読み飛ばして結果の動画だけを楽しまれたい.**)

なお, X11 で描画する版と 山口君の作成した gif を生成するプログラム (cgif, GPL) `http://www.math.kobe-u.ac.jp/HOME/yamaguti/` を図形の描画には利用している.

フォルダ cgif 以下には cgif ライブラリがおいておく. 次の Makefile では `make pt` で X11 で描画, @verb@ make pt2 @ で cgif ライブラリで描画する版を生成する.

```
pt : pt.c glib.o
gcc -o pt pt.c glib.o -L/usr/X11R6/lib -lX11 -lm

glib.o: glib.c
gcc -I/usr/X11R6/include -c glib.c

pt2: pt.c glib_cgif.o cgif.o
gcc -o pt2 pt.c glib_cgif.o cgif.o -lm

glib_cgif.o: glib_cgif.c
gcc  -c glib_cgif.c

cgif.o: cgif/cgif.c
gcc  -c cgif/cgif.c

clean:
rm *.o *~ pt pt2
```

```
#include <stdio.h>
#include <math.h>
#define PI  3.14
/* #define Heat_N  40*/  /* Mesh 分割数,  */
#define Heat_N  3  /* Mesh 分割数,  */
double DD = 1;     /* パラメータ d */
double AA = 1;    /* パラメータ a */

main(int argc,char *argv[]) {
  double K;  /* dt */
  int M;  /* steps */
  float A0; /* a */

  double H;
  double t;
  double C;
  double A[Heat_N+1];
  double B[Heat_N+1];
  int P,Q;
  int frame_max = 20;
  int frame_dt = 0;


  /* printf("0.00001, dt=?"); scanf("%f",&K); */
  K = 0.00001;
  M = 50000;
  if (argc > 1) sscanf(argv[1],"%d",&M);
  printf("M=%d. a=? (1, 11, 20) ",M); scanf("%f",&A0);

  frame_dt = M/frame_max;
  glib_init_frame_no(frame_max);
  glib_window(0.0,0.0,(double) Heat_N,1.0);
  H = 1.0/Heat_N;   /* 空間差分 */
  AA = A0;
  printf("時間差分 K="); printf("%f\n",K);
  printf("CFL like 条件. must be >0. 1-2*K*d*/(H*H)-a*K: ");
  printf("%f\n",1-2*K*DD/(H*H)-AA*K);
  t = (4*DD/(H*H))*sin(H*PI/2)*sin(H*PI/2);
  printf("分岐値 (4d/h^2)*sin^2(h*pi/2) <> a: %f\n",t);
  printf("a=%f\n",AA);

  A[0] = 0; A[Heat_N] = 0;
  /* 初期条件の設定 */
  C = 1;
  for (Q=1; Q<Heat_N; Q++) {
     if (Q <= Heat_N/2) {
        A[Q] = H*Q*C;
     }else{
        A[Q] = (1-H*Q)*C;
     }
  }

  for (P=1; P<=M; P++) {
    B[0] = 0; B[Heat_N]=0;
    for (Q=1; Q<Heat_N; Q++) {
      B[Q] = A[Q] + DD*(K/(H*H))*(A[Q+1]-2*A[Q]+A[Q-1])
             + K*AA*(1-A[Q])*A[Q];
    }
    /* print("Time=",0); print(P*K,0); print(":  ",0); print(B);*/
    /* code for DISPLAY  */
glib_clear();
if ((P-1) % frame_dt == 0) { glib_set_frame_no((P-1)/frame_dt); }
for (Q=0; Q< Heat_N; Q++) {
  glib_line((double)Q,A[Q],(double)(Q+1),A[Q+1]);
}
glib_cgif_pen_up();
    if (P % 1000 == 0) {printf("P=%d\n",P);}
glib_flush();
for (Q=0; Q<= Heat_N; Q++) A[Q] = B[Q];
  }
  printf("Type in enter to finish."); fflush(NULL);
  getchar(); getchar();
  glib_close();
}
```

ちなみにこのプログラムは変数に大文字が使われている. C 言語のスタイルととしては一般的ではない. 実は asir でプロトタイプを作成してから C 言語で書き直したので, 大文字の変数名が多い.

Todo: 結果の動画を web に.

さて 式 (3.6) の定常解のみたすべき方程式は次の代数方程式系である.

$$0 = \epsilon \frac{\partial^2 u}{\partial x^2} + u(1-u), \quad u(0) = u(1) = 0 \text{ 境界条件} \tag{3.6}$$

$$\epsilon \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} u_i(1-u_i) = 0, \quad i = 1, 2, 3, \ldots, n-1 \tag{3.7}$$
$$u_0 = 0, u_n = 0, h = \frac{1}{n}$$

なる連立の非線形代数方程式系となる. この方程式系は **??** で解析する.

もう一つ興味深い古典的なシステムとして, 三村が 1970 年代に解析した反応拡散系 [3] を紹介しておこう. Turing's diffusion instability principle を実際にみることができる.

$U(t,x)$ と $V(t,x)$ が未知関数であり,

$$\begin{aligned} \frac{\partial U}{\partial t} &= d_1 U'' + (f_1(U)U - kUU) \\ \frac{\partial U}{\partial t} &= d_2 V'' + (-f_2(V)V + kUV) \end{aligned}$$

なる発展方程式を考える. ここで $U'' = \frac{\partial^2 U}{\partial x^2}$ である. われわれはこれの定常状態の解 $u(x) = U(0,x)$, $v(x) = V(0,x)$ を調べたい. これは次の常微分方程式系を満たす.

$$\begin{aligned} & d_1 u'' + (f_1(u)u - kuv) = 0, \\ & d_2 v'' + (-f_2(v)v + kuv) = 0, \end{aligned} \tag{3.8}$$

ここで

$$\begin{aligned} f_1(z) &= (1/9) * (35 + 16 * z - z^2), \\ f_2(z) &= 1 + (2/5) * z, \\ 0 <= x <= 5, & \\ k = 1, d_2 = 4. & \end{aligned}$$

d_1 は小さい大きさのパラメータである. `http://fe.math.kobe-u.ac.jp/nobuki/sos/pps-1.png` は $d_1 = 1/20$ の時の一つの解である. その他にも非負の解がある. 赤がえさ (prey) u で, 黒が捕食者 (predator) v である.

Asir による数値解析のプログラムは

```
def draw_ax() {
  glib_line(-2,0,5,0 | color=0x00ff00);
  glib_line(0,-2,0,10 | color =0x0000ff);
}
import("glib3.rr");
import("gr");
Glib_math_coordinate=1;
Cfep_loaded = 0; /* use X11 */
glib_window(-2,-2,5,12);
glib_clear();
draw_ax();
/* Masayasu Mimura, Asymptotic behaviors of a parabolic system related to a
planktonic prey and predator model. SIAM J on Applied Mathematics, 1979.
*/

N=20;  Dt=0.005;  /* N=21 --> N=20 variables.  N=20 --> N=19 variables */
U=newvect(N+1);
V=newvect(N+1);
Unew=newvect(N+1);
Vnew=newvect(N+1);
K=1; L=5;

Pu=0.05; Pv=4.0;   /* diffusion constants */

for (I=0; I<=N; I++) {
  U[I] = 0.0;
  V[I] = 0.0;
}
V[0]=V[1]; V[N]=V[N-1];
NN=idiv(N,10);
for (I=NN*3; I<=NN*4; I++)  U[I] = deval(2*(L/N)*(I-NN*3));
for (I=NN*4; I<=NN*5; I++)  U[I] = deval(2*(L/N)*(NN*5-I));
for (I=NN*5; I<=NN*6; I++)  V[I] = deval(2*(L/N)*(I-NN*5));
for (I=NN*6; I<=NN*7; I++)  V[I] = deval(2*(L/N)*(NN*7-I));

H = deval(L/N);
for (T = 0.0, J=0; T<100; T += Dt, J++) {
 for (I=1; I<N; I++) {
   Unew[I] = U[I]+Dt*Pu*(U[I-1]-2*U[I]+U[I+1])/(H*H) +
                 Dt*( (1/9)*(35+16*U[I]-U[I]*U[I])*U[I] - K*U[I]*V[I] );
   Vnew[I] = V[I]+Dt*Pv*(V[I-1]-2*V[I]+V[I+1])/(H*H) +
                 Dt*(  - (1+(2/5)*V[I])*V[I] + K*U[I]*V[I] ) ;
 }
 Unew[0] = Unew[1]; Vnew[0] = Vnew[1];
 Unew[N] = Unew[N-1]; Vnew[N] = Vnew[N-1];

 if (J % 20 == 0) { glib_clear(); draw_ax(); J=0;}
 for (I=0; I<N; I++) {
   glib_line(I*H,U[I],(I+1)*H,U[I+1] | color=0xff0000);
   glib_line(I*H,V[I],(I+1)*H,V[I+1] );
 }
 glib_flush();

 for (I=0; I<=N; I++) {
   U[I] = Unew[I]; V[I] = Vnew[I];
 }
}

bsave(U,"tmp-pps-u-n-"+rtostr(N)+".bb");
bsave(V,"tmp-pps-v-n-"+rtostr(N)+".bb");
```

Gröbner 基底による解析は 4.6.3 節で試みる.

3.1.8 Cavity flow

差分法からでてくる非線型代数方程式をグレブナ基底の手法等でどのくらい解けるか試すのは挑戦問題として興味深い.

Cavity flow とは何か?

Todo: 方程式はまだ書いてない. メモ用紙に書いたノートはあり.

Cavity flow のシミュレータ Prog/cavity.c は未完成なので, “矢川元基, パソコンでみる流れの科学” [2] chap4 の cavity flow のシミュレータで渦が生成される様子をみてみよう. `2007/02/knx-*` へコピー済.

研究課題: `sos/Prog/fluid2d.rr` でみるように, グレブナ基底の手法は小さいサイズの問題しか解けない. しかし, 解こうとしている連立非線型方程式系の代数的性質を探求したり Raynolds 数 (パラメータ) のこの方程式での代数的役割を調べたり, いろいろと別の角度から問題をみれるのではないか? また, グレブナ基底の手法をこのような問題に適用する試みはまだあまりない.

3.2 フラクタルとソリトン: ちょっと寄り道

(補足)

カオス, フラクタルとソリトンは計算機の研究への利用の老舗である. ちょつとこの分野の不思議な図を寄り道してみておこう. この分野にも代数方程式がいろいろと登場する.

3.2.1 フラクタルって何?

3.2.2 Risa/Asir による Mandelbrot 型集合の描画

```
/* 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$
```

Todo: Newton 法の元の式を計算数学 2 のノートか arima から探す.

3.2.3 Yorick による Mandelbrot 型集合の描画

```
/*  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;
```

3.2.4 C と X11 による Mandelbrot 集合の描画

Newton 法でなく, つぎの漸化式を収束させる複素数パラメータ a の値がよくみる ひょうたん型の元祖 Mandelbrot 集合である.

$$z_0 = 0, z_{n+1} = z_n^2 + a$$

n を 1000 まで動かして 発散しなければ黒. $|z_n|^2 > 2$ となったらそのときの n の値に応じて色を変更する. ここでは C と X11 で書いてみる. 問題に応じて適切な言語を選択するのは大事であろう.

```
#include <stdio.h>
#define  MAG   100.0
main() {
  double Ar, Ai, X, Y, X2, Y2;
  int I;
  glib_window(-2.0, -1.0, 1.0, 1.0);
  for (Ar=-2.0; Ar < 1.0; Ar += 1.0/MAG) {
    glib_putpixel(Ar,0.0);
  }
  for (Ai=-1.0; Ai < 1.0; Ai += 1.0/MAG) {
    glib_putpixel(0.0,Ai);
  }

  for (Ar = -2.0; Ar < 1.0 ; Ar += 1.0/MAG) {
        for (Ai = -1.0; Ai <1.0; Ai += 1.0/MAG) {
          X = 0.0 ; Y = 0.0;
          for (I=0; I<100; I++) {
                X2 = X*X-Y*Y+Ar;
                Y2 = 2*X*Y + Ai;
                if (X2*X2+Y2*Y2 > 2) {
                  if (I < 3) {
                    glib_color(0xff0000);
                    glib_putpixel(Ar,Ai);
                  }else {
                    glib_color(0x0000ff);
                    glib_putpixel(Ar,Ai);
                  }
                  /* printf("%f %f\n",Ar,Ai); */
                  break;
                }
                X = X2; Y = Y2;
          }
        }
        gFlush();
  }
  printf("Finished\n");
  getchar();
  gclose();
}
```

一般的に, ソフトウエアを開発するときは適切な言語を選んでプログラムを書こう. たとえば Java は GUI を書くのに便利. Mac の CoCoA なども. 例. cca, cfep

3.2.5　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.

3.2.6 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
```

3.3　級数解, Fourier 展開

3.3.1　べき級数解

3.3.2　Maple

Todo: `DEtools`

Asir

Asir では多項式は再帰表現形式であらわされている．この形ではベキ級数を表示するのに都合がわるいので次の関数が用意されている．

```
poly_sort((x-y-a)^3 | v=[x,y], w=[-1,-1]);
```

例 3.1 次の級数が 2 次方程式をみたすことを確かめよ．超幾何級数が微分方程式をみたすことを確かめよ．

Todo: まだ書いてない．

次の例題として “河合, 竹井, 特異摂動の代数解析学” [4] の p.21 の WKB 形式解を構成してみよう．WKB 法に興味のある読者は上記の本を読んでみられるとよいであろう．この本を読むための予備知識としては関数論, および, 複素領域の微分方程式に関する知識が必要である．“高野, 常微分方程式” などを参照してほしい．

例 3.2 次の漸化式できまる S_j を計算するプログラムを書きなさい．

$$\begin{aligned} S_{-1}^2 &= \sqrt{x} \\ 2S_{-1}S_j &= -\left(\sum_{k+\ell=j-1,k,\ell\geq 0} S_k S_\ell + \frac{dS_{j-1}}{dx} \right) \end{aligned}$$

$$S_{odd} = \sum_{j\geq 0} S_{2j-1}\eta^{1-2j}$$

の部分和をもとめるプログラムを書きなさい．

ここでは $x^{1/2}$ の級数があらわれる．分数ベキの多項式をなんとか扱える数式処理システムはいろいろとあるが, 完全に正しく扱えるシステムはないと思った方がよい．たとえば $x^{1/2}y^{1/2} = (xy)^{1/2}$ とは限らない．多項式の計算を数式処理系は正しく扱える．したがって, $y = x^{1/2}$ とおき直してきめ細かな処理をしてやろう．

$$\frac{d}{dx}f(x^2) = \frac{df(y)}{dy}\frac{1}{2}x^{-1/2} = \frac{df(y)}{dy}\frac{1}{2y}$$

に注意してすべてを y の級数として処理する．関数 `setSS()` は S_j を $j = 9$ まで計算する．

次は S_j を計算する Asir によるプログラムである．

```
/* airy.rr */
/* Kawai-Takei, p.14, p.21 */
Airy = quote( -dx^2+ eta^2*x)$
SS=newvect(10)$

/* y = x^(1/2) */
def ss(J) {
  extern SS;
  if (J == -1) return y;
  if (SS[J] != 0) return SS[J];
  S = diff(ss(J-1),y)/(2*y);
  for (K=0; K<J; K++) {
    L = J-1-K;
    S = S+ ss(K)*ss(L);
  }
  return -red(S/(2*y));
}

def setSS() {
  extern SS;
  for (I=0; I<10; I++) SS[I]=ss(I);
}

Psi = quote((1/s(x)^(1/2))*exp(integral(s(t),t,x0,x)))$
/*where s(x) = ss(-1)*eta^1+ss(1)*eta^(-1)+ss(3)*eta^(-3)+... */


end$
```

本を読みながら, プログラムを書いてみることは, 紙で計算するのと似て本の理解をすすめる上で有益である. 本を読みながら書いてみたプログラムを本書の例題として採用している. Asir の場合公式を quote 形式で入力していくのも面白い. 変数の依存関係や公式の理解なしに正しく入力するのがむつかしいからである.

例 3.3 構成した級数が η 付き Airy 方程式をみたすことを確かめよ.

Maxima

Maxima でべき級数を扱う方法.

例題 3.2 を Maxima で解いてみよう. Maxima の場合 x の分数ベキを計算するので x の分数ベキのまま計算してみることにする. `setSS();` を実行すると `ss[-1]`, `ss[0]`, ... に S_j の値が代入される.

```
ss[-1]:x^(1/2);
ss(j):=block([s,k,L],
  s:diff(ss[j-1],x)/(2*x^(1/2)),
  for k:0 thru j-1 do (
    L : j-1-k,
    s : s+ss[k]*ss[L]
  ),
  ss[j]: s/(2*x^(1/2)),
  return(ss[j])
);

setSS():=block([i],
  for i:0 thru 9 do ( ss(i) )
);
```

Binary splitting algorithm による pi の計算．gmp ライブラリを見よ．Todo: fb のフォルダに binary splittting の論文ある?

3.4 Wave ファイル — C 言語による処理

Mathematica には `Play` コマンドというユニークなコマンドがある. これは波形のグラフを与えて音を生成するコマンドである.

音と数学の関係はさまざまである. ラプラス方程式の固有関数はその物体の音色をきめる. 固有値はその物体の作る音の音程を決める. このあたりの説明は物理の本に譲ることにして, この節ではデータから音を作るソフトウエアを作成するヒントを解説することとしよう.

とりあえず knoppix/math に適当なソフトをみつけられなかったので, C 言語で簡単なものを自作する. これを参考にこのような数学ソフトの開発に挑戦してみてくれるとうれしい.

C 言語については, 本格的に習得するには (1) とりあえず C 言語のわかりやすい入門書を読む. ポインターやバイトの説明がきちんとしてあるものが望ましい. たとえば `char` はけっして文字ではなく, 8 bit = 1 byte の大きさの変数の型であると説明してある本はよい本であろう. (2) B.W.Kernighan, D.M.Ritchie の C 言語の本 [1] を熟読し, 練習問題をやる. ことをお勧めする.

さて音楽ファイルの簡単な取り扱いは C 言語の初心者にも十分楽しめる話題であろう. C 言語の使いやすさを理解するには最適である. C 言語を利用したことがないと仮定して順をおって説明する. 予備知識, 予備的準備は以下のとおりである. 興味をもたれたら上記の本格的な C 言語のはじめてほしい. 現在のソフトウエアへの C 言語の影響はとても大きい.

Knoppix/math を利用して実習する場合の予備知識

1. ターミナル (shell) の起動法を知っている.
2. cygwin shell の基本 4 コマンド ls, cd, cp, man を使える.
3. Basic, maxima, asir, などプログラムを使ったことがあり, 2 大基礎概念, 変数, 繰り返し (for) を知っている.
4. テキストエディタ emacs や KDE のテキストエディタを使える.
5. 拡張子を知っている.
6. ファイルを 16 進数としてみる観察する方法を知っている. (hexdump コマンドを使える)

実習プログラムを cygwin の上で実行する様子およびプログラムの解説については次のビデオも参照. 録画まだ.

Windows で利用して実習する場合の予備知識

1. cygwin (gcc を利用) をインストールしてある. 参考文献:??
2. cygwin ターミナル (shell) の起動法を知っている.
3. shell の基本 4 コマンド ls, cd, cp, man を使える.
4. エクスプローラから `c:\cygwin\home\ユーザ名` へ到達できる.
5. ¥と \ が同じ文字コードに対応していることを知っている.
6. Basic, maxima, asir, などプログラムを使ったことがあり, 2 大基礎概念, 変数, 繰り返し (for) を知っている.

7. テキストエディタメモ帳 (notepad) を使える.
8. 拡張子を知っている.
9. ファイルを 16 進数としてみる観察する方法を知っている. (dump, fbdev など)

実習プログラムを cygwin の上で実行する様子およびプログラムの解説については次のビデオも参照.

Mac OS X を利用して実習する場合の予備知識

1. Xcode (gcc を利用) をインストールしてある. (インストール済みの場合が多い)
2. ターミナル (shell) の起動法を知っている.
3. shell の基本 4 コマンド ls, cd, cp, man を使える.
4. Basic, maxima, asir, などプログラムを使ったことがあり, 2 大基礎概念, 変数, 繰り返し (for) を知っている.
5. テキストエディタ (plain テキスト) や emacs を使える.
6. 拡張子を知っている.
7. ファイルを 16 進数としてみる観察する方法を知っている. (hexdump -C など)

3.4.1　はじめのプログラム

次のプログラムは　Hello を 17 回数表示し, 変数 c を 16 進数で表示するプログラムである.

```
#include <stdio.h>
main() {
  int c;
  c = 16;
  while (c >= 0) {
    printf("%x ",c);
    printf("Hello\n");
    c = c-1;
  }
}
```

`int c` は変数 `c` の宣言である. `int` は整数型 (integer type) と呼ばれるが決して数学でいう整数ではない. 32bit までの有限桁の整数しか保存できない.
`while` 文のあとの中括弧で囲まれた部分を $c \geq 0$ である限り繰り返す. `%x` は `c` を１６進数で表示せよという意味. `Hello` と表示してから `\n` で改行する.
課題: 32 回繰り返すようにプログラムを変更せよ.

課題: このプログラムを `dump0.c` という名前で作成して実行してみなさい.

```
$ gcc dump0.c

$ ./a
10 Hello
f Hello
e Hello
d Hello
c Hello
b Hello
a Hello
9 Hello
8 Hello
7 Hello
6 Hello
5 Hello
4 Hello
3 Hello
2 Hello
1 Hello
0 Hello
```

なお, これは cygwin での実行例で, unix や Mac の場合は./a.out で実行開始.

while の使い方. printf の使い方.

```
while（条件）{
   xxxx;
   yyyy;
   zzzz;
}
```

と書くと条件が真である限り 中括弧で囲ってある xxxx; yyyy; zzzzz; の部分を実行する. 中括弧で囲ってあるひとまとまりをブロックとよび字下げして見やすくするのが慣習である. **字下げを忘れない**

上のプログラムでは $c = c - 1$ なる文がある. これがないと c の値は普遍で while が永遠にくりかえされることとなる (無限ループとよぶ). (ctrl+C で中断できる.)

C 言語では 0 でない数で真をあらわす. C 言語の本を参照. printf の使い方については C 言語の本 or video を見てほしい.

練習問題: 2^k, $k = 1, \ldots, 8$ の表を作るプログラムを書け.

getchar() の使い方と標準入力.

ファイルはディスクの中に格納されている 1 バイトデータの一次元的な列に名前をつけたものである. プログラム, 図形データや文書, 音楽, 画像はすべて 1 バイトデータの一次元的な列であり, ファイルとして格納される. C 言語を用いるとバイト列としてのファイルの扱いを簡単に行なうことが可能である. まず C 言語からファイルを扱う関数の代表は getchar() (読む), putchar() (書く) である. char は character (文字) の略であるが, これらの関数は “文字を扱うわけとは限らない. 実は 1 バイトのデータを扱うんだと” いう部分が初心者の陥りやすい誤解である. ファイルはバイトの数列であり, 関数 getchar() は標準入力より 1 バイト読み込む. 標準入力はプログラムの実行の時に < のあとに指定する. 次の例では 読み込むファイルは test1.txt である. このファイルは Windows の場合は notepad で作成しておく.

ダンププログラム.

```
#include <stdio.h>
main() {
  int c;
  c = getchar();
  while ( c >= 0) {
    printf("code=%x [%c]\n",c,c);
    c = getchar();
  }
}
```

dump1.c の解説. 関数 (手続き) getchar() はファイルの内容を一バイトづつ戻す. ファイルの終わりに達すると-1 を戻す. 戻すというのはプログラム言語特有の言い方で, 要するに関数の値のことである. したがって, このプログラムはファイルから１バイトづつ読み込みそのコードを１６進数で表示し, さらにテキストエディタでこのファイルをみたときどのように表示されるかを表示する (printf の　%c). 次の例では putchar() を用いてファイルを加工する.
課題: 入力されたファイルの長さが何バイトあるか求めるプログラムを書きなさない.

読み込むファイル test1.txt. このファイルは Windows の場合は notepad, Mac や unix の場合は emacs で作成しておく. test1.txt

```
AAA
aaa
111!
```

出力結果.

```
$ gcc dump1.c

$ ./a <test1.txt
code=41 [A]
code=41 [A]
code=41 [A]
code=a [

]
code=61 [a]
code=61 [a]
code=61 [a]
code=a [

]
code=31 [1]
code=31 [1]
code=31 [1]
code=21 [!]
code=a [

]
code=a [

]
```

課題: 出力結果をアスキーコード表をみて説明せよ.

簡単な暗号化.

```
#include <stdio.h>
main() {
  int c;
  while ( (c=getchar()) >= 0) {
    if (c != 0xa) {
      putchar(c+1);
    }else{
      putchar(c);
    }
  }
}
```

プログラム dump2.c の説明. if 文は次のように使う.

```
if（条件）{
   条件が成立するときの処理.
}else{
   条件が成立しないときの処理.
}
```

処理の部分は複数の分を ; (セミコロン) で区切って書いてよい.
puthcar(c) では > で指定したファイルへデータを 1 バイトづつ書き込む. 次の実行例ではファイル test2.txt へ書き込まれる. なお c の値が 1 バイト以上（つまり 256 以上だとその部分は無視される.) たとえば c=0xff41 だとファイルには 0x41 が書き込まれる.
0xa は unix や mac の改行のコードである.
課題: test2.txt の出力はどうしてこうなっているのかアスキーコード表を用いて答よ.
課題: putchar(c+1) を putchar(c+2) とするとどうなるか？

dump2.c の出力結果.

```
$ gcc dump2.c

$ ./a <test1.txt >test2.txt

$ cat test2.txt
BBB
bbb
222"
```

dump2.c は簡単な暗号化のプログラムでもある. 簡単な復号化を紹介しておこう.

```
#include <stdio.h>
main() {
  int c;
  while ( (c=getchar()) >= 0) {
    if (c != 0xa) putchar(c-1);
    else putchar(c);
  }
}
```

さて, バイトの数列としてのファイルの取り扱いの準備が完了した. いよいよ音にノイズを加えるプログラムを紹介しよう. 実習用の wave ファイルは http://www.math.kobe-u.ac.jp/HOME/taka/2005 より prelude-2-a-3.wav をダウンロードしよう.

```
/* 44.1KHz, 16 bit, stereo only */
/* noise1 <file-name.wav > t.wav */
#include <stdio.h>
main() {
  int c,i;
  int rate;
  rate = 10;
  for (i=0; i<44; i++) {
    c = getchar();
    putchar(c);
  }
  while ((c=getchar()) >= 0) {
    if (random() % rate == 0) putchar(c +  random());
    else putchar(c);
  }
}
```

この wave ファイルの最初 44 バイトはさまざまな情報たとえば, 音楽ファイルの長さ, ステレオのチャンネル, サンプリングレート等が書かれている. これらについては参考資料 ??? を参照して欲しい. この部分を書き換えると再生ソフトが再生できなくなるので変更していない.

そのあとは 乱数を `random()` で発生し, その数を `rate` で割った余り　`%`　が 0 ならデータ c に乱数を足し, そうでなければそのまま標準出力へ書出している.

一秒間が 44100 に分割されて音楽のグラフが 4 byte 単位で保存されている. 最初 2 byte が左スピーカのデータ, 次の 2 バイトが右スピーカのデータである. 図を参照 (まだ書いてない).

初級問題:

1. バイト数を勘定する.
2. noise1.c を改良して, ノイズを増加させよ.
3. noise1.c を改良して 無音の時間を作れ. ヒント: 0 は無音である.

上級問題

1. 逆順にする.
2. 適当なラプラシアンを選び固有関数の音を作れ.

関連図書

[1] B.W.Kernighan, D.M.Ritchie, C Programming Language (2nd Edition), Prentice Hall, 1988.
日本語訳: プログラミング言語 C (第 2 版), 共立出版
プログラミング言語 C のもっとも基本的かつ重要な文献. 世界的なベストセラーでもある. この本ではさまざまな例題とともに文字列処理も解説している.

Risa/Asir は C の文法と似た文法でプログラムを記述する. Maple や Mathematica などの数式処理システムは独自の言語でプログラムを記述しないといけない. 習得には少々時間がかかる. それに比較して, Risa/Asir は C や Java を知ってる人は, すぐプログラムが書ける. また, C や Java を知らない人は, 簡単にプログラムがためせる Risa/Asir で練習してから, C や Java を覚えると習得が早いであろう.

[2] 矢川元基, パソコンでみる流れの科学. chap4 が cavity flow のシミュレータ. (Navier-Stokes) 数値流体力学入門. Genki Yagawa, Blue Backs, ブルーバックス.

[3] Masayasu Mimura, Asymptotic behaviors of a parabolic system related to a planktonic prey and predator model. SIAM J. on Applied Mathematics, 1979.

[4] 河合, 竹井, 特異摂動の代数解析学, 岩波.

第4章　グレブナ基底と代数方程式系

4.1　1変数の方程式

Todo: maxima の solve による解の公式
Todo: `pari(roots,F)` による近似解の求解
Todo: Maxima で newton を使う.
Todo: Giant によるガロア群の計算.

4.2　Risa/Asir で書く GCD 計算とその応用

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.

4.2.1　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 4.1 `+,-,*,/,^` 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 4.2 `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 4.3

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

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 4.4 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 4.5 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]
```

4.2.2 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 4.6

```
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 4.7 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.

4.2.3　情報科学からの準備

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];                    A, B が結合されて, C には
[11] B = [3,4];                    [1,2,3,4] がはいる.
[12] C = append(A,B);               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 4.8 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
```

4.2.4 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.

4.2.5　単項イデアルと 1 変数連立代数方程式の解法

“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

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

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 4.1 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 4.2 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 4.3 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.

4.2.6 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 4.4 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.

問題 4.5 $\mathbf{Q}[x]$ での 1 次不定方程式を再帰を用いて解くプログラムを書きなさい.

再帰をもちいると驚くほど短いプログラムで一次不定方程式を解くプログラムを書ける. 再帰の考え方は計算機特有の考え方である. この機会に再帰の考え方をマスターしておこう.

4.3 グレブナ基底

4.3.1 initial monomial, initial term など数学的な記号の解説

Todo: グレブナ基底の理論のひととおりはノートから. モノミアルイデアル. 順序. 記号. イデアルと方程式系. 基本性質. いまだ書く時間なし. (証明のノートも)

4.3.2 Initial term の取り出し

この節はイデアルの話題の続きである. 前の節で議論した, GCD を計算するアルゴリズムを用いると, 1 変数の多項式環の場合, 多項式 f が イデアル I に入っているかどうかを, 計算機を用いて確かめることが可能である. イデアル I が, 多項式 p と q で生成されているとしよう. このとき h を p と q の GCD とすると, f がイデアル I に入っているための必要十分条件は, $\texttt{division}(f,h)[1]$ が 0 を戻すことである. つまり, f が h で割り切れればよい. これは $I=\langle h\rangle$ であることから明らかであろう. まとめると,

1. 入力 p, q に対して, 前節の互除法のアルゴリズムによりイデアルの生成元 h を求める.
2. $f\in I$ かどうか判定したい多項式 f の $\texttt{division}(f,h)$ を計算して, その第 2 成分 (割算した余り) が 0 なら, $f\in I$ と答え, 0 でないなら $f\notin I$ と答える.

この判定アルゴリズムをイデアルメンバシップアルゴリズム (ideal membership algorithm) とよぶこともある.

2 変数以上のイデアルは, 単項生成とは限らない. たとえば,

$$I=\langle x^2,xy,y^2\rangle=\mathbf{Q}[x,y]x^2+\mathbf{Q}[x,y]xy+\mathbf{Q}[x,y]y^2$$

を考えると $I=\langle h\rangle$ となるような h は存在しない. なぜなら, そのような h があれば, x^2, xy, y^2 を同時に割り切らないといけないが, そのような h は定数のみである. h が定数だと I は $\mathbf{Q}[x,y]$ に等しくなってしまうが, $\langle x^2,xy,y^2\rangle$ はたとえば 1 を含まない (証明せよ). よって I は単項生成でない.

このアルゴリズムは多変数多項式に一般化できる. この一般化されたアルゴリズムの中心部分が Buchberger アルゴリズムである. Buchberger アルゴリズムは, イデアルに入っているかどうかの判定だけでなく, 代数方程式系を解いたり, 環論や代数幾何や多面体や微分方程式論などにでてくるさまざまな数学的対象物を構成するための基礎となっている重要なアルゴリズムである.

この節では Buchberger アルゴリズム自体の解説は優れた入門書がたくさんあるので (たとえば [1], [**?**] など), それらを参照してもらうことにして, Buchberger アルゴリズムの中で重要な役割をはたす割算アルゴリズム (または reduction アルゴリズム) について計算代数システムの内部構造の立場からくわしく考察する. それから Buchberger アルゴリズムとグレブナ基底を簡潔に解説して, イデアルメンバシップアルゴリズムを説明する. 最後にグレブナ基底の別の応用である連立方程式の求解について簡単にふれる.

イデアルメンバシップアルゴリズムを多変数へ拡張するには 1 変数の場合と同様にまず initial term を計算する関数を用意しないといけない.

注意すべきは, 2 変数以上の場合, さまざまな順序を考えることが可能となることである. 以下 3 変数多項式環 $\mathbf{Q}[x,y,z]$ で議論する. たとえば $z\succ_{lex}y\succ_{lex}x$ なる辞書式順序 (lexicographic order) は次のように定義する.

$$\begin{aligned}&Px^ay^bz^c\succ_{lex}P'x^{a'}y^{b'}z^{c'}\\ \Leftrightarrow\quad&(c-c',b-b',a-a')\text{ の最初の }0\text{ でない成分が正}\end{aligned}$$

つまり, まず z の巾で大小を比較して, z の巾が同じなら次に y の巾で比較し, y の巾も同じなら, x の巾で比較せよという順序である. なおここで P, P' は係数であり, 係数は順序の比較に影響しない.

別の順序を考えることも可能である. たとえば, $w = (w_1, w_2, w_3) \in \mathbf{R}^3$ を $w_i \geq 0$ であるようなベクトルとして,

$$\begin{aligned} & Px^a y^b z^c \succ_w P' x^{a'} y^{b'} z^{c'} \\ \Leftrightarrow \quad & d = (a-a')w_1 + (b-b')w_2 + (c-c')w_3 > 0 \\ & \text{または } (d = 0 \text{ かつ } Px^a y^b z^c \succ_{lex} P' x^{a'} y^{b'} z^{c'}) \end{aligned}$$

と順序 $\succ_w$ を定義する. これを重みベクトル w を辞書式順序で細分して得られた順序という.

あたえられた順序 $\succ$ ($\succ$ は $\succ_{lex}$ か $\succ_w$) と多項式 f に対して, 多項式 f の中の一番大きいモノミアル (単項式) をその多項式の initial term といい $\mathrm{in}_\succ(f)$ と書く.

次の関数 `in(F)` および `in2(F)` はそれぞれ, $z > y > x$ および $x > y > z$ なる辞書式順序 (lexicographic order) で initial term をとりだす関数である.

```
def in(F) {
   D = deg(F,z);
   F = coef(F,D,z);
   T = z^D;
   D = deg(F,y);
   F = coef(F,D,y);
   T = T*y^D;
   D = deg(F,x);
   F = coef(F,D,x);
   T = T*x^D;
   return [F*T,F,T];
}
```

```
def in2(F) {
   D = deg(F,x);
   F = coef(F,D,x);
   T = x^D;
   D = deg(F,y);
   F = coef(F,D,y);
   T = T*y^D;
   D = deg(F,z);
   F = coef(F,D,z);
   T = T*z^D;
   return [F*T,F,T];
}
```

結果はリストで戻しており, リストの先頭要素が多項式 `F` の initial term, リストの 2 番目の要素が initial term の係数, リストの 3 番目の要素が initial term の係数を除いたモノミアル部である.

4.3.3 多項式の内部表現と initial term の取り出しの計算効率

前の節の二つの関数 `in(F)` および `in2(F)` には違いがないと思うかもしれないが, 次の関数で実行時間をはかるとおおきな違いがある. between `in(F)` and `in2(F)`.

```
def test1() {
  F = (x+y+z)^4;
  for (I=0; I<1000; I++) G=in(F);
  return(G);
}
def test2() {
  F = (x+y+z)^4;
  for (I=0; I<1000; I++) G=in2(F);
  return(G);
}
```

```
[764] cputime(1);
0
0sec(0.000193sec)
[765] test2();
[z^4,1,z^4]
0.06sec + gc : 0.08sec(0.1885sec)
[766] test1();
[x^4,1,x^4]
0.2sec + gc : 0.08sec(0.401sec)
```

test1 と test2 では, 3 倍ちかくの実行時間の差がある. これはどういった理由からであろうか?

理由を理解するには計算機のメモリに多項式がどのように格納されているかを考えてみる必要がある.

Asir ではメモリにたとえば多項式 $5x^2$ が格納されるときには, つぎのようなセルとして格納されている.

主変数名	x
主変数にかんする次数	2
係数	5
次のモノミアルのアドレス	なし

このようなセルの和でモノミアルの和を表現する. たとえば, $5x^2+x$ であれば, 次のようなセルの集まりとして格納されている.

主変数名	x
主変数にかんする次数	2
係数	5
次のモノミアルのアドレス	AAAA

AAAA:

主変数名	x
主変数にかんする次数	1
係数	1
次のモノミアルのアドレス	なし

$5x^3+5xz^2+xz$ を asir に入力すると,

```
[755] 5*x^3+5*x*z^2+x*z;
5*x^3+(5*z^2+z)*x
```

と x の多項式として表示されるであろう. メモリ内部には, 上で説明したしたように, x に関するモノミアルを表現するセルのリストとして表現される. 上の説明との違いは, x の係数が, z の多項式でありそれが z に関するモノミアルを表現するセルのリストとして表現されているという再帰的構造を持つことである. 図示すると次のようになる.

主変数名	x
主変数にかんする次数	2
係数	5
次のモノミアルのアドレス	AAAA

AAAA:

主変数名	x
主変数にかんする次数	1
係数のアドレス	BBBB
次のモノミアルのアドレス	なし

BBBB:

主変数名	z
主変数にかんする次数	2
係数	5
次のモノミアルのアドレス	CCCC

CCCC:

主変数名	z
主変数にかんする次数	1
係数	1
次のモノミアルのアドレス	なし

この表現をみれば, x についての最高次の項を取り出す操作は 1 ステップで終るのにたいし, z についての最高次の項をとり出す操作は, x の各次数の係数の z についての最高次の項を取り出してから, その中でさらに一番大きいものをとりださねばならないことが分かるであろう. こちらの方がはるかに複雑であるので, 時間がかかるのである.

C 言語を知ってる読者は 図 4.1 のような構造体を用いて, このような多項式の足し算および `deg` 関数 `coef` 関数の実装を試みると理解が深まるであろう.

```
union object_body {
  struct polynomial_cell f;
  int c;
};
struct object {
  int tag;
  union object_body obj;
};
struct polynomial_cell {
  char *v;  /* variable name */
  int   e;  /* degree */
  struct object *coef;  /* coefficients */
  struct polynomial_cell *next;
};
```

図 4.1: 多項式を表現する構造体

Asir では, このような表現による多項式を再帰表現多項式とよぶ. この型のデータに対して type 関数の戻す値は 2 である.

さて, あたえられた順序に対して, このような多項式の表現法では, initial term を取り出す効率が順序によりかわってしまうし, 効率の点からは最適な表現とはいえない. initial term を取り出すには, 多項式は多変数のモノミアルのあたえらた順序に関するリストとして表現するのがよいであろう. この表現方法を, asir では, 分散表現多項式とよんでいる. たとえば, 多項式 $5x^3 + 5xz^2 + xz$ を $x > y > z$ なる lexicographic order での分散表現多項式としてメモリに格納するときにはつぎのようなリストとして表現する.

x, y, z についての次数	[3,0,0]
係数	5
次のモノミアルのアドレス	DDDD

DDDD :

z, y, x についての次数	[1,0,2]
係数	5
次のモノミアルのアドレス	EEEE

EEEE :

z, y, x についての次数	[1,0,1]
係数	1
次のモノミアルのアドレス	なし

このように表現しておくことにより, initial term の取り出しはリストの先頭項を取り出すだけの操作ですむことになる.

次の関数では, initial term の取り出し 1000 回のテストを, 分散表現多項式に対しておこなっている. dp_ptod(F) は F を分散表現多項式に変換する関数である. 詳しくはマニュアルを参照.

```
def test1_dp() {
  F = (x+y+z)^4;
  dp_ord(2);
  F = dp_ptod(F,[x,y,z]);
  /* print(F); */
  for (I=0; I<1000; I++) G=dp_hm(F);
  return(G);
}
```

とりだしにかかる時間は以下のとおりである.

```
[778] test1_dp();
0.01sec(0.01113sec)
```

4.4　割算アルゴリズム

1 変数多項式に対する割算アルゴリズムは, initial term に注目することにより多変数に拡張できる. すなわち, 多項式 f のある項 t が, 多項式 g の initial term で割り切れるとき, 適当な単項式 m を選んで $f - mg$ に t が現れないようにできる. この操作を reduction と呼ぶ. reduction を繰り返して, それ以上 reduction できない多項式を得たとき, それを割算による剰余とする.

3 変数 x, y, z の場合のプログラム.

```
def multi_degree(F) {
  F = in(F);
  T = F[2];
  return([deg(T,x),deg(T,y),
          deg(T,z),F[1]]);
}
```

$x > y > z$ なる辞書式順序での initial term を取り出す
各変数の次数および, initial term の係数をリストにして返す

F が G で reducible でないとき (reduction できないとき) 0 を戻す. F が G で reducible のとき (reduction できるとき) M G が F の initial term に等しいような, M を戻す. 別の言い方をすると, in(F) = in(M) in(G) となるモノミアル M を戻す.

```
def division(F,G) {
  Q = 0; R = F;
  D = is_reducible(R,G);
  while (type(D) != 0) {
    Q = Q+D;
    R = R-D*G;
    D = is_reducible(R,G);
  }
  return [Q,R];
}
```

G の initial term が F の initial term を割り切る限り, F から G の単項式倍を引いて F initial term を消去する.
この関数の出力は, initial term が G の initial term で割り切れないことは保証される.

いくつかの元を含む集合 G による剰余計算も可能である. どの項を reduction するかに任意性があるため, 一般には結果は 1 通りとは限らないことに注意しておく.

プログラム

```
def reduction(F,G)
{
  Rem = 0;
  while ( F ) {
    for ( U = 0, L = G; L != [];
          L = cdr(L) ) {
      Red = car(L);
      Mono = is_reducible(F,Red);
      if ( Mono != 0 ) {
        U = F-Mono*Red;
        if ( !U )
          return Rem;
        break;
      }
    }
    if ( U )
      F = U;
    else {
      H = in(F);
      Rem += H[0];
      F -= H[0];
    }
  }
  return Rem;
}
```

G は多項式のリストである. この関数では, F の先頭項が, G のどの要素の initial term によっても割り切れない場合に, F から先頭項をとりはずし, Rem に追加される. この関数の出力は, どの項も, G のどの要素の initial term によっても割り切れないような多項式である.

実行例

```
[216] reduction(x^2+y^2+z^2,
      [x-y*z,y-z*x,z-x*y]);
2*x^2+y^2
[217] reduction(x^2+y^2+z^2,
      [z-x*y,y-z*x,x-y*z]);
(y^2+1)*x^2+y^2
```

この例では, G の要素の並び方により, 結果が異なっている.

4.4.1 グレブナ基底

$\prec$ を 前節で考えた順序 $\prec_{lex}$ か $\prec_w$ とする. $f_1, \ldots, f_p$ を n 変数多項式環 $S_n = \mathbf{Q}[x_1, \ldots, x_n]$ に属する多項式としよう. S_n の部分集合

$$I = \langle f_1, \ldots, f_p \rangle := S_n f_1 + \cdots + S_n f_p$$

を考える. I はイデアルである. 多項式の有限集合

$$G = \{g_1, \ldots, g_m\}$$

が次の二つの条件を満たすときイデアル I の順序 $\prec$ に関するグレブナ基底であるという.

1. $I = \langle g_1, \ldots, g_m \rangle$ ($g_1, \ldots, g_m$ が I を生成.)
2. $\mathrm{in}_\prec(I) := \langle \mathrm{in}_\prec(f) \,|\, f \in I \rangle$ が $\langle \mathrm{in}_\prec(g_1), \ldots, \mathrm{in}_\prec(g_m) \rangle$ に等しい.

例として, 1 変数多項式で 順序 $1 < x < x^2 < \cdots$ の場合を考える. $f_1, \ldots, f_s$ の GCD を h とすると, $G = \{h\}$ はグレブナ基底である. グレブナ基底は余分な元をふくんでいてよくて, たとえば, $G = \{f_1, \ldots, f_p, h\}$ も I のグレブナ基底となる.
問題: ヒルベルトの基底定理を用いてグレブナ基底の存在を証明せよ.

グレブナ基底の構成をおこなうアルゴリズムが Buchberger アルゴリズムである. まず, S 多項式の計算について簡単に説明する. 二つの多項式 f, g に対し, f, g の initial term をそれぞれ $c_f t_f$, $c_g t_g$ とするとき, $\{f, g\}$ の S 多項式は

$$\mathrm{Spoly}(f, g) = \frac{\mathrm{LCM}(t_f, t_g)}{c_f t_f} f - \frac{\mathrm{LCM}(t_f, t_g)}{c_g t_g} g$$

で定義される. ここで c_f は数, t_f は係数 1 のモノミアル, LCM の意味は下のプログラムから理解頂きたい.

```
def spolynomial(F,G)
{
  DF = multi_degree(F);
  DG = multi_degree(G);
  Mx = DF[0]>DG[0]?DF[0]:DG[0];
  My = DF[1]>DG[1]?DF[1]:DG[1];
  Mz = DF[2]>DG[2]?DF[2]:DG[2];
  return x^(Mx-DF[0])*y^(My-DF[1])*z^(Mz-DF[2])*F/DF[3]
        -x^(Mx-DG[0])*y^(My-DG[1])*z^(Mz-DG[2])*G/DG[3];
}
```

このプログラムで `A?B:C` なる表現があるが, これは, `A` が 0 なら `C` を戻し, `A` が 0 でないならば, `B` を戻す.

定理 4.1 *$I = \langle G \rangle$ とするとき, G が I のグレブナ基底であることと, G から作られる S 多項式の* `reduction` *による剰余がすべて 0 になることとは同値である.*

この定理の証明は, 例えば [1, 2 章 6 節 定理 6] にある. これよりただちに次の Buchberger アルゴリズムを得る.

1. $G = F$, $D = \{G$ から作られるペア全体 $\}$ とする. (初期化)
2. D が空なら G がグレブナ基底.
3. D から 1 つペア s を取り除き, s の S 多項式を G による剰余ををを r とする.
4. もし r が 0 でなければ, r と G から作られるペアを D に追加する. さらに, r を G に追加する. (この時点で s の S 多項式の G による剰余は 0 となる.)
5. 2. に戻る.

問題 4.6 このアルゴリズムは有限時間内に停止することを証明せよ. (ヒント: G の各要素の initial term の生成するイデアルを上のアルゴリズムの 2 で考えよ. 停止しないとすると, イデアルの真の無限増大列が作れる.)

グレブナ基底の応用はいろいろなものがあるが, たとえばグレブナ基底により, ある多項式がイデアルに属するかどうかが割算により判定できる.

定理 4.2 (イデアルメンバシップ) G をイデアル I のグレブナ基底とするとき, $f \in I$ であることと, f を G で reduction した剰余が 0 になることは同値である.

イデアルの元を, そのイデアルの基底で reduction した剰余もやはり同じイデアルの元になる. これとグレブナ基底の定義により定理が成り立つことがわかる (この定理の詳しい証明については [1, 2 章 6 節 系 2] を参照).

さらに, グレブナ基底の重要な性質として, どのように割算を行なっても結果が一意的である, ということがある. これも, 上の定理によりすぐにわかるであろう.

3 変数の場合の Buchberger アルゴリズムを実行するプログラムを書いてみる.

プログラム

```
def buchberger(F)
{
  N = length(F);
  Pairs = [];
  for ( I = N-1; I >= 0; I-- )
    for ( J = N-1; J > I; J-- )
      Pairs = cons([I,J],Pairs);
  G = F;
  while ( Pairs != [] ) {
    P = car(Pairs);
    Pairs = cdr(Pairs);
    Sp = spolynomial(G[P[0]],G[P[1]]);
    Rem = reduction(Sp,G);
    if ( Rem ) {
      G = append(G,[Rem]);
      for ( I = 0; I < N; I++ )
        Pairs = cons([I,N],Pairs);
      N++;
    }
  }
  return G;
}
```

Buchberger アルゴリズムは, イデアルの基底 `G` に属する二つの多項式からつくられる S 多項式に対し, `reduction` 関数により剰余を計算して, `G` に追加していく.

実行例

```
[218] F=[x-y*z,y-z*x, z-x*y];
[219] G=buchberger(F);
[x-z*y,-z*x+y,-y*x+z,-x^2+y^2,
 -y*x^3+y*x,-x^5+x^3,-x^4+x^2,
 x^3-x,-y*x^2+y]
[220] reduction(x^2+y^2+z^2,G);
3*x^2
[221] reduction(x^2+y^2+z^2,
                reverse(G));
3*x^2
```

この実行例は, 前節で例に用いたイデアルのグレブナ基底を計算してみたものである. 順序を変えて reduction しても, 今度は同じ結果になっていることに注目してほしい.

上のプログラムは Buchberger アルゴリズムのもっともシンプルな形である. 残念ながら, このままでは, ごく簡単な例しか計算できない. 次のような点から種々の改良が行なわれており, それらを実装してはじめて実用的に使えるものが得られる.

1. `Pairs` から 1 つペアを選ぶ方法を指定する.
2. `Pairs` から, あらかじめ不必要なペアを取り除く.
3. 有理数係数の場合には, 分数が現れないような計算の工夫.

これらについては [?] でくわしく解説されている.

Asir に組み込まれている函数 `gr` は与えらた多項式と順序に対して, グレブナ基底を戻す. もちろん, 上に挙げたようなさまざまな改良が実装されている. (OpenXM 版でない Asir では, `load("gr");` でまずグレブナ基底用のライブラリをロードしておく必要がある.)

gr(イデアルの生成元，変数，順序の指定)

例:　gr([x^2+y^2-1, x*y-1],[y,x], 2);

順序は 0 が graded reverse lexicographic order, 1 が graded lexicographic order, 2 が lexicographic order である．行列を用いた一般の順序の指定も可能だが, それはマニュアルを見よ．上の例では, x^2+y^2-1 と $xy-1$ で生成されるイデアルの $y > x$ なる lexicographic order に関するグレブナ基底を戻す. 答えは,

$$\{\underline{-x^4} + x^2 - 1, \underline{y} + x^3 - x\}$$

である.

例 4.9 (大事)

1. $R = \mathbf{Q}[x, y]/\langle x^2+y^2-1, xy-1\rangle$ は, $\mathbf{Q}$ 上のベクトル空間であることを示し, 基底および次元を求めよ. 順序としては $y > x$ なる lexicographic order を用いよ.

2. $\mathbf{Z}/3\mathbf{Z}$ の乗法表のような形式で, R の乗法表を作れ. 自分で作成した割算函数を用いてみよ.

まず, ブレブナ基底の initial term が $-x^4$, y なので, これらで割れないモノミアルで係数 1 のものは,

$$1, x, x^2, x^3$$

である．これらが R の $\mathbf{Q}$ ベクトル空間としての基底になる．したがって, R の $\mathbf{Q}$ ベクトル空間としての次元は 4 である．つぎにここでは, Asir に組み込みの割算函数 p_true_nf を用いて乗法表を作成してみる.

```
[713] G = gr([x^2+y^2-1,x*y-1],[y,x],2)$
[714] G;
[-x^4+x^2-1,y+x^3-x]
[715] p_true_nf(x^4,G,[y,x],2);
[-x^2+1,-1]
[716] p_true_nf(x^5,G,[y,x],2);
[-x^3+x,-1]
[717] p_true_nf(x^6,G,[y,x],2);
[-1,1]
```

以上の出力より, R での次の乗法表を得る.

	1	x	x^2	x^3
1	1	x	x^2	x^3
x		x^2	x^3	x^2-1
x^2			x^2-1	x^3-x
x^3				-1

掛け算は可換なので, 下半分は省略してある.

例 4.10 上で書いた buchberger と, Asir の gr の速度比較をしてみよう.

```
[284] F = [-3*x^3+4*y^2+(-2*z-3)*y+3*z^2,(-8*y-4)*x+(2*z+3)*y,
           -2*x^2-3*x-2*y^2+2*z*y-z^2]$
[285] V = [x,y,z]$
[286] cputime(1)$
2.3e-05sec(1.895e-05sec)
[287] buchberger(F)$
10.38sec + gc : 0.6406sec(11.41sec)
[288] gr(F,V,2)$
0.05449sec(0.05687sec)
```

この例では gr が圧倒的に高速である. しかし, 入力をほんの少し変更すると次のようなことが起こる.

```
[289] F = [-3*x^3+4*y^2+(-2*z-3)*y+3*z^2,(-8*y-4)*x^2+(2*z+3)*y,
           -2*x^2-3*x-2*y^2+2*z*y-z^2]$
8.7e-05sec(7.892e-05sec)
[290] buchberger(F)$
49.02sec + gc : 3.573sec(53.4sec)
[291]  gr(F,V,2)$
163.1sec + gc : 4.694sec(168.2sec)
[292]  hgr(F,V,2)$
0.02296sec(0.02374sec)
```

これでわかるように, gr に実装されている改良も, あらゆる入力に対して有効なわけではなく, かえって悪影響を及ぼす場合もある. hgr というのは, gr にある前処理, 後処理を付け加えたもので, おおむね安定だが, やはり遅くなる場合もある. 計算効率の点では, グレブナ基底計算はまだ発展途上であり, 改良や新しいアルゴリズムも発表され続けている. 200310 以降の Asir には nd_ から始まる新しいグレブナ基底計算関数がテスト実装されている;
nd_gr(多項式リスト，変数リスト，標数，順序). 有限体上の計算の場合 gr より 数倍から数十倍高速である.

```
[1039] F = [-3*x^3+4*y^2+(-2*z-3)*y+3*z^2,(-8*y-4)*x^2+(2*z+3)*y,
            -2*x^2-3*x-2*y^2+2*z*y-z^2]$
[1040] nd_gr(F,[x,y,z],13,2);
ndv_alloc=4896
[z^14+4*z^13+2*z^12+7*z^11+ ....  ]
```

なお標数 0, 辞書式順序での計算は係数膨張を招きやすいので, 直接計算をする場合には nd_gr_trace(多項式リスト, 変数リスト,1,1,2) (斉次化つきトレースアルゴリズム) を使うほうが安全である.」

4.5 グレブナ基底と多変数連立代数方程式系の解法

多変数の連立方程式系

$$f_1(X) = \cdots = f_m(X) = 0 \tag{4.1}$$

($X = (x_1, \ldots, x_n)$, $f_i(X) \in \mathbf{Q}[X]$) を解く場合にも, イデアル

$$I = \langle f_1, \ldots, f_m \rangle = \{g_1(X)f_1(X) + \cdots + g_m(X)f_m(X) \mid g_i(X) \in \mathbf{Q}[X]\}$$

を考えることが有効である. 1 変数の場合とちがい, I は一般には単項イデアルではないが, グレブナ基底を計算することにより, 解を求めやすい形に変形することができる.

$\mathbf{Q}[X]/I$ の $\mathbf{Q}$ 上のベクトル空間としての次元 m が有限である場合, I を 0 次元イデアル とよぶ. 根の多重度を適切に定義してやると m は連立方程式系 (4.1) の複素解の個数に一致することが知られている ([1, 5 章 2 節 命題 3, 3 節 命題 8] に多重度のない場合の証明がある).

次の定理は 0 次元イデアルの定義とメンバシップアルゴリズムアルゴリズムを与えている定理 4.1 より容易に証明できる ([1, 5 章 3 節 定理 6] も参照).

定理 4.3 次は同値である.

1. I は 0 次元イデアルである.
2. I の, ある項順序に関する I のグレブナ基底が, 各変数 x_i に対して, initial term が x_i のべきであるような元を含む.
3. I の, 任意の項順序に関する I のグレブナ基底が, 各変数 x_i に対して, initial term が x_i のべきであるような元を含む.

この定理より, 解が有限個かどうかの判定が (任意の項順序の) グレブナ基底により判定できる.

例 4.11 $I = \langle x - yz, y - zx, z - xy\rangle$ の $x > y > z$ なる辞書式順序に関するグレブナ基底は $\{\underline{z^3} - z, \underline{-z^2y} + y, \underline{-y^2} + z^2, \underline{x} - zy\}$ である (冗長な要素は省いてある). よって定理より I は 0 次元イデアルである.

問題 4.7 与えられたイデアルの 0 次元性を判定する関数を書け.

また, この定理を, 辞書式順序の場合に適用すると, 次を得る.

定理 4.4 I を 0 次元イデアルとし, G を $x_1 > x_2 > \cdots > x_n$ なる辞書式順序に関する I のグレブナ基底とすると, G は

$$\begin{aligned}
g_1(x_1, x_2, \ldots, x_n) &= x_1^{d_1} + g_{1,d_1-1}(x_2, \ldots, x_n)x_1^{d_1-1} + \cdots + g_{1,0}(x_2, \ldots, x_n)\\
g_2(x_2, \ldots, x_n) &= x_2^{d_2} + g_{2,d_2-1}(x_3, \ldots, x_n)x_2^{d_2-1} + \cdots + g_{2,0}(x_3, \ldots, x_n)\\
&\cdots\\
g_{n-1}(x_{n-1}, x_n) &= x_{n-1}^{d_{n-1}} + g_{n-1,d_{n-1}-1}(x_n)x_{n-1}^{d_{n-1}-1} + \cdots + g_{n-1,0}(x_n)\\
g_n(x_n) &= x_n^{d_n} + g_{n,d_n-1}x_n^{d_n-1} + \cdots + g_{n,0}
\end{aligned}$$

という形の元を含む.

この定理により, まず 1 変数多項式 $g_n(x_n)$ の根を求め, それを $g_n(x_{n-1}, x_n)$ に代入すれば, x_{n-1} に関する 1 変数多項式を得る. その根を求め $g_{n-2}(x_{n-2}, x_{n-1}, x_n)$ に代入して, と, 次々に 1 変数多項式の根を求めることで, もとの方程式系の解の候補を求めることができる. (本当の解かどうかは, 他のすべて多項式の零点になっているかどうかをチェックしなければならない. 根を近似で求めてあると, このチェックは容易ではない.)

例 4.12 例 4.11 のイデアル I の零点を求めよう. まず, $z^3 - z = 0$ より $z = -1, 0, 1$.

1. $z = -1$

   $-y^2 + z^2 = 0$ より $y = -1, 1$. このとき $(x, y, z) = (-1, 1, -1), (1, -1, -1)$.

2. $z = 0$

   $-y^2 + z^2 = 0$ より $y = 0$. このとき $(x, y, z) = (0, 0, 0)$.

3. $z = 1$

   $-y^2 + z^2 = 0$ より $y = -1, 1$. このとき $(x, y, z) = (1, 1, 1), (-1, -1, 1)$.

例 4.13 n 変数の多項式を n 本ランダム生成し, それらで生成されるイデアルの辞書式順序グレブナ基底を求めて, どういう形をしているか観察せよ.

この例題の実験を行なう場合, n をあまり大きくすると計算できない. せいぜい 4, 5 程度にする. また, 次数も 2, 3 次程度にしておくこと. 辞書式順序グレブナ基底は, いきなり `gr` などで計算しても大抵ムリなので, Asir マニュアルをよく読んで, 上手な計算方法を学ぶこと. 実際にやってみればわかるように, 辞書式順序グレブナ基底は大変特徴的な形をしている場合が多い:

$$\begin{aligned} g_1(x_1, x_n) &= x_1 - h_1(x_n) \\ g_2(x_2, x_n) &= x_2 - h_2(x_n) \\ &\cdots \\ g_{n-1}(x_{n-1}, x_n) &= x_{n-1} - h_{n-1}(x_n) \\ g_n(x_n) &= h_n(x_n) \end{aligned}$$

これを shape base と呼ぶ. この形の場合には, $g_n(x_n) = 0$ の根を求めれば, 他の変数の値は代入により求まる.

例 4.14 与えられたイデアルが shape base を持つかどうか判定し, shape base を持つ場合に零点の近似値を計算する関数を書け.

1 変数多項式の根は, `pari(roots,Poly)` で計算できる. たとえば, `pari(roots,x^3-1)` で $x^3 - 1 = 0$ の (複素) 近似根を計算できる. ただし,

```
***   the PARI stack overflows !
 current stack size: 65536 (0.062 Mbytes)
 [hint] you can increase GP stack with allocatemem()
```

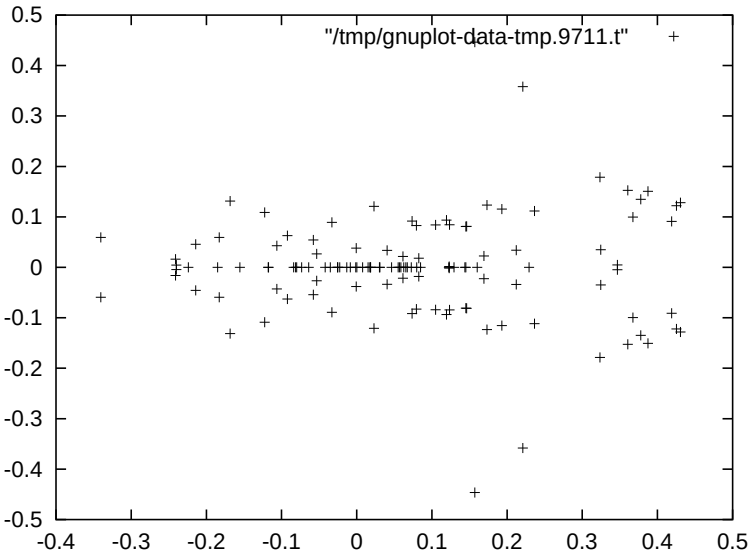
というようなエラーが出た場合には,

```
[295] pari(allocatemem,10^6)$
```

などを実行して, `pari` の使用できるメモリを増やすこと.

```
[826] load("katsura");
1
[831] katsura(7);
[u0+2*u7+2*u6+2*u5+2*u4+2*u3+2*u2+2*u1-1,
2*u6*u0+2*u1*u7-u6+2*u1*u5+2*u2*u4+u3^2,
2*u5*u0+2*u2*u7+2*u1*u6-u5+2*u1*u4+2*u2*u3,
2*u4*u0+2*u3*u7+2*u2*u6+2*u1*u5-u4+2*u1*u3+u2^2,
2*u3*u0+2*u4*u7+2*u3*u6+2*u2*u5+2*u1*u4-u3+2*u1*u2,
2*u2*u0+2*u5*u7+2*u4*u6+2*u3*u5+2*u2*u4+2*u1*u3-u2+u1^2,
2*u1*u0+2*u6*u7+2*u5*u6+2*u4*u5+2*u3*u4+2*u2*u3+2*u1*u2-u1,
u0^2-u0+2*u7^2+2*u6^2+2*u5^2+2*u4^2+2*u3^2+2*u2^2+2*u1^2]
```

Knoppix/Math で解く方程式: ギャラリ: $n = 7$ での katsura の方程式系. u0 から u7 が未知変数.



Knoppix/Math で解く方程式: ギャラリ: $n = 7$ での katsura の方程式系の解の第一成分 u0

Todo: もうすこし数値的な具体例. 特に, 方程式の差分化よりでてくる例. Todo: asir で解く例. Ask Noro.
課題: Buchberger の原論文の内容をまとめなさい.
参考: 記号の対称表.

4.6　微分方程式の差分化より得る連立代数方程式の解法例

4.6.1　Lorentz モデル

Lorentz モデルの平衡点の座標を計算してみよう.

```
A=10; B=20; C=266/100;
F=[-A*p1+A*p2, -p1*p3+B*p1-p2, p1*p2-C*p3];
V=[p1,p2,p3];
G=hgr(F,[p1,p2,p3],0);
print(["zero dim?", zero_dim(G,V,0)]);
dp_mbase(map(dp_ptod,G,V));
L=tolex(G,V,0,V);  /* tolex does not coeff like 2/3 */
P=minipoly(G,V,0,p3,t);

LG = tolex_gsl(G,V,0,V);
end$
```

4.6.2 １次元の反応拡散方程式

次は１次元の反応拡散方程式から導出した連立代数方程式 (3.6) をグレブナ基底で解析するプログラムである.

```
/* parter.rr の bug を fix cf. parger-bug.txt
   これは手計算でも 1 変数の方程式にできる.

   ここから gr のサンプルプログラムを生成.

   to_lex
   to_lex_gsl  (generalized shape lemma)
   minpoly 等.
*/

/* 項順序の設定 */
ord([u_1,a,d])$
pari(allocatemem,10^6)$
Glib_math_coordinate=1$

/*  Reaction diffusion equation.
    かめたか, 非線形微分方程式参照

    u_t = d u_xx + a(1-u) u,
    u(t,0)=u(t,1) = 0,  (Neumann でない. ==> Neumann 条件の場合も面白い.)

   Mimura, 微分方程式と差分方程式 も参照. [数値解析と非線形現象]

   u は密度 よって 0 <= u <= 1 を仮定すること多い.

   Parter, S.V., Mildly nonlinear elliptic partial differential equations and
  their numerical solutions, I, Numer. Math., 7 (1965), 338--366.
  [ 差分方程式の定常解. u_i = 0 と もう一つしか非負解が無いことを証明.
    Mimura, p.60

    a = xi_1,  xi_1= (4d/h^2)*sin^2(h*pi/2) が分岐点. a > xi_1 で面白い.

  ]
*/


/* d u_xx + a(1-u) u = 0 , [0,1] を N+1 等分した差分方程式系を戻す.
   u_1, ..., u_N についての N 変数方程式系.
*/
def parter(N) {
  H = 1/(N+1);
  Eq = [ ];
  V = [ ];
  for (I=N; I >= 1; I--) {
     U0 = util_v("u",[I-1]);
     U1 = util_v("u",[I]);
     U2 = util_v("u",[I+1]);
     if (I==1) U0 = 0;
     if (I==N) U2 = 0;
     Eq = cons( d*(U0 - 2*U1+U2) + H*H*a*(1-U1)*U1, Eq);
     V = cons(U1,V);
  }
  return [Eq,V];
}

/* Bifurcation value of a. */
def bif(N) {
   H = 1/(N+1);
   return (4*d/H^2)*sin(H*@pi/2)^2;
}

/* Return the shape base of Parter's system. */
def parter_shape(N) {
  Eqs = newvect(N+2);
  H = 1/(N+1);
  /* u_i = Eqs[i]  が関係式. Eqs[i] は u_1 で表現されている. */
  /* とくに u_{N+1} = 0 なので, Eqs[N+1] = 0 が u_1 の満たす方程式. */
  Eqs[1] = util_v("u",[1]);
  for (I=1; I<=N; I++) {
     U0 = util_v("u",[I-1]);
     U1 = util_v("u",[I]);
     if (I == 1) U0=0;
     U2 = 2*U1-U0 - (H^2/d)*a*U1*(1-U1);
     if (I != 1) U2 = subst(U2,U0,Eqs[I-1]);
     U2 = subst(U2,U1,Eqs[I]);
```

4.6.3 1 次元, 2 成分の反応拡散方程式

(3.8) に付随する代数方程式系の解析.

```
import("names.rr");
ord([u_1,a,d])$
pari(allocatemem,10^8)$
Glib_math_coordinate=1$

extern ShapeU$
extern ShapeV$
extern ShapeRule$
extern ShapeRule2$
extern ShapeRootU $
extern ShapeRootV $

ShapeRule=[]$
ShapeRule2=[[u_0,5],[u_1,5],[u_2,5],[u_3,5],[u_4,5],[u,5],
            [v_0,10],[v_1,10],[v_2,10],[v_3,10],[v_4,10],[v,10]]$


/*  cf. this04/misc-2004/A1/reaction-diff/parter.rr
*/


/* [0,1]/(N+1)
   d_1 u'' + (f_1(u) u - k u v) = 0,
   d_2 v'' + (-f_2(v) v + k  u v) = 0,
where
   f_1(z) = (1/9)*(35+16*z-z^2),
   f_2(z) = 1+(2/5)*z,
   0<= x <=5,
   k = 1, d_2 = 4.

   u'(0)=v'(0)=0, u'(5)=v'(5)=0.

   d_1=1/20
*/
def pps(N) {
  H = 5/(N+1);   D_1 = 1/20; D_2=4; K=1;
  Eq = [ ];
  V = [ ];
  for (I=N; I >= 1; I--) {
     U0 = util_v("u",[I-1]);     U1 = util_v("u",[I]);     U2 = util_v("u",[I+1]);
     V0 = util_v("v",[I-1]);     V1 = util_v("v",[I]);     V2 = util_v("v",[I+1]);
     if (I==1) {U0 = U1; V0=V1;}
     if (I==N) {U2 = U1; V2=V1;}
     Eq = cons( D_1*(U0 - 2*U1+U2) + H^2*((1/9)*(35+16*U1-U1^2)*U1-K*U1*V1),
                      Eq);
     Eq = cons(  D_2*(V0-2*V1+V2) + H^2*(-(1+(2/5)*V1)*V1                    +K*U1*V1),
                      Eq);
     V = cons(U1,V);   V=cons(V1,V);   /* in parter.rr H should be H^2. BUG. */
  }
  return [Eq,V];
}

def q_pps(N) {
  H = 5/(N+1);   D_1 = 1/20; D_2=4; K=1;
  Eq = [ ];
  V = [ ];
  for (I=N; I >= 1; I--) {
     U0 = util_v("u",[I-1]);     U1 = util_v("u",[I]);     U2 = util_v("u",[I+1]);
     V0 = util_v("v",[I-1]);     V1 = util_v("v",[I]);     V2 = util_v("v",[I+1]);
     if (I==1) {U0 = U1; V0=V1;}
     if (I==N) {U2 = U1; V2=V1;}
     E1 = ` d_1*(u0 - 2*u1+u2) + h^2*((1/9)*(35+16*u1-u1^2)*u1-k*u1*v1);
     E2 = ` d_2*(v0-2*v1+v2) + h^2*(-(1+(2/5)*v1)*v1          +k*u1*v1);
     Rule=[[d_1,D_1],[d_2,D_2],[h,H],[k,1],
           [u0,U0],[u1,U1],[u2,U2],
           [v0,V0],[v1,V1],[v2,V2]];
     E1 = base_replace(E1,Rule);
     E2 = base_replace(E2,Rule);
     print(print_input_form(E1));
     print(print_input_form(E2));
     Eq = cons( eval_quote(E1),    Eq);
     Eq = cons( eval_quote(E2),    Eq);
     V = cons(U1,V);   V=cons(V1,V);   /* in parter.rr H should be H^2. BUG. */
  }
  return [Eq,V];
```

Todo: fluid2d.rr (野呂のコメント待ち)

```
with(Groebner);
m:=9745285485474621232044564682715507257125628266320184805049402135023604972160190092249693410175276039149793804653
B:=[c+m,c]:
g:=Basis(B,tdeg(c));
```

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)
`http://www.math.kobe-u.ac.jp/HOME/taka/2007/asir-book-2007.pdf` が最新.

[5] 大阿久俊則: D-加群と計算代数, 朝倉書店.
微分方程式とグレブナ基底の入門書. グレブナ基底に関する証明付きの簡潔な解説もある.

[6] 野呂, 横山: グレブナー基底の計算 基礎篇 計算代数入門, 東京大学出版会, 2003. ISBN 4-13-061404-5.
グレブナー基底の基礎から代数方程式系の解法, イデアルの分解まで詳しく解説してある.

[7] 齊藤, 竹島, 平野: グレブナー基底の計算 実践篇 Risa/Asir で解く, 東京大学出版会, 2003. ISBN 4-13-061405-3.
一変数, 多変数の代数方程式 (系) のさまざまな解法を, Risa/Asir 上で実際に計算を行うためのプログラムを示しながら解説している.

第5章　グレブナ扇とTropical Geometry

この章で使うソフトウエア: R, 4ti2, C 言語, Asir, gfan, polymake, sm1

Todo: video://korea-tropical にプログラムのデモも加えて講義. この章は手書きノートによる.

5.1　単独方程式の Tropical zero set

Normal fan の詳しい導入は次回.

Todo: polymake の作者についてなど.

New(f) の計算には点の集合の凸包の計算が必要となる.

例 5.1 点の集合の凸包を計算し, その facet の法線ベクトルを計算せよ.

Polymake

polymake では, 入力点は原点をとおらない超平面の上においておく必要がある. 下の例では $x_1 = 1$ に点をのせてある. このファイル名を p-conv0.txt とする.

```
POINTS
1 0 0
1 2 0
1 0 3
1 1 1
```

```
polymake p-conv0.txt FACETS
```

と入力することにより, 次の出力ファイル (同じ名前の出力 p-dim.txt)

```
_application polytope
_version 2.2
_type RationalPolytope

POINTS
1 0 0
1 2 0
1 0 3
1 1 1

FACETS
0 1 0
3 -3/2 -1
0 0 1

AFFINE_HULL
```

を得る.

アルゴリズムについての覚え書き. 2 次元の凸包. Edelsbruner の coloring algorithm [**?**]. G.Ziegler の本 [2] V 表現と H 表現の変換. Fourier-Motzkin algorithm. CDD ライブラリ [3] およびアルゴリズムの説明.

Todo: 福田さんの論文.

次元の計算. このファイル名を `p-dim.txt` とする. Not found : @@code p-dim0.txt

```
polymake p-dim.txt DIM
```

と入力することにより, 次の出力ファイル

```
_application polytope
_version 2.2
_type RationalPolytope

POINTS
1 0 0 3
1 2 0 3
1 0 3 3
1 1 1 3

DIM
2
```

を得る.

Todo: volume 計算.

Todo: gfan の作者についてなど.

gfan では initial term を weight と exponent の内積との最大値で定義している. われわれのノートとは反対になっているので注意しよう.

例 5.2 われわれの running example である

$$I = (x_1 + x_2 + x_3)$$

の tropical zero set を gfan で計算してみよう.

入力プログラム (`t3.txt`)

```
{a+b+c}
```

で生成されるイデアルの tropical zero set を計算してみよう.
gfan への入力.

```
gfan_tropicalstartingcone <t3.txt | gfan_tropicaltraverse >t3-out.txt
```

出力

```
Ambient dimension: 3
Dimension of homogeneity space: 1
Dimension of tropical variety: 2
Is simplicial: true
Order of input symmetry group: 1
Number of maximal cones: 3
Modulo the homogeneity space:
{
(1,1,1)}
Rays:
{
0: (-1,0,0),
1: (1,1,0),
2: (0,-1,0)}
Printing index list for dimension 1 cones:
{
{0},

{1},

{2}}
Number of dimension 1 cones incident to each ray:
(1,1,1)
F-vector:
(3)
```

手計算のとおり 3 本の ray がある.

5.2 多面体の幾何とグレブナ扇

例 5.3 次の頂点を持つ多面体の normal fan の第一象限の部分を図示せよ.

$$(0,0,0),(2,0,0),(1,1,0),(0,3,0),(5,5,5),(0,0,1)$$

gfan

```
{1+a^2+a*b+b^3+a*b*c+c}
```

入力コマンドは

```
gfan <g-n.txt | gran_render >g-n.xfig
xfig g-n.xfig
```

例 5.4

$$I = (x_1 + x_2 + x_3, x_2 + 2x_3)$$

の Gröbner fan を計算せよ. 結果を図示せよ.

gfan

入力ファイル

```
{a+b+c,b+2*c}
```

図の生成コマンド (g-l2.xfig はなにも描かない. 何故か?)

```
gfan <g-l2.txt | gfan_render >g-l2.xfig
gfan <g-l2.txt | gfan_renderstaircase -m >g-l2-s.xfig
```

xfig での図を見るには, xfig コマンドを用いる.

少々美しい図を生成する例を紹介しておこう. (A.Jensen 自身による紹介プログラム)

```
{aab-c,bbc-a,cca-b}
{(1,2,0)}
```

```
gfan <sym.txt | gfan_render >sym.xfig
gfan <sym.txt | gfan_renderstaircase -m >sym-s.xfig
```

kan/sm1

Kan/sm1 は asir と同様国産のシステムである (実は筆者が著者). Kan/sm1 では/usr/local/OpenXM/lib/sm1/gfan.sm1 にある入力例を参考に入力を用意し, 実行する.

sm1 をスタートしたあと,

```
(gfan.sm1) run ;
cone.sample ;
```

(なお sm1 では 空白は token の分離に用いるので ； の前に空白の入力が必要) と入力すると, 1-simplex × 2-simplex に付随した affine toric variety

$$I = (x_{1i}x_{2j} - x_{1j}x_{2i} \,|\, 1 \le i < j \le 3)$$

の Gröbner fan を計算するデモを実行する.

専門家向け: cone.sample2 の方は y と $y-(x-1)^2$ に付随する Bernstein-Sato 多項式を計算するための微分作用素環のイデアルの local Gröbner fan の制限を計算する例である.

gfan と同様な次の問題をやってみよう.

例 5.5

$$I = (x_1 + x_2 + x_3, x_2 + 2x_3)$$

の Gröbner fan を計算せよ.

```
[(parse) (gfan.sm1) pushfile] extension
/mytry {
  %決まり文句
  cone.load.cohom
  /cone.ckmFlip  1 def


% 変数のリスト (形式 1)
% List of variables
/cone.vlist [(a) (b) (c)
             (Da) (Db) (Dc) (h)] def

% 変数のリスト (形式 2)
/cone.vv (a,b,c) def

% 3 変数なので 3 3 と書く.
/cone.parametrizeWeightSpace {
  3 3 parametrizeSmallFan
} def

% 以下は同次多項式の場合はきまり文句として書いておく.
/cone.type 2 def
/cone.local 0 def
/cone.w_start
  null
def
/cone.h0 1 def

% 入力はここへ書く.
/cone.input
  [
    (a + b + c)
    (b + 2 c)
  ]
def


% 以下は同次多項式の場合はきまり文句として書いておく.
/cone.DhH  0 def
/cone.gb {
  cone.gb_Dh
} def

cone.input message
%%%% Step 1.  Enumerating the Grobner Cones in a global ring.
%%%%   The result is stored in cone.fan
getGrobnerFan

%%%% If you want to print the output, then uncomment.
printGrobnerFan

} def
```

このファイル名を `s-l2.sm1` とするとき, sm1 をスタートしたあと,

```
(s-l2.sm1) run ;
mytry ;
```

で計算を遂行する.

D-加群についてのグレブナ fan を調べようとおもったら現在のところ kan/sm1 しか実装がない.

Todo: D-module の節で x2y2.sm1 を説明.

5.3 tropical geometry

gfan では initial term を weight と exponent の内積との最大値で定義している. われわれのノートとは反対になっているので注意しよう.

例 5.6

$$I = (x_1 + x_2 + x_3 + x_4, x_2 + 2x_3 + 3x_4)$$

の tropical zero set.

gfan への入力例を紹介しよう. たとえば次の入力プログラム (`t42.txt`)

```
{a+b+c+d,b+2*c+3*d}
```

で生成されるイデアルの tropical zero set を計算してみよう.
gfan への入力.

```
gfan_tropicalstartingcone <t42.txt | gfan_tropicaltraverse >t42-out.txt
```

出力

```
Ambient dimension: 4
Dimension of homogeneity space: 1
Dimension of tropical variety: 2
Is simplicial: true
Order of input symmetry group: 1
Number of maximal cones: 4
Modulo the homogeneity space:
{
(1,1,1,1)}
Rays:
{
0: (-1,0,0,0),
1: (0,-1,0,0),
2: (1,1,1,0),
3: (0,0,-1,0)}
Printing index list for dimension 1 cones:
{
{0},

{1},

{2},

{3}}
Number of dimension 1 cones incident to each ray:
(1,1,1,1)
F-vector:
(4)
```

Homogenity space が $(1,1,1,1)$ なので, $(1,1,1,1)$ のスカラー倍を足したものは同値な weight となる. $q \geq 0$, $p \in \mathbf{R}$ として

$$q((-1,0,0,0) + p * (1,1,1,1))$$
$$q((0,-1,0,0) + p * (1,1,1,1))$$
$$q((1,1,1,0) + p * (1,1,1,1))$$
$$q((0,0,-1,0) + p * (1,1,1,1))$$

の 4 つが tropical variety の最大次元の cone である.

たとえば $w = (1, 1, 1, 0)$ と時,

$$I = (x_1 + x_2 + x_3 + x_4, x_2 + 2x_3 + 3x_4)$$

の reduced Gröbner basis を $\succ_w$ で計算すると

$$G = [x_1 - x_3 - 2x_4, x_2 + 2x_3 + 3x_4]$$

となる.

$$\mathrm{in}_w(I) = (x_1 - x_3, x_2 + 2x_3)$$

はたしかにモノミアルを含んでいない.

なお現在の gfan では, tropical zero set の次元が 1 となる場合は abort と表示されて停止してしまう.

例 5.7 gfan のソースの `doc/example` にある `grassmann2_5` をためしてみよう.

```
{
bf-ah-ce,
bg-ai-de,
cg-aj-df,
ci-bj-dh,
fi-ej-gh
}
{
(3,6,8,9,0,1,2,4,5,7),
(0,6,5,4,3,2,1,9,8,7)
}
```

$(m+1) \times n$ 行列を考える. $i_1, \ldots, i_m, j_k$ 列をならべてつくった正方行列式を $p_{i_1 \cdots i_m j_k}$ と書くとき, Plücker の関係式は

$$\sum_{k=0}^{m+1} (-1)^k p_{i_1 \cdots i_m j_k} p_{j_0 \cdots \hat{j_k} \cdots j_{m+1}} = 0$$

と書ける.

いまの場合 $m = 1$, $n = 5$ である. ちなみに asir でこのような関係式の具体形を出力したい場合は,

```
plucker_relation([1],[2,3,4]);
plucker_relation([1],[2,3,5]);
plucker_relation([2],[3,4,5]);
```

などと入力する.

gfan への入力は Plucker 多項式である. $\binom{5}{2} = 10$ なので 10 変数ある. $10 - 3$ の 7 次元の variety である.

出力をみればわかるように, 同次な部分が 5 次元分あり, tropical zero set の実質的な部分は 2 次元である.

Grassmannian の tropical 幾何の理論的考察については, David Speyer, Bernd Sturmfels, The Tropical Grassmannian, http://arxiv.org/abs/math/0304218 (arXiv:math/0304218v3) [5] を参照. とくに $G_{2,n}$ の tropical zero set の simplicial complex としての構造は phylogenetic tree という graph に等しいということが証明してある.

問題: 筆者は証明を読んでいないので解読してレポートされたい. あわせて `gfan` で $G_{2,6}$ の場合のグラフを作成されよ.

超幾何関数へのノート: $(2, 5)$ Grassmann は Appell の 2 変数超幾何関数 F_1 に関係深い.

5.4 Toric variety

この節では toric variety のグレブナ基底とそれに付随する多面体の組み合わせ論を Macaulay や 4ti2 などを利用してしらべていくことにしよう.

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.

5.4.1 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 5.8 `+,-,*,/,^` 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 5.9 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 5.10 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 5.11 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 5.12 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
```

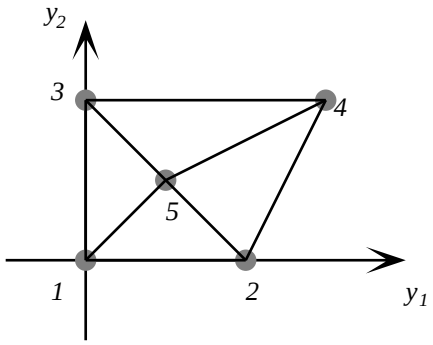
上の $1,4,4,1$ を betti 数とよぶ. betti 数に関する数学の定理... **まだ書いてない**

5.4.2 Macaulay2 を用いて多面体を三角形に分割する

Todo: 正則三角形分割と toric ideal の initial については黒板で説明.

課題: 論文 2007/knx/st.pdf [12] を読み証明を理解する.

ここでは図 5.4.2 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 triangulate.m2[1] 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．

[1]なお Macaulay 0.9.95 とそれ以前はパッケージの構成方法が大きく変わっている． 0.9.95 以降では上の入力で load("LLL.m2"); は必要ない． ちなみに LLL パッケージを利用場合は loadPackage("LLLBases"); コマンドで読み込む．

```
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 5.4.2 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.

問題 5.1 Voronoi 図は 各点 (x,y) の重みを $x^2 + y^2$ とした正則三角形分割の Dual である．Knoppix/math icms2006 DVD2 収録の DEpthLaunay による Voronoi 図を M2 による代数計算で構成してみよう．

数学関連事項はテキスト以外に板書でも説明. ビデオ: 2007-05-01 の学生向け解説.

5.4.3　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．

5.4.4　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
```

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

Toric 専用の Gröbner fan の計算システム

asir-contrib の tigers を見よ.

5.4.5 4ti2

4ti2 は toric ideal 専用のグレブナ基底計算プログラムである. (localization と saturation を上手に用いて高速化をはかってるようだ???)

例 5.13

入力プログラム

```
3 4
1 1 0 0
1 0 1 0
0 1 0 1

row column  3  4
```

実行

```
groebner 2x2.txt
```

すると, 2x2.txt.gro というファイル が自動生成される.

```
1 4
-1  1  1 -1
```

これの見方の説明.

-1 1 1 -1 は $x_1^1 x_4^1 - x_2^1 x_3^1$ に対応.

Todo: Et/4ti2 E(2,5) の例.

5.5 Feasible set の上の乱歩

5.5.1 Toric variety のグレブナ基底の応用

数学関連事項はテキスト以外に板書でも説明. ビデオ: 2007-5-1 にした random sampling の学生への説明

5.5.2 Codon の代数的統計学

Todo: この章は当分完成しないかも.

命題 5.1 $w(i \to k)$ を状態 i から状態 k への遷移確率とする. 次の条件

$$w(i \to k) \exp\left(-\frac{E_i}{k_B T}\right) = w(k \to i) \exp\left(-\frac{E_k}{k_B T}\right) \tag{5.1}$$

が成立していれば, この遷移確率で変化していく系は正準分布に近づいていく.

証明. このような系が多数あるとし, いま状態 i にある系が N_i 個であるとする. この多数の系が次の状態に同時に遷移する. この時 状態 i から k へ遷移するものの個数は $w(i \to k)N_i$ であり, 状態 k から i へ遷移するものの個数は $w(k \to i)N_k$ である. この和の差は

$$\begin{aligned} w(i \to k)N_i - w(k \to i)N_k &= N_i w(k \to i)\left(\frac{w(i \to k)}{w(k \to i)} - \frac{N_k}{N_i}\right) \\ &= N_i w(k \to i)\left(\frac{\exp\left(-\frac{E_k}{k_B T}\right)}{\exp\left(-\frac{E_i}{k_B T}\right)} - \frac{N_k}{N_i}\right) \end{aligned}$$

となる. N_k/N_i の比が想定値より小さいなら上の式の右辺は正となる. このとき左辺をみれば i から k へ遷移するものの方が, k から i へ遷移するものより多い. よって N_k/N_i の比は増える. N_k/N_i の比が想定値より大きい場合も同様である.

右辺が 0 に収束していくことになるが, この時 N_k/N_i の比は正準分布の比に等しい.

条件 (5.1) により一意的に $w(i \to k)$ がきまるわけではない. これの決め方には任意性がある. Metropolis 法では $E_k - E_i \leq 0$ の時 1, $E_k - E_i > 0$ の時

$$\exp\left(-\frac{E_k - E_i}{k_B T}\right)$$

と選ぶ. この条件は (5.1) を満たす.

この方法を応用して一次元の超幾何多項式の近似計算を試みよう. 我々が考える超幾何多項式は次の和である.

$$\Phi(x) = \sum_{Ak=\beta, k_i \geq 0} \frac{1}{k!} x^k, \quad k! = k_1! \cdots k_n! \tag{5.2}$$

ここで A は 行列 $A = (1, 2)$, $n = 2$ である. たとえば $\beta = 6$ だと

$$\Phi = \frac{x^0 y^3}{0!3!} + \frac{x^2 y^2}{2!2!} + \frac{x^4 y^1}{4!1!} + \frac{x^6 y^0}{6!0!}$$

となる. β の値を大きくするとこれは巨大な多項式となっていく. ここで $\Phi(x)/\Phi(1)$ の値を $x = 1$ の近傍で近似計算する問題を考えよう. 考え方としては, $\Phi(x)$ に出現する各項を Metropolise 法で

超幾何分布に従い, 生成する. すべての場合を生成するわけではない. したがって部分和にほかならない. 生成された指数の集合を E と書こう. 我々が計算するのは $Phi_E(x)/Phi_E(1)$ である. ここで

$$\Phi_E(x) = \sum_{k\in E, Ak=\beta, k_i\geq 0} \frac{x^k}{k!} \tag{5.3}$$

である. このような方法でどの程度の近似が計算できるのかは計算実験が必要である. **まだやってない**

同様に重み付の和

$$\Phi(c_k;x) = \sum_{Ak=\beta, k_i\geq 0} \frac{c_k x^k}{k!} \tag{5.4}$$

を考えて, $\Phi(c_k;x)/\Phi(x)$ の値を近似計算する問題も同様の手法で近似計算できると予想できる. 分母が正確に計算できて, 大きい数だとすると, $\Phi_E(c_k;x)/\Phi(x)$ の近似度はかなりよいはず. **まだやってない**

プログラム 5.1 頻度をグラフ表示する関数. `metro1.rr:histgram`. `metro1.rr:test_hist`

プログラム 5.2 逆関数法で正規分布を生成する関数. `metro1.rr:g1` `metro1.rr:test_g1`

何故中心部の分布が少ないのか?

プログラム 5.3 metropolis 法で正規分布を生成する関数. `metro1.rr:g2t` `metro1.rr:test_g2t`

とりあえず正規分布みたいな形を得た. やはり真ん中が引っ込んでるが.

Metropolis 法で正規分布を作ろう. このとき $\frac{-2 \text{ 以下のものの個数}}{\text{生成した乱数の個数}}$ は $\int_{-\infty}^{-2} e^{-x^2}dx / \int_{-\infty}^{+\infty} e^{-x^2}dx$ に近づくと予想できる. **まだためしていない**

注意: `mpi();` をみればわかるように, 積分値の計算に Metropolise 法は使えない.

ところで分母の積分値は $\sqrt{\pi}$ だが, この値を計算できるシステムはあるか?

```
Integrate[Exp[-x^2],{x,-Infty,Infty}]
```

Todo: tobe continued.

関連図書

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

[2] Günter M.Ziegler, *Lectures on Polytopes* (Graduate Texts in Mathematics 152, Springer-Verlag New York 1995)

[3] Komei Fukuda cdd Home Page (http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html)

[4] H.Edelsbrunner, *Algorithms in Combinatorial Geometry* (Springer-Verlag 1986)

[5] David Speyer, Bernd Sturmfels, The Tropical Grassmannian, `http://arxiv.org/abs/math/0304218` (arXiv:math/0304218v3)

[6] B.Sturmfels, *Gröbner basis and convex polytope*, 1995

[7] K.Fukuda, A.Jensen, R.Thomas, Computing Gröbner fans, `math.AC/0509544`

[8] T.Bogart, A.Jensen, D.Bayer, B.Sturmfels, R.Thomas, Computing tropical varieties, `math.AC/0507563`

[9] J.Richter-Gebert, B.Sturmfels, T.Theobald, First steps in tropical geometry, `math.AG/0306366`

[10] Grigory Mikhalkin, Tropical geometry and its applications, `math.AG/0601041`

[11] Walker, *Algebraic curves*, 1950.

[12] B.Sturmfels, Gröbner bases of toric varieties, Tohoku Mathematical Journal, **43**, (1991), 249–261.

第6章　代数方程式の数値解法

この節でとりあげるソフトウエア. phc, PHoM.

6.1　Newton法

Asir ドリルおよび 計算数学 2(2005) のノートを援用する.
Todo: いわゆる quasi-newton method と PDE の解法.

6.2　ホモトピー法による1変数代数方程式の求解

代数方程式

$$x^n + a_{n-1}x^{n-1} + \cdots a_1 x + a_0 = 0$$

の数値近似解を求める方法を考えよう. Newton 法は十分近くの解があれば解を求めることが可能だが全部の根を求めるにはどうすればよいのであろうか? ここでは変数が 2 変数以上の場合にも使える方法としてホモトピー法による解法を紹介しよう.

$$F(t,x) = x^n + ta_{n-1}x^{n-1} + \cdots ta_1 x + a_0 = 0$$

とおく. このとき $F(0,x) = x^n + a_0$ であり, $F(1,x) = 0$ は解きたい方程式である. 根 x は t の関数となる. $F(t,x(t))$ を t で微分しよう. すると,

$$x'nx^{n-1} + (a_{n-1}x^{n-1} + ta_{n-1}x'(n-1)x^{n-2}) + \cdots (a_1 x + ta_1 x') = 0$$

をえる. これは関数 $x(t)$ についての常微分方程式となる. ただし非線形である. この方程式は

$$x'(t) = -\frac{a_{n-1}x^{n-1} + \cdots + a_1 x}{nx^{n-1} + ta_{n-1}(n-1)x^{n-2} + \cdots + ta_1} \tag{6.1}$$

とかける. $t = 0$ での初期条件としては $x^n + a_n = 0$ の根をとる. この初期条件から出発して次のようなアルゴリズムで $t = 0$ から代数方程式系の解を $t = 1$ まで延長していく.

アルゴリズム 6.1 (1 変数代数方程式でのホモトピー法)

1. $t = a$ での値 $x(a)$ がわかっているとしたとき, 式 (6.1) を Chapter **??** の差分化したものを用いて $t = a + h$ (h は十分小さい量) での量 $x(a + h)$ の近似を求める.
2. Chapter **??** で説明した Newton 法を用いて $x(a + h)$ の値を根により近い値にする. 1 へ戻り繰り返す.

この方法をプログラムにして, $x^3 - x^2 + x - 8 = 0$ の数値解を求めてみよう.

```
extern Cc  $
Cc = 8 $

def homot() {
  Ans = [];
  /* The first solution */
  Xs = 2.0;
  Ans = append(Ans,[homot_aux(Xs)]);
  print("------------------------------");

  /* The second solution */
  Xs = (-1.0000000000000000000+1.732050807568877293 46*@i);
  Ans = append(Ans,[homot_aux(Xs)]);
  print("------------------------------");

  /* The third solution */
  Xs = (-1.0000000000000000000-1.732050807568877293 46*@i);
  Ans = append(Ans,[homot_aux(Xs)]);
  print("------------------------------");
  return Ans;
}

def homot_aux(Xs) {
   X0 = Xs;
   H = 0.1; T = 0.0;
   while (1) {
     X1 = X0+H*((X0^2-X0)/(3*X0^2-2*X0*T+T));
     T += H;
     print("T=",0); print(T,0);
     print(", old=",0); print(X1);
     X1=newton_aux(X1,T);
     print("new=",0); print(X1);
     E=T-1.0; if (E < 0.0) E = -E;
     if (E < 0.0001) break;
     X0 = X1;
   }
   return X1;
}

def newton_aux(A,T) {
  Epsilon = 0.0001;
  Q = A;
  P = Q+1.0;
  while (!( (Q-P > -Epsilon) &&
            (Q-P < Epsilon))) {
     P = Q;
     print(Q);
     Q = P-(P^3-P^2*T+P*T-Cc)/(3*P^2-2*P*T+P);
```

いまの場合 $F(x,t) = x^3 - x^2 * t + x * t - 8$ とおいている．$x(t)$ についての方程式は

$$\frac{dx}{dt} = (x^2 - x)/(3x^2 - 2xt + t)$$

である．関数 `homot()` をよびだして解を求める．`Xs` に $F(x,0) = x^3 - 8 = 0$ の解をセットする．関数 `homot_aux(Xs)` ではアルゴリズム 6.1 にしたがって解を延長していく．

```
X1 = X0+H*((X0^2-X0)/(3*X0^2-2*X0*T+T));
```

が差分化により得られた漸化式である．関数 `newton_aux(X1,T)` は $F(x,t) = 0, t =$ `T` の解を初期値 `X1` をスタートとして Newton 法を用いて求める．

問題 6.1 (1) このプログラムに $F(x,t) = 0$ の根を $t \in [0,1]$ の範囲で複素平面でプロットするような関数を加えよ．glib を使う．
(2) 数学の応用に自然に出現する代数方程式をホモトピー法で解きなさい．

最後にこの方法の欠点について考察しよう．微分方程式 (6.1) の分母 $nx^{n-1} + ta_{n-1}(n-1)x^{n-2} + \cdots + ta_1$ が 0 か 0 に十分近い数になると微分方程式の数値解の誤差が大きくなる．これは $F(x,t) = 0$ 重根を持つ場合に対応する．このような場合重根に向かうのを避けるか, 重根の周りでは巾級数等を用いて数値解析を続ければよい．しかし非線形方程式の特異点の解析は一般的にはむづかしいので, 解を満たす線形の微分方程式があるとより解析が容易になる．実はそのような線形の方程式が存在している．この問題については $\mathcal{A}$-超幾何関数と微分作用素環のアルゴリズムの節でまた考察しよう．

問題 6.2 上のような問題点が生じる $F(x,t) = 0$ の例を作りなさい．

6.3 Numerical Homotopy 法

Birk-Sturmfels の論文に従い解説する．Todo:
Todo: 反応拡散系等の例題も試す．
Todo: Eh?? phc のサンプル．

6.4 SOS による代数方程式の解法

ここでは J.Lasserre の論文 [1] “Global optimization with polynomials and the problem of moments” の紹介 (modulo semi-definite programming) およびこの論文を読みながら SOS を援用した代数方程式系の解法を説明し, また論文を読むのにどのように数学ソフトを使ってみたかの実例を紹介する.

$p(x)$ を n 変数 $x_1, \ldots, x_n$ の実数係数多項式, ($\mathbf{P}$ を n 変数の実数係数多項式の集合とする.?)

$\mathcal{P}(\mathbf{R}^n)$ を $\mathbf{R}^n$ の上の, 符号付き有限ボレル測度の集合とする. たとえば, $\delta(x_1 - a_1)dx_1$ を一点 a_1 にのみ台のある $\int_{\mathbf{R}} f(x_1)\delta(x_1 - a_1)dx_1 = f(a_1)$ なるデルタ測度とするとき, これは符号付き有限ボレル測度である. また a を $\mathbf{R}^n$ の点とするとき $\delta(x - a)dx = \delta(x_1 - a_1)dx_1 \cdots \delta(x_n - a_n)dx_n$ は符号付き有限ボレル測度である.

最適化問題

$$\mathcal{P} \mapsto p^* := \min_{\mu \in \mathcal{P}(\mathbf{R}^n)} \int p(x)\mu(dx) \tag{6.2}$$

と

$$\mathbf{P} \mapsto p^* := \min_{x \in \mathbf{R}^n} p(x) \tag{6.3}$$

を考えよう.

二つの問題の p^* の値は有限値として存在すれば一致する. なぜなら $p(x) \geq p^*$ であるので,

$$\int p(x)\mu(dx) \geq \int p^*\mu(dx) = p^* \int \mu(dx) = p^*$$

である. よって $\inf\mathcal{P} \geq p^*$. 一方 (6.3) の解を与える x を x^* とすれば, $\mu = \delta(x - x^*)dx$ ととれば, $\int p(x)\mu = p^*$ である. よって $\inf\mathcal{P} \leq p^*$ である.

さて, μ に対してその $\alpha \in \mathbf{N}_0^n$ 次のモーメント

$$y_\alpha = \int x^\alpha \mu(dx) \tag{6.4}$$

を考えよう. ここで $x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ である.

以下 $n = 2$ で考える. 行列 $M_{i,j}(y)$ は $i + 1 \times j + 1$ 行列で, その (p, q) 成分 (添字は 0 から数える) がモーメント $y_{i+j-p-q,p-q}$ であるものと定義する. 行列 $M_{i,j}(y)$ を生成するプログラムを書いてみよう.

```
def mij(I,J) {
  M = newmat(I+1,J+1);
  for (P=0; P<=I; P++) {
    for (Q=0; Q<=J; Q++) {
       M[P][Q]=util_v("y",[I+J-P-Q,P+Q]);
    }
  }
  return(M);
}


end$
```

たとえば関数 `mij(1,1)` の戻り値は... となる.

行列 $M_m(y)$ を生成するプログラムを $n = 2$ の時に書いてみよう.

```
import("mm1.rr")$

def mm(M) {
  Size=0;
  for (I=0; I<=M; I++) {
    Size += I+1;
  }
  A = newmat(Size,Size);
  for (I=0; I<=M; I++) {
    for (J=0; J<=M; J++) {
       T = mij(J,I);
       P=I*(I+1)/2; Q=J*(J+1)/2;
       for (II=0; II<size(T)[0]; II++) {
         for (JJ=0; JJ<size(T)[1]; JJ++) {
            A[Q+II][P+JJ] = T[II][JJ];
         }
       }
    }
  }
  return (A);
}


end$
```

関数 `mm(2)` の戻り値は... となる. $M_m(y)$ は ?×? の実対称行列である.

内積計算を用いてこの行列が positive semidefinite であることを示そう. m 次以下の多項式 $p(x)$, $q(x)$ を次元 $s(m)$ のベクトルとみなす. p, q に対して実数

$$\langle q(x), p(x)\rangle = \int q(x)p(x)\mu(dx)$$

を対応させる. これは positive semidefinite である. $p(x), q(x)$ の係数をとりだして作った長さ $s(m)$ の縦ベクトルを p, q と書く事にすれば, $\langle q(x), p(x)\rangle = q^T M_m(y)p$ がなりたつ. よって $M_m(y)$ は positive semidefinite な行列である. 記号では $M_y(y) \succ 0$ と書く. $M_m(y)$ の固有値に 0 がなければ, positive definite な行列であり, $M_m(y)$ は内積を定義する.

さて $p(x)$ の最小値を求める問題を semi-definite programming の問題に緩和 (relaxiation) して解こう. 緩和法とは解を探す範囲をすこし広めて解をさがし, その解が元の問題の解になっていることを示す方法である.

$p(x)$ を実数係数の $2m$ 次の多項式とする. $p(x)$ を最小化する問題を考えたい. これを解くために, 次の LMI 最適化問題 (PSD — positive semidefinite プログラム) を考える. (LMI って何?)

$$\mathbf{Q} \mapsto \inf_y \sum_\alpha p_\alpha y_\alpha, \quad M_m(y) \succ 0 \tag{6.5}$$

または

$$\mathbf{Q} \mapsto \inf_y \sum_\alpha p_\alpha y_\alpha, \quad \sum_{\alpha\neq 0} y_\alpha B_\alpha \succ -B_0 \tag{6.6}$$

ここで B_α は B_α の定義は上の式から直ちにわかると思うが, 2 変数, $m = 1$ の時に書いておくと次のようになる.

$$\begin{pmatrix} y_{0,0} & y_{1,0} & y_{0,1} \\ y_{1,0} & y_{2,0} & y_{1,1} \\ y_{0,1} & y_{1,1} & y_{0,2} \end{pmatrix}$$

$$= y_{0,0}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + y_{1,0}\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + y_{0,1}\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} +$$

$$
\begin{aligned}
& y_{2,0}\begin{pmatrix}0&0&0\\0&1&0\\0&0&0\end{pmatrix} + y_{1,1}\begin{pmatrix}0&0&0\\0&0&1\\0&1&0\end{pmatrix} + y_{0,2}\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix} \\
= \quad & y_{0,0}B_{0,0} + y_{1,0}B_{1,0} + y_{0,1}B_{0,1} + y_{2,0}B_{2,0} + y_{1,1}B_{1,1} + y_{0,2}B_{0,2}
\end{aligned}
$$

コラム: ここで少々補足. このような行列 B_α を TEX で作るには, print tex form で自動生成がとっても便利. それにちょっと触れておこう.

上の問題の双対問題は次のようになる. (参照はなにか?)

$$\mathbf{Q}^* \mapsto \sup_X \langle X, -B_0\rangle_t,\ \langle X, B_\alpha\rangle_t = p_\alpha, (\alpha \neq 0), \quad X \succeq 0 \tag{6.7}$$

ここで A, B を行列とするとき, $\langle A, B\rangle_t = \mathrm{tr}(AB)$ と定義する. したがってたとえば, $n = 2, m = 1$ の場合, $\langle X, B_{1,0}\rangle_t = x_{12} + x_{21}$ となる.

緩和により元の問題が解けてしまう場合の条件を調べよう. たとえば次のような条件がある.

定理 6.1 *[1, Theorem 3.2]* $p(x)$ の最小値が存在しその値が p^* だと仮定する. このとき $p(x) - p^*$ が *SOS,* つまり多項式 q_i が存在して

$$p(x) - p^* = \sum_{i=1}^{r} q_i(x)^2 \tag{6.8}$$

と書けるならば, $\min\mathbf{Q} = \min\mathbf{P} = p^*$ である. すなわち緩和問題の解がもとの解に等しくなる. さらに X を緩和問題の *dual* $\mathbf{Q}^*$ の解とすると,

$$X = \sum_{i=1}^{r} q_i q_i^T \tag{6.9}$$

となる. ここで q_i は $q_i(x)$ の係数から作ったベクトル. したがって, X の固有ベクトルを計算することにより q_i を求められる.

証明: 計算により $p(x) - p^* = \langle X, M_m(y)\rangle_t$ となることがわかる. ($n = m = 2$ の時に計算でたしかめるプログラムを書け. $X, M_m(y)$ は 3×3 行列となる.) $M_m(y) = \sum x^\alpha B_\alpha$ なので, $\langle X, B_\alpha\rangle_t = p_\alpha$, $\alpha \neq 0$ および $X(1,1) = -p^*$ となる. よって $\mathbf{Q}^*$ は X の存在ゆえ feasible である. すなわち X が $\mathbf{Q}$ の制約条件をみたし, さらに

$$p^* \leq \sup\mathbf{Q}^* \tag{6.10}$$

となる.

x^* は $p(x)$ の最小値を与える x である. このとき $y^* = (x_1^*, \ldots, x_n^*, (x_1^*)^2, \ldots)$ とおくと, $\mathbf{Q}$ は y^* の存在ゆえ feasible である. すなわち X が $\mathbf{Q}$ の制約条件をみたし, さらに

$$\inf\mathbf{Q} \geq p^* \tag{6.11}$$

となる. よって (6.11) および (6.10) より p^* で挟み撃ちされるので, $\inf\mathbf{Q} = \sup\mathbf{Q}^*$ となる. つまり y^* および X が最適問題の解であることがわかる. なお, y^* より x^* の値が, X の固有ベクトルを計算することにより, q_i がわかることになる.

計算例については [1] の例 1 を見よ.

固有ベクトルを求める関数については

`dvd-2006/src-ima/math-polyglot2/src/mat-eigen` を参照. (虚数が出てくる!). 対応システムは maple, math, maxima, octave.

6.4.1　SeDuMi と SOS tools

Todo: 続きはまだ書いてない. SeDuMi は商用システムの Matlab が必要 (iyokan-6).

関連図書

[1] J.Lasserre, Global optimization with polynomials and the problem of moments, SIAM Journal of Optimization, **11** (2001), 796–817.

第7章　積分とコホモロジ論

パラメータ付き積分のみたす微分方程式は超幾何方程式である. 逆に超幾何方程式を解こうと思うと, パラメータ付き積分を深く調べる必要がある. パラメータ付き積分とは homology 群と cohomology 群のペアリングに他ならない. この立場で超幾何方程式を計算で解こう (理解しよう) と思うとコホモロジ群の計算が不可欠となる.

7.1　超幾何積分の基礎

Todo: まだ書いてない. [SST; chap5].

7.2　D 加群の積分とそのアルゴリズム

黒板で. 積分とはなにか? 積分アルゴリズムとは?

数学関連事項はテキスト以外に板書でも説明. ビデオ: `d-integral-example-ja.mpg`

数学関連事項はテキスト以外に板書でも説明. ビデオ: `d-integral-ja.mpg`

sm1

Airy 積分

$$\int_C \exp(t^3 - xt)dt$$

のみたす微分方程式の導出.

```
[(parse) (cohom.sm1) pushfile] extension
[ [(Dt-(3 t^2 -x)) (Dx + t)] [(t)]
  [ [(t) (x)] [ ]] 0] integration message
```

積分アルゴリズムで重要な役割をはたすのが b 関数の計算である. b-function criterion.

M2

`2007/knx/leykin-2002-m2.pdf` 参照.

`globalBfunction`

asir

`bfct`

課題:

1. T.Oaku, Algorithmic methods for Fuchsian systems of linear partial differential equations [1] を読む. b-関数の計算の基本アイディア.

2. Toshinori Oaku and Nobuki Takayama, An algorithm for de Rham cohomology groups of the complement of an affine variety via D-module computation, [2] を読む (math.AG にもあり).

7.3　対数的コホモロジ群の計算

利用するシステム: asir, sm1

数学関連事項はテキスト以外に板書でも説明. ビデオ: `video risc-2005` の内容にしがう

```
module sm1;
localf integration0$

def integration0(II,V,VV) {
  P = sm1.get_Sm1_proc();
  if  (P < 0) sm1.start();
  A =[II,V,[VV,[ ]],0];
  push_int0(P,A);
  sm1(P," integration ");
  ox_check_errors2(P);
  R = pop(P);
  return(R);
}

endmodule;


def myhilb(I) {
  G = sm1.gb([I,[x,y]]);
  G = G[1];
  print(G);
  return sm1.hilbert([G,[x,y,dx,dy]]);
}


def  log2d(F) {
  Fx=diff(F,x);
  Fy=diff(F,y);
  S=sm1.syz([ [Fy,-Fx,F],[x,y]]);
  S=S[0];
  M = length(S);
  L = [ ];
  for (I=0; I<M; I++) {
    L = cons(adjOp(S[I][0],S[I][1],F),L);
  }
  return L;
}

def log2(F) {
 L = log2d(F);
  print(L);
  print(myhilb(L));
  return sm1.integration0(L,[x,y],[x,y]);
}

def adjOp(P,Q,F) {
  Ans=
  -Q*dx-diff(Q,x)+P*dy+diff(P,y)
  +diff(Q,x)-diff(P,y)
  +red((diff(F,y)*P-diff(F,x)*Q)/F);
  return Ans;
}

/*
  log2(x*y*(x-y));
  sm1.deRham([x*y*(x-y),[x,y]]);
  log2(x^4 + y^5  + x*y^4);
  sm1.deRham([x^4+y^5+x*y^4,[x,y]]);
*/

end$
```

```
$Computation of H2(log x*y*(x-y))$ message
[(parse) (cohom.sm1) pushfile] extension
[ [(x Dx + y Dy + 3),  ((2 y - x) x Dx + (2 x - y) y Dy)],
   (x,y)
  [[(x) 1 (Dx) -1 (y) 1 (Dy) -1]] ]  gb
 0 get dehomogenize /g1 set    g1 pmat ;
%% gb of adjoint is different !
g1 getRing ring_def ;
g1 0 get, (x,y) adjoint  /L1 set  g1 1 get, (x,y) adjoint /L2 set
g1 2 get, (x,y) adjoint /L3 set  g1  3 get, (x,y) adjoint /L4 set
[L1,L2,L3,L4] pmat
L1 (1). action pmat
L1 (x). action pmat
L1 (y). action pmat
L2 (1). action  pmat
L3 (1). action pmat
L4  (1). action pmat
(Dx).  , g1 3 get, mul (x,y) adjoint (1). action pmat
(Dy).  , g1 3 get, mul (x,y) adjoint (1). action pmat
L4 (x). action pmat
L4 (y). action pmat
```

従来の DeRham との比較

```
cd /usr/local/OpenXM/lib/k097/lib/restriction
openxm k0

NoX=1;
load["demo.k"];;

;;
```

```
load["demo.k"];;
nonquasi2(4,5);
nonquasi2(6,8);

nonquasi2(8,10);

DeRham2WithAsir(x^4+y^5+x*y^4);
DeRham2WithAsir(x^6+y^8+x*y^7);
DeRham2WithAsir(x*y*(x-y));
quit ;
```

関連図書

[1] T.Oaku, Algorithmic methods for Fuchsian systems of linear partial differential equations, Journal of Mathematical Society of Japan **47** (1995), 297–328.

[2] Toshinori Oaku and Nobuki Takayama, An algorithm for de Rham cohomology groups of the complement of an affine variety via D-module computation, Journal of Pure and Applied Algebra, **139**, (1999), 201–233.

第8章　$\mathcal{A}$超幾何方程式系

Knoppix/Math を利用して $\mathcal{A}$-hypergeometric system を調べよう.

8.1　方程式系の定義

課題: knx/g.pdf (GKZ) を読む.

8.2　$\mathcal{A}$超幾何方程式系と代数方程式系

8.3　$\mathcal{A}$-超幾何偏微分方程式系

z についての方程式

$$x_0 + x_1 z + \cdots + x_n z^n = g(z) = 0$$

を考える. $z = f(x_0, \ldots, x_n)$ は $x_0, \ldots, x_n$ の関数としてどのような関数なのであろうか.

定理 8.1 f は次の偏微分方程式系の解である.

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_k \partial x_\ell}, \quad i + j = k + \ell \tag{8.1}$$

$$\sum_{k=0}^{n} k x_i \frac{\partial f}{\partial x_i} + f = 0, \tag{8.2}$$

$$\sum_{k=0}^{n} x_i \frac{\partial f}{\partial x_i} = 0 \tag{8.3}$$

この偏微分方程式系は行列

$$A = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 2 & \cdots & n \end{pmatrix}$$

に付随した $\mathcal{A}$-超幾何方程式系 (GKZ-超幾何方程式系) に他ならない.

証明: $f(x_0, tx_1, t^2 x_2, \ldots, t^n x_n) = \frac{1}{t} f(x_0, x_1, \ldots, x_n)$ である (両辺とも $x_n (tz)^n + \cdots + x_0 = 0$ の解だから). これを t で偏微分して, $t \to 1$ とすれば (A.5) を得る. 方程式を t 倍しても根は不変なので, $f(tx_0, tx_1, \ldots, tx_n) = f(x_0, \ldots, x_n)$. 両辺を t で偏微分して, $t \to 1$ とすれば, (A.6) を得る.

次に $(\log g(t))' = \frac{g'(t)}{g(t)}$ に注意して, $\frac{\partial^2}{\partial x_i \partial x_j} (\log g(t))'$ を計算する.

$$\begin{aligned} & \frac{\partial^2}{\partial x_i \partial x_j} (\log g(t))' \\ = \quad & \frac{\partial^2}{\partial x_i \partial x_j} \frac{d}{dt} \log g(t) \\ = \quad & \frac{d}{dt} \frac{\partial}{\partial x_i} \frac{t^j}{g(t)} \end{aligned}$$

$$= \quad \frac{d}{dt}\left(-\frac{t^{j+i}}{g^2(t)}\right)$$

$\alpha_i(t)$ を方程式 $g(t)=0$ の根とする. $\frac{tg'(t)}{g(t)}dt = t\sum_i \frac{1}{t-\alpha_i(x)}$ なので, 道 Γ を根 $\alpha_p(x)$ 周りを回る道とすると,

$$2\pi\sqrt{-1}\alpha_p(x) = \int_\Gamma \frac{tg'(t)}{g(t)}dt = \int_\Gamma t\frac{d}{dt}\log g(t)dt$$

となる. 以上の計算より (A.4) がいえる. 証明おわり.

この方程式系の toric コンパクト化した無限遠の近傍での級数解の議論は面白い話題であるが, 長い議論となる.

8.4　超幾何微分方程式から導かれた常微分方程式

第**??**章では, 有理関数係数の 1 変数微分作用素環を考えた. ここでは有理関数係数の n 変数の微分作用素環を考える. $R = R_n = \mathbf{C}(x_1,\ldots,x_n)\langle\partial_1,\ldots,\partial_n\rangle$.
この環ではかけ算はたとえば次のようになる. $\partial_1\left(\frac{x_1}{1-x_2}\right)\partial_1 = \left(\frac{x_1}{1-x_2}\right)\partial_1^2 + \frac{1}{1-x_2}\partial_1$. J を R_n の左イデアルとする.

定理 8.2 *J が R_n の 0-次元イデアルであるとき, $J\cap\mathbf{C}\langle x,\partial_i\rangle$ は, 0 でない元を含む (つまり x_i についての常微分方程式を含む).*

この元は weight vector $(1,1,\ldots,1,0,1,\ldots,1)$ で D のグレブナ基底を計算することにより求めることが可能である. 別の方法としては weight vector $(0,\ldots,0;1,1,\ldots,1,0,1,\ldots,1)$ で D のグレブナ基底を計算することにより求めることが可能である. [D の解説まだ]

代数方程式系を解くのと同様にこの常微分を Runge-Kutta 法で数値的に解いたり, 級数解をもとめ, それからその解を別の方向へ延ばしていくという手法で数値解を構成することが可能である.

さて $J\cap\mathbf{Q}(x)\langle\partial_1\rangle$ を求めるといった消去法は時間がかかる計算でありなるべく避けたい. おなじような問題は多項式環でも発生している. 消去法 (lex order のグレブナ基底など) を用いない方法をまず多項式環で復習しよう.

定理 8.3 *$y^{\alpha(1)},\ldots,y^{\alpha(r)}$ を $\mathbf{Q}[y_1,\ldots,y_n]/I$ の $\mathbf{Q}$ 上のベクトル空間としての基底とする. 次の条件をみたす行列 $M_1,\ldots,M_n\in M(r,r,\mathbf{Q})$ が存在してかつ構成できる. 構成には グレブナ基底による normal form algorithm を用いる.*

$$\begin{aligned}
y_1\begin{pmatrix} y^{\alpha(1)} \\ \cdot \\ \cdot \\ y^{\alpha(r)}\end{pmatrix} &= M_1\begin{pmatrix} y^{\alpha(1)} \\ \cdot \\ \cdot \\ y^{\alpha(r)}\end{pmatrix} \mod I \\
\cdots &\quad \cdots \\
\cdots &\quad \cdots \\
\cdots &\quad \cdots \\
y_n\begin{pmatrix} y^{\alpha(1)} \\ \cdot \\ \cdot \\ y^{\alpha(r)}\end{pmatrix} &= M_n\begin{pmatrix} y^{\alpha(1)} \\ \cdot \\ \cdot \\ y^{\alpha(r)}\end{pmatrix} \mod I
\end{aligned}$$

この定理の D-analogy はつぎのようになる.

定理 8.4 $\partial^{\alpha(1)},\ldots,\partial^{\alpha(r)}$ を $\mathbf{Q}(x_1,\ldots,x_n)$ のベクトル空間 $\mathbf{Q}(x_1,\ldots,x_n)\langle\partial_1,\ldots,\partial_n\rangle/I$ の基底とする. このとき次の条件をみたす行列 $P_1,\ldots,P_n\in M(r,r,\mathbf{Q}(x))$ が存在してかつ構成できる. 構成には R でのグレブナ基底による *normal form algorithm* を使う.

$$\begin{aligned}
\partial_1\begin{pmatrix}\partial^{\alpha(1)}\\ \cdot\\ \cdot\\ \partial^{\alpha(r)}\end{pmatrix} &= M_1\begin{pmatrix}\partial^{\alpha(1)}\\ \cdot\\ \cdot\\ \partial^{\alpha(r)}\end{pmatrix} \mod I\\
\cdots &\quad \cdots\\
\cdots &\quad \cdots\\
\cdots &\quad \cdots\\
\partial_n\begin{pmatrix}\partial^{\alpha(1)}\\ \cdot\\ \cdot\\ \partial^{\alpha(r)}\end{pmatrix} &= M_n\begin{pmatrix}\partial_{\alpha(1)}\\ \cdot\\ \cdot\\ \partial^{\alpha(r)}\end{pmatrix} \mod I
\end{aligned}$$

このような形の方程式を Pfaffian とよぶ. さて f を解として, $F=(\partial^{\alpha(1)}\bullet f,\ldots,\partial^{\alpha(r)}\bullet f)^T$ とおこう. すると上の行列からつくった次の連立常微分方程式をとけばその解 F の第一成分をとることにより f が求まることとなる.

$$\partial_1\bullet F=M_1F,\cdots,\partial_n\bullet F=M_nF$$

大事な点はこの場合消去をやるような順序を選ぶ必要はなく, どんな順序を選んでも連立常微分方程式を作れるという点である. したがって計算に有利な順序を用いればよい. 差分化の場合には, 方程式に応じた差分の仕方により数値計算の様相がかわる. R での 0 次元系の場合には, 差分化のみならず順序の選び方による連立系への変形方法で数値計算の様相が変わることとなる. このあたりの詳しい事情はまだ深く調べられていない. 順序の選び方以外にも基底の選び方でも連立常微分方程式化の様子は変わる. たとえば $\partial^{\alpha(i)}$ の代わりにオイラー作用素 $\theta^{\alpha(i)}$, $\theta_i=x_i\partial_i$ を基底に選ぶと連立常微分方程式は簡単になることも多い.

例 8.1 $x_1+x_2z+x_3z^2=0$ の根 f のみたす偏微分方程式系を Pfaffian に変形しよう. $(f,x_3\partial_3f)$ をベースとしたとき M_1, M_2, M_3 は次のようになる.

$$M_1=\begin{pmatrix}-1/x_1 & 1/x_1\\ -x3/(4x_1x_3-x_2^2) & 0\end{pmatrix}$$

$$M_2=\begin{pmatrix}1/x_2 & -2/x_2\\ 2x_1x_3/(4x_1x_2x_3-x_2^3) & -1/x_2\end{pmatrix}$$

$$M_3=\begin{pmatrix}0 & 1/x_3\\ -x_1/(4x_1x_3-x_2^2) & 1/x_3\end{pmatrix}$$

この形の微分方程式は線形である. 第**??**章のホモトピー法で用いた方程式は非線形であった. 非線形方程式は一般に線形方程式より難しい. たとえば非線形方程式では, 解が爆発してしまう点の予測が難しい. 今の場合は線形なので係数の分母が 0 になる点を避けて計算をやっていけばよい.

```
dp_weyl_gr_main([x*dx+y*dy-1,x*dx-3],[x,y,dx,dy],1,1,2);
[y*dy+2,x*dx-3]

dp_weyl_set_weight(newvect(4,[0,0,1,0]));
```

```
/* (The last argument 0 implies the following.)
   Computing the Grobner basis with the weight vector [0,0,1,0]
   The tie-breaker is reverse lex order
*/
```

8.5 級数解の計算アルゴリズム

Todo: Eh/?? dsolv.

8.6 連立線形偏微分方程式系を数値計算で解くには

Todo: 微分方程式系の数値計算 ([OST] 準拠).

8.7 modified $\mathcal{A}$-超幾何方程式系についての計算実験

Todo: この節は未完成.
今後の課題: 特異点解消... Todo: Singular の 特異点解消の例.

第9章　Knoppix/Math について

9.1　概観

9.1.1　math-polyglot

Todo: まだ書いてない.

9.1.2　研究集会対応フォルダについて

9.1.3　Free Documents

9.2　n 計算機言語対応表

代入

繰り返し

条件判断

関数

変数

入出力

9.3　開発者になる方法

cvs とか sourceforge など.
Todo: まだ書いてない.

9.4　今後の課題など

第10章　数学公式集

まだ書いていない.

10.1　公式集としての数学ソフト

数学関連事項はテキスト以外に板書でも説明. ビデオ: OpenXM/fb のビデオあり． iamc2003

10.2　公式集から数学支援へ

付 録A　超幾何方程式, 関数について

序文でも述べたように, “方程式はいろいろなものがあるが, 筆者はいろんな偶然がかさなり, 微分方程式とくに超幾何方程式系を親しく研究することとなった. 超幾何方程式系は数学のさまざまな分野に関係がある.”

というわけでちょっとだけ超幾何関数って何？という数学の話題にふれておこう. この部分はこの本の Knoppix/Math の例題に出現する例の動機やバックグラウンドの理解に有益かもしれない.

A.1　初等的性質

数学関連事項はテキスト以外に板書でも説明. ビデオ: 黒板での説明:　knx-05-18

Todo: 級数としての定義.　　収束半径.

$$\tan^{-1}(x) = xF(1/2,1,3/2;-x^2) \tag{A.1}$$

$$\log(1-x) = -xF(1,1,2;x) \tag{A.2}$$

$$F(1/2,1/2,1;x) \tag{A.3}$$

Todo: 微分方程式, 積分表示, グラフ.

以上の参考文献は, 高野, 常微分方程式 (朝倉).

Todo: 一般化. 応用. Airy, Legendre, Bessel など. 代数幾何.

Todo: 代数方程式との関連.

A.2　代数方程式の根と超幾何微分方程式

A.3　$\mathcal{A}$-超幾何偏微分方程式系

z についての方程式

$$x_0 + x_1 z + \cdots + x_n z^n = g(z) = 0$$

を考える. $z = f(x_0,\ldots,x_n)$ は $x_0,\ldots,x_n$ の関数としてどのような関数なのであろうか.

定理 A.1 *f は次の偏微分方程式系の解である.*

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_k \partial x_\ell}, \quad i+j = k+\ell \tag{A.4}$$

$$\sum_{k=0}^{n} k x_i \frac{\partial f}{\partial x_i} + f = 0, \tag{A.5}$$

$$\sum_{k=0}^{n} x_i \frac{\partial f}{\partial x_i} = 0 \tag{A.6}$$

この偏微分方程式系は行列

$$A = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 2 & \cdots & n \end{pmatrix}$$

に付随した $\mathcal{A}$-超幾何方程式系 (GKZ-超幾何方程式系) に他ならない.

証明: $f(x_0, tx_1, t^2x_2, \ldots, t^nx_n) = \frac{1}{t}f(x_0, x_1, \ldots, x_n)$ である (両辺とも $x_n(tz)^n + \cdots + x_0 = 0$ の解だから). これを t で偏微分して, $t \to 1$ とすれば (A.5) を得る. 方程式を t 倍しても根は不変なので, $f(tx_0, tx_1, \ldots, tx_n) = f(x_0, \ldots, x_n)$. 両辺を t で偏微分して, $t \to 1$ とすれば, (A.6) を得る.

次に $(\log g(t))' = \frac{g'(t)}{g(t)}$ に注意して, $\frac{\partial^2}{\partial x_i \partial x_j}(\log g(t))'$ を計算する.

$$\begin{aligned} & \frac{\partial^2}{\partial x_i \partial x_j}(\log g(t))' \\ = & \frac{\partial^2}{\partial x_i \partial x_j}\frac{d}{dt}\log g(t) \\ = & \frac{d}{dt}\frac{\partial}{\partial x_i}\frac{t^j}{g(t)} \\ = & \frac{d}{dt}\left(-\frac{t^{j+i}}{g^2(t)}\right) \end{aligned}$$

$\alpha_i(t)$ を方程式 $g(t) = 0$ の根とする. $\frac{tg'(t)}{g(t)}dt = t\sum_i \frac{1}{t-\alpha_i(x)}$ なので, 道 Γ を根 $\alpha_p(x)$ 周りを回る道とすると,

$$2\pi\sqrt{-1}\alpha_p(x) = \int_\Gamma \frac{tg'(t)}{g(t)}dt = \int_\Gamma t\frac{d}{dt}\log g(t)dt$$

となる. 以上の計算より (A.4) がいえる. 証明おわり.

この方程式系の toric コンパクト化した無限遠の近傍での級数解の議論は面白い話題であるが, 長い議論となる.

A.4 超幾何微分方程式から導かれた常微分方程式

第**??**章では, 有理関数係数の 1 変数微分作用素環を考えた. ここでは有理関数係数の n 変数の微分作用素環を考える. $R = R_n = \mathbf{C}(x_1, \ldots, x_n)\langle\partial_1, \ldots, \partial_n\rangle$.
この環ではかけ算はたとえば次のようになる. $\partial_1\left(\frac{x_1}{1-x_2}\right)\partial_1 = \left(\frac{x_1}{1-x_2}\right)\partial_1^2 + \frac{1}{1-x_2}\partial_1$. J を R_n の左イデアルとする.

定理 A.2 *J が R_n の 0-次元イデアルであるとき, $J \cap \mathbf{C}\langle x, \partial_i\rangle$ は, 0 でない元を含む (つまり x_i についての常微分方程式を含む).*

この元は weight vector $(1, 1, \ldots, 1, 0, 1, \ldots, 1)$ で D のグレブナ基底を計算することにより求めることが可能である. 別の方法としては weight vector $(0, \ldots, 0; 1, 1, \ldots, 1, 0, 1, \ldots, 1)$ で D のグレブナ基底を計算することにより求めることが可能である. [D の解説まだ]

代数方程式系を解くのと同様にこの常微分を Runge-Kutta 法で数値的に解いたり, 級数解をもとめ, それからその解を別の方向へ延ばしていくという手法で数値解を構成することが可能である.

さて $J \cap \mathbf{Q}(x)\langle\partial_1\rangle$ を求めるといった消去法は時間がかかる計算でありなるべく避けたい. おなじような問題は多項式環でも発生している. 消去法 (lex order のグレブナ基底など) を用いない方法をまず多項式環で復習しよう.

定理 A.3 $y^{\alpha(1)},\ldots,y^{\alpha(r)}$ を $\mathbf{Q}[y_1,\ldots,y_n]/I$ の $\mathbf{Q}$ 上のベクトル空間としての基底とする. 次の条件をみたす行列 $M_1,\ldots,M_n \in M(r,r,\mathbf{Q})$ が存在してかつ構成できる. 構成には グレブナ基底による *normal form algorithm* を用いる.

$$
\begin{aligned}
y_1 \begin{pmatrix} y^{\alpha(1)} \\ \cdot \\ \cdot \\ y^{\alpha(r)} \end{pmatrix} &= M_1 \begin{pmatrix} y^{\alpha(1)} \\ \cdot \\ \cdot \\ y^{\alpha(r)} \end{pmatrix} \mod I \\
\cdots &\quad \cdots \\
\cdots &\quad \cdots \\
\cdots &\quad \cdots \\
y_n \begin{pmatrix} y^{\alpha(1)} \\ \cdot \\ \cdot \\ y^{\alpha(r)} \end{pmatrix} &= M_n \begin{pmatrix} y^{\alpha(1)} \\ \cdot \\ \cdot \\ y^{\alpha(r)} \end{pmatrix} \mod I
\end{aligned}
$$

この定理の D-analogy はつぎのようになる.

定理 A.4 $\partial^{\alpha(1)},\ldots,\partial^{\alpha(r)}$ を $\mathbf{Q}(x_1,\ldots,x_n)$ のベクトル空間 $\mathbf{Q}(x_1,\ldots,x_n)\langle\partial_1,\ldots,\partial_n\rangle/I$ の基底とする. このとき次の条件をみたす行列 $P_1,\ldots,P_n \in M(r,r,\mathbf{Q}(x))$ が存在してかつ構成できる. 構成には R でのグレブナ基底による *normal form algorithm* を使う.

$$
\begin{aligned}
\partial_1 \begin{pmatrix} \partial^{\alpha(1)} \\ \cdot \\ \cdot \\ \partial^{\alpha(r)} \end{pmatrix} &= M_1 \begin{pmatrix} \partial^{\alpha(1)} \\ \cdot \\ \cdot \\ \partial^{\alpha(r)} \end{pmatrix} \mod I \\
\cdots &\quad \cdots \\
\cdots &\quad \cdots \\
\cdots &\quad \cdots \\
\partial_n \begin{pmatrix} \partial^{\alpha(1)} \\ \cdot \\ \cdot \\ \partial^{\alpha(r)} \end{pmatrix} &= M_n \begin{pmatrix} \partial_{\alpha(1)} \\ \cdot \\ \cdot \\ \partial^{\alpha(r)} \end{pmatrix} \mod I
\end{aligned}
$$

このような形の方程式を Pfaffian とよぶ. さて f を解として, $F=(\partial^{\alpha(1)}\bullet f,\ldots,\partial^{\alpha(r)}\bullet f)^T$ とおこう. すると上の行列からつくった次の連立常微分方程式をとけばその解 F の第一成分をとることにより f が求まることとなる.

$$\partial_1\bullet F = M_1F,\cdots,\partial_n\bullet F = M_nF$$

大事な点はこの場合消去をやるような順序を選ぶ必要はなく, どんな順序を選んでも連立常微分方程式を作れるという点である. したがって計算に有利な順序を用いればよい. 差分化の場合には, 方程式に応じた差分の仕方により数値計算の様相がかわる. R での 0 次元系の場合には, 差分化のみならず順序の選び方による連立系への変形方法で数値計算の様相が変わることとなる. このあたりの詳しい事情はまだ深く調べられていない. 順序の選び方以外にも基底の選び方でも連立常微分方程式化の様子は変わる. たとえば $\partial^{\alpha(i)}$ の代わりにオイラー作用素 $\theta^{\alpha(i)}$, $\theta_i = x_i\partial_i$ を基底に選ぶと連立常微分方程式は簡単になることも多い.

例 A.1 $x_1 + x_2 z + x_3 z^2 = 0$ の根 f のみたす偏微分方程式系を Pfaffian に変形しよう. $(f, x_3\partial_3 f)$ をベースとしたとき M_1, M_2, M_3 は次のようになる.

$$M_1 = \begin{pmatrix} -1/x_1 & 1/x_1 \\ -x3/(4x_1x_3 - x_2^2) & 0 \end{pmatrix}$$

$$M_2 = \begin{pmatrix} 1/x_2 & -2/x_2 \\ 2x_1x_3/(4x_1x_2x_3 - x_2^3) & -1/x_2 \end{pmatrix}$$

$$M_3 = \begin{pmatrix} 0 & 1/x_3 \\ -x_1/(4x_1x_3 - x_2^2) & 1/x_3 \end{pmatrix}$$

この形の微分方程式は線形である. 第**??**章のホモトピー法で用いた方程式は非線形であった. 非線形方程式は一般に線形方程式より難しい. たとえば非線形方程式では, 解が爆発してしまう点の予測が難しい. 今の場合は線形なので係数の分母が 0 になる点を避けて計算をやっていけばよい.

```
dp_weyl_gr_main([x*dx+y*dy-1,x*dx-3],[x,y,dx,dy],1,1,2);
[y*dy+2,x*dx-3]

dp_weyl_set_weight(newvect(4,[0,0,1,0]));
/* (The last argument 0 implies the following.)
   Computing the Grobner basis with the weight vector [0,0,1,0]
   The tie-breaker is reverse lex order
*/
```

付 録B　あとがき

ざったな紹介になってしまったが仕方がない. 私はこう knoppix/math を楽しむという, 一つの例である.

章毎に独立して読めるように, 重複をいとわない書き方をとった.

このような概論的話が, 講義として成り立つが不安があったが, 聴講者のみなさんは私のサンプルプログラムをもとに, 自分の興味と知識に従い自由にいろいろな例を試したり, プログラムを書いていろんなことを試してみてくれた (LAPTOP 持ち込み歓迎). というわけで, 不安は杞憂であった.

knoppix/math の楽しさは個性. 商用システムは匿名の作者の部分が多い.

ソフト開発の仕組みにもふれたい. cvs や sourceforge など.

Knoppix/Math や unix 一般に関するその他のご相談歓迎です.

マニュアルはかなり計算機に習熟した人にも読みにくい. video アーカイブやこのノートや math-polyglot, サンプルプログラムが助けになるとうれしい.

数学の海を knoppix/math で自由自在に泳ぎ回ってみたい人のためのガイド.

計算機と数学の議論をしたい. 人生をすべて記録, 検索する方法 (日経サイエンス 2007-06, Gordon Bell, Jim Gemmell, 暮らしをまるごとディジタル記録, 58–67).

利用方法もいろいろある. ちょっとつかってみるつまみ食い. やる気. 復習. 研究プログラム. hobby projects.

Todo: RISC(2006, 5 月) の presentation はそのままもっていくといいかも...hd=bp

付 録C　講義のための補足ノート—神戸大学大学院毎週開講

C.1　2007-04-13

単位: レポートの提出 (回数未定) と出席をかねた小レポート提出 60 %です. LAPTOP コンピュータの持ち込みは歓迎.

サンプルプログラムを実行したり, レポートプログラムを提出するには knoppix/math 環境での実行が必須です. (たとえば maxima は Windows でも動きますが, knoppix/math で動かすのが条件)

C.1.1　概要と数学の題材

最初の章はは一部の学部の 1 年生が利用している教科書 “三宅敏恒, 入門線形代数” から問題をとりあげています. TA としての仕事などにこの講義のソフトを工夫活用するのも大歓迎です. この本にでてくる簡約な行列というのは reduced Gröbner basis の特別な場合にほかなりません.

ではテキストの 1 章の行列計算の部分の説明の前に knoppix/math がたとえばどのように動くのか, `xmaxima` を用いて x^5-1 の因数分解 `factor(x^5-1)` や x^2-y^2 のグラフの描画 `plot3d(x^2-y^2)` や $\log x$ の不定積分の計算 `integrate(log(x))` をデモしてみよう.

C.1.2　実習課題

講義中にサンプルのプログラムを動かすのをデモする. とりあえずサンプルのプログラムを独力で動かす. 動かした小レポート提出 (形式は講義中か次回に説明) を次次回に提出 (出席兼用)

C.1.3　Knoppix/Math の入手方法/実行環境の構築方法

数学科の計算機 ID を持ってる人

数学科内部からなら unix 機 (orange2 か Mac) の xterm か kterm (ターミナル) より, ssh コマンド

```
ssh  -X -Y iyokan-6
```

で Knoppix/math が動作している iyokan-6 へログインできる. (-Y がないと Mac では X Error (BadAtom)).

CD/DVD を入手する

講義中にあまってるのを配ります.

VMware/Knoppix/Math をインストールする

VMware/Knoppix/Math は Windows (や Mac) の VMWareplayer の上で動作する knoppix/math のイメージです.
`VMWare/Knoppix/Math` で google 検索するとダウンロードのページに到達. くわしい解説もついてます.

なおメインメモリが 512M 未満の人は VMWare/Knoppix/Math を動作させるのはちょっと厳しいと思います. メインメモリの大きさは, マイコンピュータを右クリックしてプロパティを選択すると全般タブに表示されます.

メインメモリが小さい場合は, CD から KDE を使わないモードでつぎのように Knoppix/Math を起動します.

```
knoppix:  knoppix  desktop=fluxbox
```

とか

```
knoppix:  knoppix  desktop=twm
```

など軽量のウインドーマネージャで起動して, なるべくコマンドラインで利用します. テキストモードだけで利用可能なソフトは

```
knoppix:  knoppix  4
```

としてテキストモードで起動します (unix シェルがひとつあるだけ)

C.1.4 Knoppix/Math のブート (起動方法) の仕方

knoppix の boot のビデオを見せる. Parallels でも実演.

C.1.5 その他

TEX を学習中の人には, `print_tex_form` の出力は参考になるでしょう.

C.2 2007-04-20

行列の簡約な形

行列の簡約な形については黒板で復習.

Knoppix/math, VMWare/Knoppix/Math のビデオによる紹介

Macaulay2

(時間があればやる.)
iyokan-6 で emacs 版の Macaulay2 を起動するには

```
emacs --load 'M2.el'
```

起動のあと, esc x M2 ENTER

出席用フィードバック用紙の形式

1. 返却しません. 疑問に対する解答等は講義中に一括してとりあげます.
2. A4 用紙 1 枚.

氏名	学籍番号	学科

提出日:

やってみたことの記述

例:

1. `iyokan-6` での使い方がよくわからなかった.
2. `VMWare/Knoppix/Math` を google で検索して Windows にインストールを試みた.
3. unix シェルから maxima を起動してみた.
4. 例 1.3 を参考にして $i-1$ 行が $(x_1^i, \ldots, x_n^i)$ であるような行列の逆行列を計算してその一般形が予想できないか試した.
5. 疑問: 正 n 角形を作図するソフトはありませんか?

結果, フィードバック, 疑問等

例:

1. インストールに問題はなかったが, ダウンロードの見積もり時間や, 必要なハードディスクの容量の記載がなかった.
2. やはり `xmaxima` の方が使いやすかった.
3. knoppix/math で作成したファイルのコピー方法がわからなかった.
4. 以下, 省略. 出力結果の張り付けなど.

C.3 2007-04-27

Leck0/3.tex

C.3.1 線形代数編

Singular や Macaulay は, 可換環論, 代数幾何用の専門家システムでもある. コマンドに登場するイデアル等の用語については後の方の章でくわしく説明する. 開発者等のプロフィールを video 等を交えて講義では紹介する.

線形代数編は統計ソフト R による Winger 則の実験で終了. 来月よりは微分積分微分方程式編.

語学とおなじでソフトは毎日使うのが大事.

1. ソフトの名前を覚えたか?
2. ある特定の仕事だけをやらせるためだけに使うのもいいかも.

C.3.2 課題

講義中に相談. 次回も講義用フィードバック用紙 (前回と同じ形式) を提出.

C.3.3 質問: VMware/Knoppix/Math をインストールしたが動作がおそい

A. VMware/Knoppix/Math は Windows (や Mac) の VMWareplayer の上で動作する knoppix/math のイメージです.

`VMWare/Knoppix/Math` で google 検索するとダウンロードのページに到達. くわしい解説もついてます.

なおメインメモリが 512M 未満の人は VMWare/Knoppix/Math を動作させるのはちょっと厳しいと思います. メインメモリの大きさは, マイコンピュータを右クリックしてプロパティを選択すると全般タブに表示されます.

メインメモリが小さい場合は, CD から KDE を使わないモードでつぎのように Knoppix/Math を起動します.

```
knoppix:  knoppix  desktop=fluxbox
```

とか

```
knoppix:  knoppix  desktop=twm
```

など軽量のウインドーマネージャで起動して, なるべくコマンドラインで利用します. テキストモードだけで利用可能なソフトは

```
knoppix:  knoppix  4
```

としてテキストモードで起動します (unix シェルがひとつあるだけ)

C.3.4 質問: `kterm` を MacOS X で起動できませんでした

A. `kterm` は大学の Mac にはインストールし忘れたようです. xterm と kterm の違いは漢字をつかえるか, つかえないかだけです. とりあえず次のようにします.

```
   ssh -X -Y iyokan-6
```
iyokan-6 号より
```
   kterm -km euc &
   kterm -km euc &
   kterm -km euc &
```
で必要なだけ iyokan-6 号の kterm を開く.

PATH 環境変数

一般にシェルからコマンドを起動できないときはまず, set コマンドで環境変数 PATH の値を確認します. シェルは環境変数 PATH に書かれているディレクトリ（フォルダ）からコマンド（プログラム) を探します. たとえば PATH の値が

```
/usr/sbin;/usr;/usr/bin
```

の場合は /usr/sbin, /usr, /usr/bin の順番にコマンドを探します. ls /usr/sbin とかをやってみて, kterm がなければすくなくとも標準の PATH にはこのようなファイルは存在していないということですね.

PATH にフォルダ (たとえば /usr/X11R6/bin) を追加するには csh 系の場合は

```
set path=(/usr/X11R6/bin $path)
```

を .cshrc ファイルに書いておきます. bash 系の場合は,

```
PATH=/usr/X11R6/bin:$PATH
export PATH
```

を .bashrc ファイルに書いておきます. unix では個人の環境設定はホームディレクトリ直下の “.” ではじまるファイル (ドットファイル) を編集することにより変更できます. これらのファイルはテキストファイルなので emacs などのテキストエディタで編集します. たとえば .cshrc を編集したいときは, emacs .cshrc で OK.

システム全体の環境設定は /etc 以下に書いてあるファイルを編集することにより変更できます. くわしくは unix の本をみてください.

C.3.5 数字の大きさ

ためしてがってん, 2007-04-25 より.

$$1 \times 2 \times \cdots \times 8 \times 9$$

はいくつくらい?

$$9 \times 8 \times \cdots \times 2 \times 1$$

はいくつくらい? 結構まちがう. アンカリング効果というらしい. 正しい値を数式処理系で計算してみよう.

C.4 2007-05-11

Leck0/4.tex

微分積分編

微分の計算, 代入, Taylor 展開, やさしい微分方程式とその解のグラフの表示.

参考 Wiki

`http://cc1.math.kobe-u.ac.jp/pukiwiki.php?KnxK2007`
cc1 の名前は時々かわります.

フィードバック 1

Q. 実行結果を記録してファイルに保存するにはどうしますか?
A. いろんな解決方法があります.

1. コマンドライン版のソフトの場合は, `emacs` の中から実行してそれを cut and paste で別のファイルへ保存します.

   ```
   emacs -e shell
   ```

   または emacs の中から `esc x shell` と入力すると, unix shell が emacs の中で動きます.

   Todo: カラー表示用の escape sequence を off にするには? `export PS1='> '` では prompt を変更できるだけ.

2. KDE のシェルの cut and paste を使う.

3. TeXMacs を使う.

4. 入力をファイルにしておいて,
   `maxima <`入力ファイル `>`出力ファイル とする.

なお, TEX に出力例を取り込むには `verbatim` 環境を用います.

Q. `kpdf` が日本語表示できない.
A. 一部 non-free soft が必要なため `apt-get` でパッケージ `xpdf-japanese` をインストールしてください. `apt-get` はパッケージ配布サーバの情報をダウンロードする初期化が必要ですので, 詳しいやりかたは,

`http://www.knoppix-math.org/mediawiki/index.php/KNOPPIX/Math/Japanese`

の記述を参考としてください.

Q. asir で行編集ができません.
A. メニューから起動するか, コマンドラインからの起動では `openxm fep asir` と起動する. `touch ~/.feprc` しておくとコマンドが記憶される.

Q. 動くけど遅い...
A. VMWare/Knoppix/Math は メインメモリ 512M 以上のマシンで動作させることをお薦めします. マイコンピュータを右クリックしてプロパティの表示で CPU やメモリ容量をしらべることが可能です. VMware/Knoppix/Math のホームページに注意事項として加筆しました.
起動のときの cheatcode を変更することにより, よりメモリの少ないマシンで動かせます.

Q. CD から起動できず.
A. BIOS で boot sequence は CD, HD の順番か?

Q. CD から起動できるが, DVD からは起動できない.
A. 工場でプレスした DVD 以外ではこのような現象がよくおきます.

Q. Kile はなかなか便利な TEX 環境ですね.
A. そのうち特集しなきゃ. わたしは emacs + 野鳥? で十分

Q. Display を高解像度にしたい.
`http://www.knoppix-math.org/mediawiki/index.php/KNOPPIX/Math/English` の記述をまねしてください. 以下引用します.

```
Q. I cannot get full resolution on intel MacBookPro Parallels?

A. Edit /etc/xorg.conf, apparently one cannot change
  the color depth from the default of 16.
  To change resolution, add in the subsection corresponding
  to depth 16 the 1440x99 mode, as in the following example:

  Section "Screen"
       Identifier "Screen0"
       Device     "Card0"
       Monitor    "Monitor0"
       DefaultColorDepth 16
       SubSection "Display"
               Depth     16
               Modes "1440x900" "1024x768" "800x600" "640x480"
       EndSubSection
```

Q. 統計システム R で hist グラムを細かくするには?
A. たとえば `winger.r` で `hist(e, breaks=15)` と入力します. R では, 名前=値 の形式でいろいろな補助的な値 (optional arguments) を与えることができます.

Q. プリンタの設定方法.
A. プリンタドライバが knoppix に対応してれば, VMware/knoppix/math の場合 samba の共有でできるはずですが, とりあえずは windows 側から knoppix がネットワークに見えるので,
`http://www.math.kobe-u.ac.jp/OpenXM/knoppix/vmkm-ja/html#q3` を参考にして ファイルをコピーして下さい.

Q. “command not found” って何ですか?
A. はいはい, 実演もしますが. C.3 節の PATH 環境変数に対応するフォルダにそのプログラムが存在していないとこのエラーがでます. Todo: 検索方法. unix では find コマンド.

C.5 2007-05-18

Leck0/5.tex

微分積分編, Graphic 表示など

1. おくればせながら全体の序章として超幾何関数の簡単な説明があるといいだろう, (板書).

$$\begin{aligned} \tan^{-1}(x) &= xF(1/2,1,3/2;-x^2) \\ \log(1-x) &= -xF(1,1,2;x) \\ & \quad F(1/2,1/2,1;x) \end{aligned}$$

2. 板書: 例題についての数学のバックグラウンド.
3. 板書. 各システムの概略の説明.

JavaView の実行方法

(個人メモ) 私の Mac で実行するには

```
cd Ec ; ~/Desktop/JavaView/javaview.term jv-simp.obj
```

右ボタンでコントロールパネルが出現.
orange2 で実行するには `jv-simp.obj` をテキストエディタで入力してから

```
/home/taka/JavaView/bin/javaview.orange2  jv-simp.obj
```

JavaView のサンプルを orange2 で見るには,

```
cd /home/taka/JavaView/bin ; ./javaview.orange2
```

フィードバック 5/10 日版より

Q. うまく動かない. うまくインストールできない. よって一回もフィードバックシートを提出してない.
A. 個別にご相談にのります.

Q. Windows の R がうまくインストールできない.
A. `http://www.math.kobe-u.ac.jp/HOME/taka/2007/knx`
に `R-2.3.1-win32.exe` (自己解凍形式, Windows 版 R システム) および `R-2.3.1-install.txt` (インストールに関する注意点) を書いておきました (フォントの設定).

Q. 日本人として国産システム asir を応援したいが, インタフェースが古風でなんとかなりませんか?
A. asir の主たる作者の考える基本インタフェースはファイルにお好みのテキストエディタでプログラムを書いて, `load` コマンドでファイルを読み込み利用します. (これが古風か...)

ちなみに Mac 版には GUI 版の asir (`cfep/asir` (google で download)) があります. TEX による出力を ON にすると, 出力も綺麗で TeXmacs より軽い動作をします. エディタはまだまだ改良の余地ありですが... 講義中にデモします.

asir は通信機能を持ってるので, それを利用して Windows 版の GUI を作成する試みもあります.
`http://cc1.math.kobe-u.ac.jp/pukiwiki.php?ShimoNo1130`
や `http://cc1.math.kobe-u.ac.jp/pukiwiki.php?ShimoNoAsir` を参照して下さい.
とにかく開発への参加大歓迎です.

Q. Asir の `load` コマンドで, カレントディレクトリのファイルを読めません.
A. ちなみにカレントディレクトリのファイルを読み込むには　`load("./ファイル名");`　と　`./`　が必要です. 環境変数 `ASIRLOADPATH` に　`.`　を加えておけば不要. これは `gcc` でコンパイルした C 言語のファイルを　`./a.out`　とか　`./a`　と実行するのと同じですね. 要するに `PATH` の値の設定のやり方と同じ.

Q. Macaulay2 の `res` コマンド (極小自由分解を生成するコマンド) のアルゴリズムを知りたい.
A. “代数幾何と数式処理” (高山信毅), 26–32, 数理科学, 7 月号, 1998 に入門的な解説および参考文献が書いてあります. 基本は多項式係数の一次不定方程式を解くことと, その解の基底の中から余分なものを除くのにまた一次不定方程式系を解きます. 一次不定方程式を解くことはグレブナ基底の応用なのであとの節でふれます. D 加群の filtered minimal resolution algorithm やその応用は最近の研究の話題です. 上の文献は `resol-surikagaku-1998.pdf` で `2007/knx` へおいておきます.

Q. Macaulay2 の標数の大きさの制限はどうにかなりませんか?
A. 標数に応じて最適なアルゴリズムが変っていくという理論的な問題, それから `int` のサイズをこした場合, 内部的な設計をどう変更していくかの問題など, 効率面からいろいろ研究課題が山積してるので, どうにもなってません. たとえば asir には小標数専用の多項式掛け算関数等があり, 特別なアルゴリズムを利用してますね.

Q. Macaulay2 の vector bundle の moduli 関連のパッケージ楽しみです
A. vector bundle の moduli の専門知識はないもので, feedback form には初歩的解説も (できたら) お願い. でもサイズの大きい問題はとけるのかな?

Q. Windows 版の xmaxima でグラフを書こうとすると, Windows のファイアウオールに止められる.
A. 理由の説明には IP アドレスとポート番号の知識が必要. とりあえずいつも許可を選んでください.

その他まだありますが, 次回以降とりあげます.

C.6 2007-05-25

Leck0/6

微分積分編, プログラミング入門. 数列の漸化式

`http://cc1.math.kobe-u.ac.jp/pukiwiki.php?KnxK2007`
cc1 の名前は時々かわります. 10 進数 basic へのリンクも.

iyokan-6 で

```
BASIC
```

と入力すれば, いちおう動きますが, ファイルの保存のとき画面が凍り付いたり (対策: kill コマンドでプロセスを消す), メニューのフォントがない等, いろいろと問題があるので, 各自自分のホームディレクトリにダウンロード, インストールして下さい. (自分のホームへインストールするとすこし改善する部分あり)

インストールのページに説明がある, `tar` はアーカイブ (多くのファイルを配布用に一つにまとめたもの) を展開, 作成する unix のコマンドです. Windows の zip と lha みたいなもの.

なお, 10 進 basic は windows で実習するのがトラブル知らずでお勧めです.

Q and A

Q. Mac と unix の間のファイルのコピーはどのようにしますか?
A. `scp` コマンド (ssh による copy コマンド) をつかうのが一番汎用的です.
例: Mac から orange2 のユーザ `sugaku` のホームへの ファイル `test.txt` のコピー. Mac の xterm か Mac のターミナルの window から

```
scp  test.txt  sugaku@orange2:
```

(orange2 のあとの : を忘れぬように.

Q. `xmaxima` が動かない.
A. ssh に `-X` オプションをつけてますか?

Q. Asir でプログラムを書いて $\zeta(2)$ の値の高精度計算を試みましたが, 時間とメモリを大量消費する. A. 有理数計算を多用すると約分に gcd 計算が多用されてメモリと時間を多用します. マニュアルの `ctrl("bigfloat",1);  setprec` を参照し, bigfloat を用いて, かつ和を小さい方から足していくとすこしよくなります. $zeta(s)$ の計算には特別な高速アルゴリズムがあると思います (私しらず). たとえば数論システムの pari では, 次のように計算します.

```
gp                  unix shell より pari/gp を起動.
zeta(2)
sqrt(6*zeta(2))     たしかに pi の値が...
```

Todo: 桁数を増やす方法.

付 録D　講義のための補足ノート—東京大学大学院集中講義

D.1　2007-05-28

No.1

単位: レポートの提出 (回数未定) と出席をかねた小レポート提出 60 %です. LAPTOP コンピュータの持ち込みは歓迎.

サンプルプログラムを実行したり, レポートプログラムを提出するには knoppix/math 環境での実行が必須です. (たとえば maxima は Windows でも動きますが, knoppix/math で動かすのが条件)

D.1.1　概要と数学の題材

最初の章は “三宅敏恒, 入門線形代数” から問題をとりあげています. TA としての仕事などにこの講義のソフトを工夫活用するのも大歓迎です. この本にでてくる簡約な行列というのは reduced Gröbner basis の特別な場合にほかなりません.

数学関連事項はテキスト以外に板書でも説明. ビデオ: `Castro-Urdiales` の風景, `icms2006`

ではテキストの１章の行列計算の部分の説明の前に knoppix/math がたとえばどのように動くのか, `xmaxima` を用いて x^5-1 の因数分解 `factor(x^5-1)` や x^2-y^2 のグラフの描画 `plot3d(x^2-y^2)` や $\log x$ の不定積分の計算 `integrate(log(x))` をデモしてみよう.

消去法を板書. Buchberger algorithm の特別な場合であることを強調.

D.1.2　実習課題

講義中にサンプルのプログラムを動かすのをデモする. とりあえずサンプルのプログラムを独力で動かす.

D.1.3　Knoppix/Math の入手方法/実行環境の構築方法

CD/DVD を入手する

講義中にあまってるのを配ります.

VMware/Knoppix/Math をインストールする

VMware/Knoppix/Math は Windows (や Mac) の VMWareplayer の上で動作する knoppix/math のイメージです.

`VMWare/Knoppix/Math` で google 検索するとダウンロードのページに到達. くわしい解説もついて

ます.

なおメインメモリが 512M 未満の人は VMWare/Knoppix/Math を動作させるのはちょっと厳しいと思います. メインメモリの大きさは, マイコンピュータを右クリックしてプロパティを選択すると全般タブに表示されます.

メインメモリが小さい場合は, CD から KDE を使わないモードでつぎのように Knoppix/Math を起動します.

```
knoppix:  knoppix  desktop=fluxbox
```

とか

```
knoppix:  knoppix  desktop=twm
```

など軽量のウインドーマネージャで起動して, なるべくコマンドラインで利用します. テキストモードだけで利用可能なソフトは

```
knoppix:  knoppix  4
```

としてテキストモードで起動します (unix シェルがひとつあるだけ)

D.1.4 Knoppix/Math のブート (起動方法) の仕方

knoppix の boot のビデオを見せる. Parallels でも実演.
(神戸での Mac から ssh での iyokan-6 への login によるデモは混乱している学生もいた.)

D.1.5 講義のみの資料 (PDF) はパスワード保護されています.

```
http://www.math.kobe-u.ac.jp/HOME/taka/2007/knx
```