PARI/GP チュートリアル

C. Batut, K. Belabas, D. Bernardi, H. Cohen, M. Oliver

Laboratoire A2X, U.M.R. 9936 du C.N.R.S
Université Bordeaus I, 351 Cours de la Libération
33406 TALENCE Cedex, FRANCE
e-mail: pari@math.u-bordeaux.fr

Home Page:
http://pari.math.u-bordeaux.fr/

Primary ftp site:
ftp://pari.math.u-bordeaux.fr/pub/pari/

last updated September 17, 2002
(this document distributed with version 2.2.8)

Translation: TOMINAGA Daisuke
Computational Biology Research Center,
National Institute of Advanced Industrial Science and Technology
Aomi 2-41-6, Koto, Tokyo, 135-0064, JAPAN
tominaga@cbrc.jp, http://www.cbrc.jp/~tominaga/
February 24, 2004

このテキストは計算プログラム GP のカイドまたは入門として書かれています。多くの実例を用意しますが、テキスト中に新しい関数が出てきたときには、その定義をユーザーマニュアルで見た方がいいでしょう。この後の章はそれぞれ独立して読むことができますが（たとえば、マニュアルを全部読まなくても GP で何ができるかを理解できます)、リファレンスマニュアルをあわせて読むと、より得るものが多くなるでしょう。

1 ようこそ - Greetings!

まずワークステーション（あるいは端末やパソコンなど）の前に座って、gp と入力してプログラムを起動してください（Enter キーを押すのを忘れないように。Macintosh の場合は Retern キーと間違えないように[1])。

すると GP の起動メッセージが表示され、プロンプト（?または gp >など）を表示して、ユーザーからの入力を待ちます。

ではキーボードで 2 + 2 と打ってみてください。どうなりますか？予想したとおりにはならなかったかもしれません。これはもちろん、あなたがどこまでを一度に入力したいのかを GP に伝える必要があるということです。Return キー（または Newline キー、Macintosh の場合は Enter キー）を 押すことで、そこまでが入力であることを示します。ここまで正確に行えば、期待する出力が得られます。もしあなたが、Macsyma や Maple などの他のプログラムを使ったことがあるなら、そのときは Return キーを押す前にセミコロン; を入力していたと思います。これらのプログラムではそうすることになっていますが、GP で同じようにすると、何を気取ってか、なんの返事もせずに次のプロンプトを表示します。GP での行末のセミコロンは、そこでの結果を表示せずに保持しておくことを指示するために使われ、このときは何も出力されません（出力が何ページにも及ぶようなときに使います)。

次は 27 * 37 と入力してみてください（*の両側には空白があります)。かけ算が行われますね。空白文字（スペース）は実際には必要ありません。27*37 と入力してみましょう。うまく行ってそうですね。このテキスト中では、入力例を読みやすくするために空白文字を入れているところがありますが、GP では空白は無視されるため、入力しなくてもかまいません。

さて、GP を起動してからもうずいぶん長い時間がたってしまいましたね。次に覚えることは GP の終了方法です。さまざまな計算システムはそれぞれに終了する方法がありますが（quit、exit、system など)、GP では quit または\q（バックシュラッシュq）です。q でも終了します[2]。やってみてください。

さて、できましたか？ GP は終了しているはずです。ではこのチュートリアルを続けるためにはどうしたらいいか分かりますか？前の方に書いてありますから、見てみてください。

今度はもっと意味のあることをしてみましょう。1/7 の小数で表現するには、いくつか面白い点があります。GP で見てみましょう。1 / 7 と入力してください。どうですか？ おかしなことに、入力したものをそのままオウム返ししてきましたね。こんな結果が見たかったわけじゃありませんよね。

などとボヤくのをやめて考えてみましょう。GP はもともと、物理学などのためというよりは、純粋な数学者のために作られたシステムです（もちろん物理学者も歓迎です)。そして数学的には、1/7 は有理数体 **Q** 上の元です。計算機が 1/7 を他にどう表現できるでしょう（2/14 でもかまいませんが、混乱するだけです)。PARI、そして GP はおおよそ常に、可能な限り精密な結果を返そう

[1]訳注：Return キーがある場合はそれでもかまいません。
[2]訳者検証 ver. 2.2.8 (development CHANGES-1.907) では q では終了しませんでした

とします（なぜ「おおよそ」なのかは後で説明します）。GP に入力された演算の結果が厳密に表現できるものであれば、それが出力されます。

しかし、そうは言ってもやはり小数で演算結果が見たい場合もあるでしょう。そのときは以下のように入力してください。

```
1./ 7
1 / 7.
1./ 7.
1 / 7 + 0. ...
```

分数は小数で表示され、すぐにたくさんの数字が画面に並び（多くの計算機環境では 28 個、一部では 38 個の数字）、142857 が繰り返し連なっているはずです。これは、たとえば `0.`、`1.` や `7.` などの「精密でない」実数値を含む演算を入力した場合、GP は厳密な演算結果を返すことができないからです。

計算機環境によって 28 個だったり 38 個だったりするのはなぜでしょう。精度の桁数は、デフォルトの初期値が GP を起動したときに表示されますが、これは計算が非常に速くできるように、また従来の倍精度浮動小数点演算よりも十進数で 12 桁正確に行えるように設定されています。この値は技術的な問題に依存しています。つまり、GP が実行されている計算機が 64bit integer をサポートしていれば（標準ライブラリが 2^{64} までの整数を扱えるなら）、デフォルトの精度は 38 桁に、そうでなければ 28 桁になります。ほとんどの場合は（`DEC alpha` などでなければ）後者なので、ここでは後者とします。

大型のメインフレームやスーパーコンピュータは精度が 28 桁の標準ライブラリを搭載していますが、その場合はそれが絶対的な精度の上限になります。しかしここではそうではなく、すぐ後に示すように、精度の桁数は好きなだけ多く設定できます。

なかなか本格的な例に進まないと思っているかもしれませんね。それでは、`exp(1)` と入力してみてください。すぐに e の値が 28 個の数字で出てきましたね。では `log(exp(1))` と入力してみてください。出てきた数値は、1 にかなり近くはありますが一致しません。数値的に演算を行った場合には、このように精度が失われるのです。さて、次には何ができるでしょうか。では `pi` と入力してください。パッと答えが出てきませんね。では `Pi` ではどうでしょう。これはうまくいきます。GP は英字の大文字と小文字を区別することを覚えていてください。

小文字でつづった `pi` は、ゴミ変数（`stupid garbage`）がそうであるように、それ自体では何ら意味はありません。どちらの場合も、GP はユーザーが入力したそのままに面白い名前の変数を作るだけです。試してみてください。入力に含まれる空白文字は無視されるので、これは `stupidgarbage` と入力したのと同じことになります。上の `27 * 37` の例では演算子と二つの演算数の間に空白文字を入れていますが、特に不自然な感じはしません。このことは GP スクリプトを書くときに重要になります（これについては後述します）。

話は変わりますが、GP に対して、知りたい識別子について質問することもできます。クエスチョン・マーク `?` に続けて、その語を続けて入力するだけです。`?Pi` や `?pi` と入力してみてください。`??Pi` などのようにクエスチョン・マークを二つ重ねると、環境によってはもっと詳しい説明を見ることができます。GP は起動時、著作権表示の前にこういった詳しい説明を用意しているかどうかを表示しています。同様に、そのときにたとえば `readline enabled` のように表示されていれば、ユーザーマニュアルの readline の章を見てみるといいでしょう。これは例を入力したり打ち間違いを修正したりするのに便利な機能です。

ではここで `exp(Pi * sqrt(163))` を試してみましょう。結果を見ると、少し前の例から察する

に、出力の最後の桁は恐らく間違った値で、正しい答えは整数と考えてもよさそうです。では精度を変更してみましょう。`\p 50` と入力してからまた `exp(Pi * sqrt(163))` と入力してください。さきほどの答えの、最後の 1 桁は間違ってるのではないか、という予想は正しかったようです。今度は 9 がたくさん出力されましたが、それに続く部分を見るとどうやら真の値は整数にはならないように見えます。これは PARI に含まれるバグで本当の答えは整数ではないのか、と思う人もいるでしょう。では続けて `sqr(log(%) / Pi)` としてみてください（`%`は直前の演算結果です。演算結果は順に番号が振られ、`%1`、`%2`... にはセミコロンを使ってわざわざ表示させなかった演算の結果も含まれます。これらは、それ以降の GP への入力に使うことができます。`sqr` は二乗（`sqr(x) = x * x`）です。二乗根の `sqrt` と混同しないようにしてください)。出力は 163 と区別がつきません。つまりバグではなかったと思われます。

$\exp(\pi * \sqrt{n})$ は整数でも有理数でもないことが知られています。n が零でない有理数の時に超越数になります。

GP は数値演算以外のこともできる素晴らしい計算プログラムでしょう？言い方を変えると、GP は従来の計算プログラムよりも遙かに強力な、任意の精度で計算ができる途方もない計算機です。

付記：（最初は読まなくてもかまいません。）

1) 以前の PARI を使ったことがあるなら、ver. 1.39.xx から非常に多くの変更点があるのに気づく（あるいはすでに気づいた）でしょう。特に関数名の多くが変わりました。これまでよりも論理的な名前になり、接頭辞が体系的に付けられたため、入力を自動補完する際には非常に便利になりました。関数名を見れば何に関するものなのかすぐにわかります。たとえば楕円曲線に関する関数はすべて `ell` という接頭辞が付いています。

すると当然、あなたが過去に作ったスクリプトが動かなくなるかもしれません。そこで互換レベルを設定することができます（`default(compatible, 3)` とすると、バージョン 1.39.15 の、太古の動作をさせることができます)。または、スクリプトを書き直しましょう。以前よりもわかりやすく書けるようになっているので、スクリプトがあまりに長い場合を除けば書き直すことをお薦めします（特に `break`、`next`、`return` といった新しい制御構文を使えるからです)。また関数名なども新しいのを用いる方がいいでしょう。将来のバージョンアップで、自動変換機能を搭載する**かも**しれません。

どの関数の名前が変更されたかは、`whatnow(function)` で表示されます。

2) このテキスト中では、厳密でない数値が入力されたときには厳密でない出力が得られることを前提にしています。しかしこれは必ずそうなるというわけではなく、一つだけ例外があります。厳密でないな数値と、厳密な 0 をかけ算した場合の演算結果は、厳密に 0 になります。`0 * 1.4` と `0. * 1.4` を比較してみてください。

3) 実数を表現する桁数は大きくできますが、整数も大きくできます。`100!`を試してみてください。演算結果が要するであろう桁数をあらかじめ指定しておく必要はありません。自動的に桁数が調整されます。一方で実数を含む多くの演算では、デフォルトの精度（初期値は 28 桁）を指定する必要があります。

4) 精度を 28 桁に戻して（`\p 28` と入力)、`exp(24 * Pi)` と入力してみてください。指数表記で出力されたはずです。GP では、演算結果が正しくならないときには、ユーザーもそれがわかっていなければいけません。

これまで精度は 28 桁として例を見てきましたが、$\exp(24 * \pi)$ の整数部は 33 桁あります。GP が忠実に 33 桁出力したとすると、末尾の数桁はおそらく間違った値になります。そういった意味の

ない数字を出力しないようにするために指数表記になります。

5) これを避けるためには二通りの方法があります。一つはもちろん、精度を 33 桁以上にすることです。試しに余裕を見て 40 桁にしてみましょう。それから最後の結果をもう一度表示させてみてください（`%n` と入力。`n` は結果の番号）。どうなりましたか？ 指数表記のままですね。なぜだか分かりますか？

もう一度、何が起こったか見てみましょう。精度を 40 桁にする前に行った演算は、28 桁の精度で行われました。したがって演算結果は 28 桁分しかありません。たとえ 40 桁でなく 1000 桁を指定したとしても、33 桁の整数部に対して今の結果は精度が 28 桁しかないことを GP は認識しています。そのために精度を後で大きくしても、表示される結果を高精度にすることはできません。ではどうしたらいいでしょう？ もう一度 `exp(24 * Pi)` と入力するとどうなるでしょうか。40 桁表示されるでしょうか。 試してみると、 今度は指数表示ではありませんね。固定小数点方式で表示されると、最後の 6 桁には間違った値が表示されます。

6) **注意** 以下を試してみてください。精度 28 桁で `default(format, "e0.50")`、続けて `exp(24 * Pi)` と入力してみてください。なぜ表示される結果が 50 桁の精度にならないか、なぜたくさん 0 がつながっている[3]か分かりますか？`log(exp(1))` と入力して、確認してみてください。`default(format,)` コマンドは出力の形式だけを変更し、精度は変わり**ません**。つまり、`\p` コマンドはその両方を変更すると言うことです。

7) もし現在の精度を忘れてしまって、表示されている結果の桁数を数えるのも面倒なときは？そんなときは、`default` コマンドを使って GP の内部変数を調べましょう（そしてそれを変更してみましょう）。`default(realprecision)` と入力し、続けて `default(realprecision, 38)` と入力してみましょう。後者は`\p(38)` とまったく同じことです。(単に前者は入力が面倒なだけです)。“default” コマンドは `format` と `realprecision` 以外にもあります。`default` だけを入力してみると、すべてが表示されます。

8)`default` コマンドは引数の数によって働きが異なることに注意してください。この点はほかの GP のコマンドと異なる振る舞いをするということになります。オンラインヘルプで見ることができます（第 3 章にも詳しく説明しています）が、関数のプロトタイプ中で括弧に囲まれた引数はオプション（指定しなければデフォルト値が指定されているのと同じことになる）です。これで、例に挙げられている値がどういう働きをするのか、またデフォルトの値がどうなのかを調べることができます。

2 腕ならし - Warming up

エラーメッセージについて見てみるのも大事なことです。`1/0` と入力してみてください。よく分からない出力が出ましたね。次に精度 28 桁で、いつもの例を試してみましょう。`floor(exp(24 *Pi))` と入力してみてください（`floor` は数学者の言う整数部のことで、計算科学者の `truncate` と混同しないように。`floor(-3.4)` は -4 に、`truncate(-3.4)` は -3 になります)。暗号のようなメッセージが表示されますが[4]、前章の付記を読めばすぐに意味がわかるはずです。それは読まなくてもいいと前述しましたが、簡単に言うと `exp(24 * Pi)` の整数部は 33 桁あるので、精度 28 桁では正確には計算できないと言うことです。

他にももっと分かりにくい、暗号化されたかのようなエラーメッセージがいくつかあります（TEX のエラーメッセージほどではありませんが)。

[3]訳者検証では 28 桁のままです。
[4]訳注：gp を起動し直すか、精度を元に戻しておいてください。

試しに log(x) と入力してみましょう。これもよくわかりません。しかしメッセージはそう難しいことを言っているわけではなく、GP は log(x) がなんだかわからない、と表示しているだけです[5]（しかし log 関数はあります。?log と入力すると説明が表示されます）。

次に sqrt(-1) ではどんなエラーメッセージが出るか見てみてください。GP は複素数を扱えるので、このやり方ではエラーになりません。同じように今度は log(-2) や exp(I*Pi)、I^I などを試してみてください。様々な実数、複素数の処理が行えます（多価の複素超越数の関数がいくつかユーザーマニュアルに説明してあります）。また、I と i を混同しないように気をつけましょう。I^2 と i^2 を比べてみると分かります。

本題ではありませんが、6*zeta(2) / Pi^2 はどうなりますか？ よく似た値になりますね。

さて前出の例で、GP は log(x) がなんだか理解できないというエラーがでましたが、log の値を計算することはできました。これはもどかしい感じがします。指数関数は理解してくれないでしょうか。やってみましょう。exp(x) と入力するとどうなりますか？ これはなんでしょうね？ これまでに他の計算プログラムを使ったことのある人は、結果は単に exp(x) がまた出てくるだけだろうと思っていたでしょう。しかし GP の出力は x = 0 での 16 項のテイラー展開です。16 という値はデフォルトの seriesprecision の値で、\ps$n$ または default(seriesprecision, n) で任意に変更できます。n はべき級数の項の数です（数列の末尾は O(x^16) と表示されますが[6]、GP でのこの型のオブジェクトに特有なものです。数値解析におけるビッグ・オー “Big-O” 記法と似ています）。

$x = 0$ で展開できるものについては、自動的にテイラー展開が行われます。したがって、0 で定義されない log(x) については何も行われません。log(1+x) ならうまくいくでしょう。では 0 以外の点で展開したいときはどうしたらいいでしょう？ x を $x - a$ に置き換えたいときは？ それは単に、たとえば x= 2 について log を得たいときは log(x+2) とするだけです。serieslength を（\ps で）変えながら以下の例で練習してみましょう。

```
cos(x)          cos(x)^2 + sin(x)^2     exp(cos(x))
gamma(1+x)      exp(exp(x) - 1)         1 / tan(x)
```

他の例も試してみましょう。(1 + x)^3 はどうでしょう。演算結果は多項式なので、ここでは O(x) はありません。また指数部が零または正の整数でないときは無限個の非零の項のべき級数を得られます[7]。それを (1 + x)^(-3) で見てみましょう（-3 の前後の括弧は不要ですが、見やすくするために書いてあります）。おどろいたことに、期待されるのとは違ってべき級数ではなく、有理式が出力されます。これは 1 / 7 という入力に対してそのまま 1/7 が返されるのと同じ理由です。PARI は、厳密な演算結果が得られているのにあえて厳密でない結果を表示することはありません。

しかしやはり、1/7 の例で示したようにたとえ厳密でない結果でも、見たい形でほしいときはあります。たとえばべき級数でほしいときは、

```
(1 + x)^(-3) + O(x^16)
(1 + O(x^16)) * (1 + x)^(-3)
(1 + x + O(x^16))^(-3), ...
```

のようにします。また演算数をべき級数に変換する演算子も使えます。その時で seriesprecision で行うには、以下のようにします。

[5]訳者検証では ‘log is not analytic at 0.” と表示され、少し意味が違います。
[6]訳者検証では O(x^17) でした。
[7]訳注：項数が有限個で打ち切られないということでしょうか？

```
Ser( (1 + x)^(-3) )
```

この例 `(1 + x)^(1/2)` はどうでしょう？この演算では、PARI では厳密に結果を得ることができないのでべき級数で表示されます（PARI に代数演算の方法を指示することもできますが、それは `(1 + x)^Pi` の例で示します）。前出の例を同様に、`(1 + x)^(-3.)` としてみましょう。`-3.` は厳密な値ではないので、PARI は入力が有理式であるとは判断せず、代わりにべき級数を用います。係数も整数ではなく実数値になります。

最後に単なる気晴らしですが、`taylor((1 + x)^(-3), x)` を試してみてください（`taylor` の使い方をマニュアルで見てみてください）。これは実際に使われることはまずないでしょう。

この章をまとめると、整数、実数、有理数に加えて PARI では複素数、多項式、べき級数、有理式を取り扱うことができ、これらを扱う多くの関数があり、このテキストでその一部を見てみた、ということになります。

付記：（前章と同様に、最初はこれらは読み飛ばしてかまいません。）

1) 以下の例が再現できるように、`\y` で自動簡素化機能を無効にしてください。

複素数には実部と虚部があります（誰が考えたか知りませんが）。虚部が厳密に 0 のときは実部しか表示されません。しかしそれは、その複素数が実数に変換されているということではないことに注意してください。虚部が表示されないと見た目は実数と同じですが、実数を期待して動作するプログラムでは不都合を生じることがあります。たとえば `n = 3 + I - I` では 3 が出力されますが、これは複素数です。`type(n)` で確認してみてください[8]。それから `(1+x)^n` としてみると、多項式 `1 + 3*x + 3*x^2 + x^3` が期待されますが、代わりにべき級数が出力されます[9]。まずいことに、数論的関数を使おうとするとき、たとえばオイラーのトーシェント関数（`eulerphi` で使えます）では、「数論的関数は整数引数を要求します（“arithmetic functions want integer.”）」というエラーメッセージを表示します。しかし 3 は一見整数にしか見えませんから、一人で考えてもよく意味の分からないメッセージになってしまいます（ここがよく理解できない場合は、もう一度よく読んでください。ここが原因で理解できないことが数多くあります）。

同様に `3 + x - x` も整数の 3 にはならず（変数 `x` に関する）定値の多項式になり、$3 = 3x^0$ と同値です[10]。

もし可能な限り簡素な形の解がほしい場合（たとえば数論的関数を適用する前やとにかく手早くやりたい場合）は `simplify` 機能を演算結果に適用してください。実際には GP は，コマンドの終了時にいつもこの機能を自動的に適用しています。この機能は処理途中の結果を簡素化するわけではないことに注意してください。この機能は`\y` で有効／無効を切り替えることができます。この章の最初にこれを入力するようにしたのは、これが理由です。

2) すでにふれたように、べき級数展開は常に `x=0` について行われます。`x=a` で行いたいときには、展開したい関数の `x` を `z + a` で置き換えてください。複雑な関数では置換機能 `subst` を使った方が簡単かもしれません。たとえば `p = 1 / (x^4 + 3*x^3 + 5*x^2 - 6*x + 7)` の場合、これを入力し直そうとは思わないでしょう。`subst(p, x, z+a)` と入力することで `x` を `z + a` で置き換えることができます（マニュアルで `subst` 機能を調べてみてください）。

`p = 1 + x + x^2 + O(x^10)` を試してみましょう。続けて `subst(p, x, z+1)` と入力して出てくるエラーメッセージの意味がわかりますか？

---

[8] 訳者検証では困ったことに `t_INT` と出てきました。複素数なら `t_COMPLEX` と表示されます。

[9] 訳者検証ではそういうわけで、普通に展開されます。

[10] 訳注：`type(3 + x - x)` は `t_POL` になります。

3 その他の型 - The Remaining PARI Types

PARI で用いられる型について、もう少し触れます。

`p = x * exp(-x)` と入力すると、期待されるとおりに 16 項のべき級数が出力されます（デフォルト値を変更してなければ)。ここで `pr = serreverse(p)` としてみましょう。これはべき級数の「逆戻り」、つまり逆関数を求めるコマンドです。これはべき級数の最初の非零の係数が x^1 のときにだけ求めることができます。得られた結果が正しいかどうか確かめるために `subst(p, x, pr)` または `subst(pr, x, p)` としてみると、`x + O(x^17)` という出力が得られるでしょう[11]。

ここでの `pr` の係数は非常に簡単な公式に従います。まず `x^n` の項の係数に `n!`を乗じます（指数関数の場合、これでだいぶ簡単になります）。PARI の `serlaplace` 機能でこれができます。`ps = serlaplace(pr)` としてみてください。係数は整数になり、目で見てわかりやすくなりました。x^n の係数が n^{n-1} になっています。言い換えると

$$\mathtt{pr} = \sum_{n\geq 1} \frac{n^{n-1}}{n!} X^n$$

ということです。これが証明できますか？(これを初めて見た人には証明は難しいかもしれません。)

PARI ではもちろんベクトルと行列が扱えます（数学的に意味はないとも言えるかもしれませんが、ベクトルでは行と列を区別します)。例として `[1,2,3,4]` と入力してみてください。これは座標が 1、2、3、4 の行ベクトルを与えます。列ベクトルがほしいときには `[1,2,3,4]~`としてください。想像はつくでしょうが、`~`は転置を表します。出力を見比べてもこの二つは、行末のチルダ以外には違いはありません。しかし`\b` と入力してみると、なんとベクトルが列になって表示されます。`\b` は主にベクトルの列表示のために使われます。

`m = [a,b,c; d,e,f]` と入力すると、これは 2 行 3 列の行列を入力したことになります。行列は列ごとに入力され、各列はセミコロン`;` で区切られることを覚えておいてください。行列は、普通に長方形の形で表示されます。入力するときのように 1 行で表示したいときには`\a` と入力し、ずっとその形式で表示させたいときには `default(output, 1)` とします。`default(output, 0)` と入力すると、デフォルトの表示形式に戻ります。

では `m[1,2]`、`m[1,]`、`m[,2]` と入力してみてください。説明はいりませんね？(`m[j,k]` のような形式では、`j` が行番号、`k` が列番号を示します。これらの番号の最初は 1 で、0 ではありません。これは変更できません。)

`m[1,2] = 5; m` と入力してみましょう（セミコロンは 1 行で複数の命令を入力するのに使えます。その場合、行の最後にある命令の出力だけが表示されます)。続けて `m[1,] = [15,-17,8]` を入力してください。問題ないでしょう。そして `m[,2] = [j,k]` としてみると、`m[,2]` が列ベクトルなのに行ベクトルを代入しようとしたことになり、エラーメッセージがでるはずです。代わりに `m[,2] = [j,k]~`とすればうまくいくでしょう。

今度は `h = mathilbert(20)` としてみてください。ヒルベルト行列と呼ばれる、i 行 j 列の要素が $(i+j-1)^{1}$ で表される行列が得られます。この `h` は画面上に表示すると場所をとるので、行末にセミコロン`;` をつけると表示させないようにできます（`h = mathildert(20);` とする)。これは PARI に組み込まれている “precomputed”（生成の際に値が計算される）行列の一例です。しかしこのようなものはほんのわずかしかありません。後でもっと一般的な例について触れます。

[11] 訳者検証では `p` は 17 項+`O(x^18)` で、したがって `x + O(x^18)` になります。`default` すると `seriesprecision = 16 significant terms` です。

ヒルベルト行列で面白いのは、まず第一に逆行列と行列式が陽に計算できること（しかも逆行列の要素が整数になります)、第二に数値計算的にはそれらは非常に不安定で、数値解析ソフトウェアの線形代数パッケージに対する厳しいテストとして用いられることです。PARI ではもちろん問題は生じません。というのは行列要素は有理数で与えられ、演算は厳密に行われ数値演算の誤差は生じないからです。試してみましょう。`d = matdet(h)` と入力すると（複雑な演算を要するので、すこし待つ必要があるでしょう)、結果はもちろん有理数になり、分子が 1，分母が 226 桁になります。ところで、どうやって桁数がわかったと思いますか？ もちろん 226 桁を数えたわけではありません。代わりに `1. * d` と入力しました。こうすると結果は指数表示なり、その指数部がそのまま桁数になります。もっとわかりやすい方法として、`sizedigit(1/d)` というやり方もあります。

次は `hr = 1. * h;` として（セミコロンを忘れないでください。わけのわからないものを見たくはありませんよね)、続けて `dr = matdet(hr)` としてみてください。二つのことに気づくでしょう。一つは計算が一瞬ではないにしろ、有理数で行われる場合よりは早くすむことです[12]。これは有理数が 226 桁あったのに対して、実数演算は 28 桁の精度で行われたことによります。

もう一つはもっと大事なことで、結果が非常に悪いということです。さきほどの正しい計算の結果 `1. * d` と比べると 2 桁しかあっていません。このどうしようもない不安定性は、前述したヒルベルト行列の特徴の一つです。この例での状況はさらに悪く、`norml2(1/h - 1/hr)` と入力してみてください（`norml2` は L^2 ノルムの二乗を返す関数で、たとえば係数の二乗和になります)。`1/h` の計算は `matdet(h)` より（そう多くはないですが）時間がかかるので今度も少し時間がかかるかもしれません。結果は 10^{50} よりも大きくなり、`1/hr` の要素はいくつか、10^{24} ほど値が間違っているものがあります（誤差の最大のものは 15 行 14 列（28 桁の整数）の $7.644 \cdot 10^{24}$ です)。

逆行列に戻したときに正しい値を得るには、精度を 56 桁にする必要があるでしょう（試してみてください)。

ベクトルや行列は各要素を陽に手入力することもできますが、要素が簡単な式で表現できることも多く、この場合は違った方法を使うことができます。たとえば i 番目の要素が i^2 になるベクトルだとすると、長さ 10 のベクトルの場合は `vector(10, i, i^2)` で入力できます。

```
matrix(5,5,i,j, 1/(i+j-1))
```

とすれば 5 次のヒルベルト行列を入力できます（つまり `mathilbert` 機能はなくてもいいということです)。`i` と `j` はダミー変数でそれぞれ行と列を示すために使われます（ベクトルの場合はダミー変数はもちろん一つです)。ベクトルや行列の次元数に加えて、これらの変数を指定することを忘れないようにしてください。

注意：`I` という文字は -1 の平方根である虚数単位として予約されています。したがって変数名として使うことはできません。試しに `vector(10, I, I^2)` と入力してみると、GP は `I` を変数名としては認識できません、というエラーメッセージが表示されます。こういった予約語の変数名は他に二つあります。`Pi` と `Euler` です。同様に関数名になっているものも使えません。一方、`i` や `pi`、`euler` を使う分にはなんの制約もありません。

ベクトルや行列を作るとき、論理演算子や `if()` 文を使うと便利なことがあります（詳しくはプログラミングの章を見てください)。`if` 文は値を持ち、それは `if` 文中で評価した値になります。例を挙げると

```
matrix(8,8, i,j, if ((i-j)%2, x, 0))
```

は `x` と `0` の市松模様になる行列を生成します。ここでベクトルや行列は、使われる前に「生成」さ

[12] 訳者検証では 30 ミリ秒:1 ミリ秒以下、になりました。体感できません。`mathilbert(50)` だと約 2 秒:30 ミリ秒になりました。

れなければならないことに注意してください。たとえば

```
for (i = 1, 5, v[i] = 1/i)
```

といった命令を v が（v = vector(5,j,0) のような方法で）生成される前に行うことはできません。

まだ例に挙げてない PARI で用意された型の最後は、数論に関するものです（数論に興味のない人はとばしてかまいません）。

その最初は「整数剰余」です。例を挙げて考えてみましょう。n = 10^15 + 3 と入力してみてください。これが素数がどうか知りたいとします。PARI にはこれを調べる関数が当然用意されていますが、他の方法でやってみましょう。まず組み込みの素数表で割ることにしますが、ここでちょっとずるをして、厳密にやってくれる factor 機能を使ってみましょう。factor(n, 200000) と入力してください。（引数の後者は factor に割る数の上限を指定し、そこまでで止まるようにします。これを 0 に指定すると、組み込みの素数表にあるものすべてを使います。これはデフォルトでは 500000 までになっています）。

結果は 2 列の行列になり（これはすべての因数分解機能で同じです）、第一列は素数で第二列がべき指数になります。200000 までの素数で n を割り切るものがなければ、行は 1 行だけになります。さらに簡単にやるには、PARI の factor 機能に第二引数を与えずに実行してみることですが、その前に自分で解を得るにはどうするか見てみましょう。

フェルマーの小定理によると、n が素数の時 n で割れないすべての a に対して $a^{n-1} \equiv 1 \pmod{n}$ が成立します。ここで $a = 2$ の場合についてやってみましょう。しかし 2^{n-1} はおおよそ $3 \cdot 10^{14}$ 桁ありますから、とても入力する気になりません。計算させることにしましょう。a = Mod(2,n) とすると、環 $R = \mathbf{Z}/n\mathbf{Z}$ 上の元として 2 を生成します。これは整数剰余と呼ぶ R の元で、常に n よりも小さな数で表現します。したがって非常に小さい数と言えます。フェルマーの定理は R 上で a^{n-1} =Mod(1,n) と書き直すことができて、非常に効率的に計算できます。a^(n-1) と入力してみてください。まるで n が素数ではないことを証明するかのように、Mod(1,n) とは異なる結果になります。（一方でもし結果が Mod(1,n) になっていたら、n が素数である可能性を示すヒントになるかもしれませんが、決して証明にはなりません）。

因数を見つけるのはまた別の話です。次々と割って試していくよりももっと楽な方法があるでしょう。この例題を終わらせるために、fa = factor(n) として因数を見てみましょう。もっとも小さな因数は 14902357 です。割ってみて試していたのではとても耐えられませんね。

素数を返す機能を使う上で気をつけなければならないことは、factor 機能で返される素数は強擬素数であり、真の（証明された）素数ではないということです。厳密に素数かどうか確認したいときには isprime(fa[,1]) を使ってください。後者のコマンドは isprime を fa の第一列のすべての要素、たとえばすべての擬素数に適用し、列ベクトルを結果として返します（それがすべて 1 ならその擬素数は素数です）。すべての数論的関数ではこういった方法でベクトルや行列の要素を適用できます。

二つ目の数論的な型は p 進数です。ここで定義を述べる余裕はないので、この悪魔のような型に用がなければ読みとばしてかまいません。p 進数は p が素数、n が p 進数での精度の桁数のとき、O(p^n)（n= 1 のときは単に O(p)）を加えた有理数または整数として入力できます。普通の数論的な演算と違って、超越関数を適用することができます。たとえば n = 569 + O(7^8)、続けて s = sqrt(n) とすると、n の平方根のうちの一方が得られます（確認したければ、s^2-n でできます）。ここで l = log(n)、続けて e = exp(l) としてみましょう。p 進数での対数を知っていれば、e が n にならないのを見ても驚くことはないでしょう。(n/e)^6 と入力してみましょう。e

は n と 1 の $p-1$ 乗根の積になります。

ついでに、p 進数の値 n から整数 569 に戻りたいときには、lift(n) または truncate(n) としてください。

三つ目の数論的な型は「二次数」型です。この型は基礎体（$\mathbf{Q}$ など）を二次に拡張したときに使いやすいようにうまく設計されています。これは複素数型の一般型です。使ってみるためにはまず、対象となる二次数体を指定する必要があります。これは quadgen 機能を二次数体の「判別式」d に（被開数に対して）適用することで行います。すると結果は w である、というふうな表示がされますが、数値を返します。これは $(\mathrm{d}+a)/2$ で、ここで a は d が偶数の時 0、奇数の時 1 をとる数です。quadgen の振る舞いは少し変わっていて、その結果はいつも w として表示されますが、変数 w の値には反映されません。だから w という名前の変数を使うときでも機械的に w = quadgen(d) とする必要があります（複数の二次数体を使うときには w1 のようにしてください）。混乱しがちなので注意してください。

では w = quadgen(-163)、続けて charpoly(w) と入力して w の特性方程式を求めてみましょう（こうすると変数 x に関して行います。charpoly(w, y) とすると、subst で置き換えをすることなく他の変数に関して直接行うことができます）。w がどう表されるかが出力されます。1.*w とすると二次式の根のどれがとられているかを知ることができますが、これが必要になることはまれでしょう。ここまですることで、体 $\mathbf{Q}(\sqrt{-163})$ 上でいろいろな演算を行うことができるようになります。w^10、norm(3 + 4*w)、1 / (4+w) と入力してみましょう。さらに a = Mod(1,23) * w、続けて b = a^264 とするともっと面白い例が見られます。これは二次数体上でのフェルマーの定理の一般化です。これ以上は 23 を法として見なくてもいいと思ったら、lift(b) としてください。

別の例として p = x^2 + w*x + 5*w + 7、続けて norm(p) を試してみましょう。すると $\mathbf{Q}$(w) 上で p によって定義される二次拡張に対応する $\mathbf{Q}$ 上の 4 次方程式が得られます。

また、wr = sqrt(w^2) と入力してみてください。w になるわけではありません。。sqrt は代数的な関数ではありますが、超越関数として扱われ数値解を返します。algdep(wr,2)（これは w の二乗に関連のある代数関係の関数に見えますね）と入力してください。これがもう一度 w を得る方法の一つです。同様に algdep(sqrt(3*w + 5), 4) でもかまいません。ユーザーマニュアルの algdep のところを見てください。

四つ目の数論的な型は「多項式剰余」、つまり多項式を多項式で割った剰余です。一般に代数拡大体、たとえば数体の元（基礎体が $\mathbf{Q}$ の場合）や有限体の元（基礎体が素数 p に対して整数剰余で定義される $\mathbf{Z}/p\mathbf{Z}$ の場合）における演算を扱うときに用いられます。二次数体型の一般化と思っていいでしょう。使い方は整数剰余と同じです。たとえば w = quadgen(-163) とする代わりに w = Mod(x, x^2 - x + 41) と入力します。そして二次体の場合と全く同様に w^10、norm(3 + 4*w)、1 / (4+w)、a = Mod(1,23)*w、b = a^264 と試してみると、同じ結果が得られるでしょう（lift(...) としておけば、毎回表示される多項式 x^2 - x + 41 を消すことができます）。ここではもちろん、二次だけではなく任意の次数で演算が行えることに、基本的には興味があるわけです。（二次数の場合よりも二次式の場合の方が、それぞれ対応する演算には当然時間がかかります）。

しかし、振る舞いには若干の違いがあります。多項式剰余 w が上で求められたままである時に 1.*w としてみてください。このときは結果は同じにはなりません。sqrt(w) とすると要素数 2 のベクトルが得られ、それらは $\mathbf{C}$ 上にありえるすべての w の埋め込みの二乗根の第一分枝になっています（二乗根ではありません）。一般に w が n 次であれば要素数 n のベクトルが得られ、超越関数の場合も同様です。

いろいろと普通の算術関数を見てきましたが、他にもいくつかあります。三次の拡大を定義して

みましょう。a = Mod(x, x^3 - x -1) と入力してください。するとたとえば b = a^5 などを調べてみることができます。ここで a を b の多項式として表したいとします。b は同じ体の上でも大きいのでこれは可能です。modreverse(b) とするとしてみてください。これにより同じ体の上で新たに定義される多項式が得られ（たとえば b の特性多項式)、この新しい多項式剰余における a を表します。後述する数体の章でさらに詳しく見てみましょう。

4 基本的な算術機能 - Elementary Arithmetic Functions

PARI は数論の研究者を対象に作られていて、したがって算術関数が非常に多く用意されていても驚くことではありません（ユーザーマニュアルのそれぞれ対応する章を見てください)。たとえば factor など、そのうちのいくつかはすでに見てみました。factor は整数だけでなく、(一変量の) 多項式も扱えます。例として factor(x^15 - 1) と入力してみてください。素数 p を法とする多項式の因数分解もできます（factormod)。また素数体ではなく有限体上でも可能です（factorff)。

実際、GCD（gcd)、拡張 GCD（bezout)、中国の剰余定理（chinese）などを計算する機能があります。

これらの因数分解に関する機能に加えて、isprime、ispseudoprime、precprime、nextprime などの素数判定に関する機能もいくつかあります。すでに述べたように、後二者は強擬素数のみを生成します（ispseudoprime によるテストをパスするものです)。isprime はもっと洗練されたテストを行いますが、デフォルトではこれは使われません。

また乗法的な算術関数もあります。メビウスの μ 関数 (moebius)、オイラーの ϕ 関数 (eulerphi)、ω および Ω 関数（omega、bigomega)、与えられた整数の正の約数の k 乗の和を計算する σ_k 関数 (sigma) などです。

連分数の計算もできます。\p 1000、続けて contfrac(exp(1)) とすると、非常にシンプルなパターンを繰り返す、自然対数を底とする連分数が得られます。(証明できますか？)

ある算術的な条件下でのみ、特定の処理をしたいような場合が多々ありますが、GP では以下のような機能が用意されています。isprime は上述しました。issquare、isfundamental はある整数が基本判別式であるかどうかを調べ（たとえば 1 または二次体の整数環の判別式であるかどうか)、forprime、fordiv、sumdiv はそれぞれ処理を繰り返して行います。仮に整数 n のすべて約数の積を計算したいとすると、一番簡単な方法は

```
p=1; fordiv(n,d, p *= d); p
```

とすることです（これは非常に簡単な式です。証明してみましょう)。p *= d という命令は（C 言語のやり方にあわせてありますが)、p = p * d と同じ処理を行います。

さて、1000 以下でその原始根の剰余が 2 になるような素数 p を求めたいときは、一つには以下のような方法があります。

```
forprime(p=3,1000, if (znprimroot(p) == 2, print(p)))
```

ここでは znprimroot が原始根のうち最小のものを返すことを仮定していて、この場合は実際にそうなることに注意してください。そうなるかどうかわからない場合は、

```
forprime(p=3,1000, if (znorder(Mod(2,p)) == p-1, print(p)))
```

とする必要があります（この場合、該当する次数を持つ元をすべて調べて生成元を探す代わりに、$\mathbf{Z}/p\mathbf{Z}$ 上で二次の場合だけを計算するので計算時間は短くなります)。

二次数体関連の機能（双二次形式と一般の数体に関するもの）は次の章でふれます。

5 線形代数 - Performing Linear Algebra

様々な線形代数プログラムがもちろん使えますが、さらに **Z** 上の線形代数、たとえば束（lattice）上での演算も行えます。どのように使うのか見てみましょう。まず最初に**列**ベクトルで表されるベクトルの集合（基底であることが多い）を生成してベクトル空間（一般に言うと加群）として与えます。この集合はまた行列としても表され、ここでは、PARI での列ベクトルの行ベクトルによる行列とします。基礎体（または基礎環）は PARI で扱えるものならなんでもかまいません（p 進数は現在のところ、線形代数パッケージではうまく扱えないので除きます）。演算子の多くは基礎体が **Z** のときのために用意されていますが、基礎体が実数または複素数体の場合に使えるものもあります。

続きはそのうち...

6 超越関数 - Using Transcendental Functions

初等超越関数はすべて、また高度な超越関数もいくつか（ガンマ関数、不完全ガンマ関数、誤差関数、指数積分、K 次のベッセル関数、合流型超幾何関数、リーマンのゼータ関数、ポリログ（polylogarithm）、ウェーバー関数、シータ関数）が使えます。他にも必要に応じて説明します。

これらの関数ではデフォルトの精度が本質的な役割を担います。超越関数においてはほとんどの場合、引数が厳密な値で表現されているときは、結果はデフォルトの精度で計算されます。そうでないときは、引数の精度で計算が行われます。しかしこの場合は、**表示されている値**が実数形式になることに注意してください（普通はデフォルトの精度と同じです）。この章では 32 ビット整数のコンピュータ上で行うことを想定します。もしそうでない場合は、以下の内容をどう修正する必要があるかを考えてみると非常にいい演習になるでしょう。

デフォルトの精度 28 桁になっているとしましょう（32 ビットの計算機で GP を起動するとこうなっています）。`e = exp(1)` と入力してみてください。`e = 2.718...` と 28 桁の数字が表示されます。この出てきた数値がどの程度正しいのか確認してみましょう。大変で（しかし妥当な）方法は以下のようにすることです。精度を実際に意味のある程度に多く（たとえば`\p 50` などと）します。そして `e`、`exp(1)` と続けて入力してください。最後に表示された定数 e の値は正しく 50 桁得られますが、`e` は 28 桁で計算された値です。この二つの数値は、上位 29 桁が一致します。

得られた計算結果が何桁あるのかを知るには、`length(e)` としてみます。これを見ると、仮数部がぴったり 3 ワード[13]であることがわかるでしょう。$3\ln(2^{32})/\ln(10) \approx 28.9$ なので、上位 28 桁または 29 桁ということになります（32 ビット計算機の場合）。

精度を 28 桁に戻してください（`\p 28` と入力）。ここで `exp(1.)` とすると 28 桁に戻っているのが分かります。しかし `f = exp(1 + 10.^(-30))` としてみると、精度が 28 桁に指定されているにもかかわらず、上の式では 59 桁あることが上に挙げた二つの方法のうちのどちらかを使うことでで分かります。これは、まず `1 + 10.^(-30)` が PARI の哲学的方針、つまり可能な限り高い精度を保つことにしたがって計算されたことによります。つまり `10.` は精度 29 桁ですが、`1` は精度「無限」で、`1 + 10.^(-30)` は小数点以下 $59 = 29 + 30$ 桁の精度を持つということです。f も同じです。

では `cos(10.^(-15))` と入力してみてください。結果は 1.0000... と表示されますが、すぐわかるように結果は厳密に 1 にはなりません。`length(%)` とすると仮数部が 7 ワードあり、つまり 67

[13]訳注：1 ワードは 32 ビット計算機では 32 ビットという解釈です。ここでは 3 words = 96 bits = 12 bytes です。

桁の精度があり得ることが分かります。実際には（精度を 100 桁にしてみるとわかりますが）正しいのは 60 桁です。 PARI は可能な限り桁数を多くしようとしますが、6 ワードでは 57 桁しか表現できないので、7 ワード使ったということです。それにしても、なぜこの結果はこんなに正確なのでしょう。それは前述の例と同じ理由で、x が 1 に近い値になると $\cos(x)$ は $1-x^2/2$ に近くなり、桁数もこの値、つまり $1-0.5*10^{-30}$ になっていきます。ここで $0.5*10^{-30}$ は 28 桁の精度と考えられ、したがって結果は $20+30=58$ 桁になります。

しかし、PARI の哲学的方針はどこまでも貫けるわけではありません。たとえば `cos(0)` と入力したときには、GP は厳密な 1 を返べきです。しかし超越関数を扱うときには決して厳密な値を扱わないとすることが妥当であり、GP は現在の桁数より多い仮数部を使って 1.000... と表示します。

他の超越関数も使ってみましょう。`gamma(10)` と入力してみましょう。なんの問題もないはずです（`9!`で確認しましょう）。次は `gamma(100)` としてみましょう。今度は指数表示で結果が出てきます。これはデフォルトの桁数では小さすぎるからです（`99!`でちゃんとした答えを見てみましょう）。ちゃんとした答えを得るためには、二通りのやり方があります。最初の方法はごく自然に、精度の桁数を増やすことです。`gamma(100)` は 156 桁を要するので（余裕を見て）、`\p 170` としてからもう一度 `gamma(100)` としてみましょう。今度はちゃんと完全な結果が表示されます。

しかしこのやり方を続けるのは、あまりよくありません。デフォルトの精度に戻してください（`\p 28` と入力）。ガンマ関数は値の増加が急激なので、普通はその対数を使います。`lngamma(100)` と入力してください。今度は見やすい値で表示され、結果は `log(99!)` と厳密に同じになるでしょう。

では `gamma(1/2 + 10*I)` を試してみてください。複素数のガンマ関数が問題なく計算されます。では、

```
t = 1000; z= gamma(1 + I*t) * t^(-1/2) * exp(Pi/2*t)/sqrt(2*Pi)
```

と入力して、続けて `norm(z)` としてみてください。`norm(z)` はスターリングの公式にしたがって 1 に非常に近い値になります。

次はリーマンのゼータ関数を試してみましょう。まずタイマーをセットしてください（`#`と入力）。そして `zeta(2)`、`Pi^2/6` と入力してください。正しく計算されているようですね。では `zeta(3)` はどうですか。どれもほとんど時間がかかりません。しかし `zeta(3.)` になるとどうでしょう。結果は同じになりますが、かなり時間がかかるでしょう（もし計算機の性能がよくて違いがわからない場合は、精度の桁数を上げてみましょう[14]）。これは `n` が正または負の整数のとき、PARI では `zeta(n)` を計算するのに別の方程式を使っているからです。

`zeta(1 + I)` を試すと、やはりうまくいきます。ここで試しに `zeta` の一番目の複素数零点を計算する素朴な方法をやってみましょう。これが $1/2+i*t$ で表され、t が 14 と 15 の間であることは知られています。これを以下の命令を順に行うことでできます。しかしこれを直接入力する代わりに、たとえば `zeta.gp` という名前のファイルに書いておき、GP 上で`\r zeta.gp` として読み込ませてみましょう。

```
{
  t1 = 1/2 + 14*I;
  t2 = 1/2 + 15*I; eps = 10^(-50);
  z1 = zeta(t1);
  until (norm(z2) < eps,
    z2 = zeta(t2);
    if (norm(z2) < norm(z1),
```

[14]訳注：Apple 製/PowerBook G4(867MHz、640MB) では、`p 500`、`zeta(3.)` で 1060 ミリ秒でした。

```
      t3 = t1; t1 = t2; t2 = t3; z1 = z2
    );
    t2 = (t1+t2) / 2.;
    print(t1 ": " z1)
  )
}
```

括弧を付けるのを忘れないでください。これによって、一連の命令が複数の行に分かれていることを GP に示します（もっと簡単には、行末に\を書いておくとパーザ（訳注：GP の構文解析ルーチン）が、入力がまだ続いていることを認識します)。また、拡張子.gp は入力しなくても GP が付け足してくれます（拡張子はつけてもつけなくてもかまいません[15]。自分で作った GP スクリプトを他のファイルと区別しやすくするために、つけておくと便利だということです)。

これで 25 桁の精度で第一零点が得られます。

このチュートリアルの最初でふれたように、p 進数で使える超越関数もあります。そろそろこれらに馴染んでみてもいいでしょう。`a = exp(7 + O(7^10))`、`log(a)` と入力してください。うまくいきますね。では `b = log(5 + O(7^10))`、`exp(b)` と入力してください。なにかおかしいですね。もとの値に戻らないのでしょうか? そんなことはありません。`exp(b) * teichmuller(5 + O(7^10))` としてみてください。`teichmuller(x)` 機能は Teichmüller 指標で、`x` を p 進数の単数としたときに、p を法として `x` と合同な原始 $p-1$ 乗根は一つしかないということです。

ここでちょっと実数に戻ってみましょう。`agm(1,sqrt(2))` は 1 と $\sqrt{2}$ の算術幾何平均（arithmetic-geometric mean、AGM）を返し、これが（完全）楕円積分の基本的な計算方法になります。

```
Pi/2 / intnum(t=0,Pi/2, 1 / sqrt(1 + sin(t)^2))
```

とすると同じ結果が得られます。基本変換 `x = sin(t)` は若い頃のガウスの重要な発見、

$$\int_0^1 \frac{dx}{\sqrt{1-x^4}} = \frac{\pi}{2\mathrm{agm}(1,\sqrt{2})}$$

という等式を与えます。

`2 * agm(1,I) / (1+I)` とすると、複素数の算術幾何平均も計算できることが分かりますが、その定義に注意しなければなりません。返された結果は前のものとほとんど同じですね。なぜだか考えてみてください。

最後に `agm(1, 1 + 7 + O(7^10))` をやってみてください。p 進数の算術幾何平均が得られます。一般に p 進数の二乗根は同じ p 進数の体の上にくるとは限らないので、すべての p 進数算術幾何平均が計算できるわけではありません。また $p=2$ のときには、`a/b` が（8 ではなく）16 を法として 1 と合同なときにだけ `agm(a,b)` が計算できるという、合同制約があります。

そこで`?3` と入力してみましょう。すべての超越関数のリストが表示されます。ここでこれ以上例を挙げる代わりに、このリストをみていろんな関数を試してみることをお勧めします。各関数について整数、実数、複素数、p 進数を与えてみてください。しかし実装されてない場合もあります（または定義上、引数としてとらない型もあります)。

7 数値計算 - Using Numerical Tools

PARI は数値計算用のパッケージとして作られたわけではありませんが、それでもできる数値計算もあります。以下の章で線形代数と多項式の計算を見ていきましょう。

[15]訳注：`zeta.gp` でなく `zeta` でもいいということです。

$\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots)$ がべき級数から `atan(x)` に書き直せることはよく知られています。またこの公式は収束が非常に遅くて、π を実際に求めるのには使えないことも同様に知られています。これは本当でしょうか？ そんなことはありません。`\p 100`（単によく見てみるためだけです）として `4 * sumalt(k=0, (-1)^k/(2*k + 1))` を見てみてください。すこし待つと（実は単に `Pi` とすればいいんですが）π の値が 100 桁でてきます（`Pi` で確かめてみてください）。PARI のバージョン 1.38 ではこの方法は、オイラーの交代和の加速法とオランダの数学者ヴァインハーデン（Wijngaarden）のプログラミング法を組み合わせたものでした（数値計算の手法の本などに説明があります）。現在用いられている手法はこれよりは良いと思われるもので、ヴィレガス（F. Villegas）、ザギエル（D. Zagier）、コーエン（H. Cohen）のもの[16]との組み合わせです。

同様に`\p 28` として（こうしなければ計算に非常に時間がかかります）`sumpos(k=1, 1 / k^2)` を試してみましょう。やはり収束は遅いのですが、同じような工夫で項が正のときは級数の計算ができます（`Pi^2/6` で厳密な値を計算したときと比べてみましょう）。こういった時間のかかるやり方はあまり興味を引きませんが、便利なときもあります。

`sumalt` を使えば発散級数も計算できます。

```
zet(s) = sumalt(k=1, (-1)^(k-1) / k^s) / (1 - 2^(1-s))
```

と入力してみましょう。`s` の値が 1 ではないとき、`zet(s)` は `zeta(s)` と同じになり、遅いながらも級数は収束していきます（`sumalt` は速度は度外視です）。`s` が負のときは級数は発散しますが `zet(s)` は正しい結果を返します。`zet(-1)`、`zet(-2)`、`zet(-1.5)` などとして `zeta` 関数の値と比べてみましょう。しかしあまり遊びすぎては行けません。たとえば `zet(-14.5)` とすると間違った結果になります。

`zet(I)` と `zeta(I)` を比べてみてください。複素数（の一部）でも計算はできますが、級数はもう交互にはなりません。

同様に、`sumalt(n=1, (-1)^n/(n+I))` を試してみてください。

もっと古典的な機能に数値積分があります。たとえば `intnum(t=1,2, 1/t)` とすると非常に早く計算され、log(2) が 26 桁得られます。第 3 章[17]に他の数値積分機能も説明があります。

PARI ではまた複素数型も積分に使えます。たとえば $h(z) = e^z - z$ の零点がどこにあるのか知りたいとします。コーシーの定理を使えば、原点から半径 r の円盤内の零点の数が

$$\frac{1}{2i\pi}\int_{C_r}\frac{h'(z)}{h(z)}dz$$

で表されます（ここで C_r は原点を中心とする半径 r の円）。以下を入力してみてください。

```
fun(z) =
{
  local(u);
  u = exp(z);
  (u-1) / (u-z)
}
zero(r) = r/(2*Pi) * intnum(t=0, 2*Pi, fun(r*exp(I*t)) * exp(I*t))
```

（ここで `u` は関数 `fun` 内の局所変数です。関数が呼び出されたときに GP が与えられた実引数の値で関数の仮引数の値を設定し、他の宣言されているパラメータと局所変数の値を 0 にします。そし

[16] Cohen, Villegas, Zagier. “Convergence Acceleration of Alternating Series”, Experimental Mathematics, vol. **9**, No. 1, 3-12. (2000)

[17] ユーザーマニュアルの Chapter 3、3.9 Sums, product, integrals and similar functions

て関数から抜けるときに以前の値に戻します。しかしもし関数のプロトタイプ宣言のときに u の宣言をしなかったら、その値は変更されたままになります[18]。すべてのパラメータをこの方法で宣言しなければならないわけではありませんが、この副作用には注意しましょう)。

これにより、zero(r) で零の個数を数えることができます（単純に変数 $z = r * \exp(i * t)$ を書き換えただけ です)。

では\p 9 として（そうしないと計算時間が非常に長くかかります。それに結果が整数であることは分かっているので)、zero(1)、zero(1.5) としてみてください。単位円の中には零点がなく、絶対値が 1 と 1.5 の間には二つある（必然的に複素共役になりますが）ことが分かります。ではそれらを計算してみましょう。z をそういう零点とし、 x と y を実数として $z = x + iy$ とします。$e^z - z = 0$ という式は簡単な変換で $e^{2x} = x^2 + y^2$ になり、$e^x \cos(y) = x$ と書けます。また $y = \pm\sqrt{e^{2x} - x^2}$、$e^x \cos(\sqrt{e^{2x} - x^2}) = x$ となります。そして以下を入力してください。

```
fun(x) =
{
  local(u);
  u = exp(x);
  u*cos(sqrt(u^2 - x^2)) - x
}
```

すると fun(0) は正で fun(1) は負の値になります。ここで精度を 28 桁に戻して

```
x0 = solve(x=0,1, fun(x))
```

とすると x の値がすぐに求まります。そして

```
z = x0 + I*sqrt(exp(2*x0) - x0^2)
```

とすると求める零点が（その共役複素数とともに）得られます。exp(z) - z、また abs(z) として確認してみましょう。

円よりも複雑な等高線上でも積分は可能ですが、線分上の積分計算の回数を減らすために、上の例で示したような変数の変換をそのたびに行う必要があります。

上に上げた例でも solve 機能を使いました。複素数に solve 機能を使う場合は、問題を実数の問題に還元する必要があります。上述のリーマンのゼータ関数の第一零点を求める場合に

```
solve(t=14,15, real(zeta(1/2 + I*t)))
```

とすると[19]t=14 および t=15 の場合に実部が正になるため、正しい結果になりません。そうなったときは虚部ではうまくいっています。

```
solve(t=14,15, imag(zeta(1/2 + I*t)))
```

とするといいでしょう。または探索範囲を狭くするために

```
solve(t=14,14.2, real(zeta(1/2 + I*t)))
```

としてもかまいません。

8 多項式とべき級数 - Functions Related to Polynomials and Power Series

まず、不注意にならないように言っておかねばなりませんが、「厳密（exact)」と「厳密でない (inexact)」には決定的な違いがあることをよく理解しておいてください（第 1 章参照)。多項式を扱う場合にはこれが非常に重要です。それでは早速やってみましょう。

[18]訳注：関数の外で使っている変数 u の値も変更されるということです。
[19]訳者検証ではエラーになります。“roots must be bracketed in solve.”

```
gcd(x^2 - 1, x^2 - 3*x + 2)
```

は特に問題なく計算でき、結果は予想通り `x - 1` になります。しかし

```
gcd(x^2 - 1., x^2 - 3.*x + 2.)
```

とすると、運のいいことに結果はおおよそ正しく出てきますが、因数として出てきた 3 は PARI で採用している計算方法のおかげで、おかしな具合になっています。しかしどんな場合でも、本質的にはこれが適切な表示なのです。`gcd(x - Pi, x^2 - 6*zeta(2))` は `x - Pi` と等しいはずなのに PARI は π を数値で返します。これは厳密でない値を含む多項式では GCD は意味がないからです。また `polresultant(x - Pi, x^2 - 6*zeta(2))` なら、その結果はほとんど 0 になりますが、自明ではなく、結果を見てもそれがなんなのかよくわかりません。したがってここでは、多項式（とべき級数）は厳密な値を持つものだけを扱うことにします。

`pol = polcyclo(15)` と入力してください。これは 15 次の円周等分多項式を与え、次数 $\varphi(15) = 8$ になります。ここで `r = polroots(pol)` としてください。与えられた多項式の 8 個の複素根が 28 桁の精度で得られます。`\b` とすると見やすくなります。共役複素根の対がランダムに並んで表示されます。一方で `polroots` は、実数根は昇順に表示します。

入力した多項式 `pol` の根は単数の原始 15 乗根の定義により与えられます。これを確認するに単に `rc = r^15` と入力すると、なぜかエラーメッセージが表示されます。これは当然、ベクトルのかけ算はこんなに簡単にはできないからです（もちろんべき乗もできません）。しかし末尾に `.` をつけて `rc = r^15.` としてみるとうまくいきます。非整数（ここでは実数）が指数のべき乗は超越関数で、ここでは一項ずつ計算されるからです。15. という実数は、数学的には完全に整数で表現可能ですが、ここではそれはなんの意味もありません[20]。

この例では結果が非常に 1 に近いものでした。しかしこれらの実部と虚部の値をすべて見るのはめんどうなことではあります。もっと増えるととても不可能でしょう。というわけでこれを自動でやってみましょう。`rr = round(real(rc))`、`sqrt(norml2(rc-rr))` とすると、`rr` が全部 1 で `rc - rr` の L2 ノルムが 10^{-27} になり、精度 28 桁での計算では妥当な値であることが分かります。`norml2` 機能が見た目とは違って、L2 ノルムではなくその二乗を返すので、その平方根をとらねばならないことに注意してください（この場合は必要はありませんが）。

今度は `pol = x^5 + x^4 + 2*x^3 - 2*x^2 - 4*x -3` として `factor(pol)` とすると、`pol` が $\mathbf{Q}$（または $\mathbf{Z}$）上で二つの因数に分解されます。また `fun(p) = factorpadic(pol,p,10)` と入力すると、精度 10 桁で $\mathbf{Q}$ 上で p 進数で `pol` を因数分解する関数 `fun(p)` を定義できます。ここで `factor(poldisc(pol))` とすると判別式を割り切る素数として 11、23、37 が得られます。`fun(5)`、`fun(11)`、`fun(23)`、`fun(37)` とすると他の因数分解を見ることができます。

同じように、`lf(p) = lift(factormod(pol,p))` として `lf(2)`、`lf(11)`、`lf(23)`、`lf(37)` としてみましょう。それぞれ異なる $\mathbf{F}_p$ 上での因数分解が得られます。しかしこれは以下のようにやるのがよりいいでしょう。

```
pol2 = x^3 + x^2 + x - 1
fq = factorff(pol2, 3, t^3 + t^2 + t - 1)
centerlift(lift(fq))
```

これは θ を多項式 `t^3 + t^2 + t - 1` の根とする有限体 $\mathbf{F}_3(\theta)$ 上で多項式 `pol2` を因数分解します。これは当然 $\mathbf{F}_{27}$ を構成します。$\mathrm{Gal}(\mathbf{F}_{27}/\mathbf{F}_3)$ がフロベニウス準同形写像 $u \mapsto u^3$ で生成される次数 3 の輪体であり、上で得られた根で `t` 上でのフロベニウス準同形のべき乗が行えることが知ら

[20]訳注：要するに `.` がついてると厳密な値ではなくなるということです。

れています（これがなんのことだか分からないなら、もっと例をやってみてください。理解するのはそう難しいことではありません）。

また同様に、pol3 = x^4 - 4*x^2 + 16、fn = factornf(pol3,t^2+1) としてみると、多項式 pol3 の t^2+1 で、たとえば $\mathbf{Q}(i)$ 上で定義される数体での因数分解が得られます。わかりやすく見るには lift(fn) とします。t がその数体での生成元であることを思い出してください（この場合は $i = \sqrt{-1}$ と同じことです）。

あまり勧められませんが、多項式と数体とで同じ変数名を使うこともできます。試しにやってみましょう。fn2 = factornf(pol3, x^2 + 1) とすると、正しい答えが得られます[21]。しかしここで作られた PARI 変数はあまりよくありません。たとえば lift(fn2) とつづけてやってみると、なにか x*x と入力したかのようなおかしな記号が出力されます。これは PARI ではダミー変数 x と明示的に指定した変数 x が別のものとして扱われるからです。lift するときにはかならず表示されるので、それを見てください。

まとめると、因数分解は整数に加えて $\mathbf{C}$ 上および $\mathbf{R}$ 上（これは polroots の機能）、$\mathbf{F}_p$ 上（factormod の機能）、 $\mathbf{F}_{p^k}$ 上（factorff の機能）、$\mathbf{Q}_p$ 上（factorpadic の機能）、$\mathbf{Q}$ または $\mathbf{Z}$ 上（factor の機能）、数体上（factornf と nffactor の機能）で行えます。さらに factor は、どの環で因数分解するかを賢く推測します。pol = x^2+1 として factor に pol、pol * 1.、pol * (1+0.*I)、pol * Mod(1,2)、pol * Mod(1,Mod(1,3)*(t^2+1)) などを与えて試してみてください。

現在のバージョンでは上に挙げた環以外では因数分解はできません。たとえば多変数の多項式の因数分解はできません。

因数分解に関する他の機能に、padicappr、rootsmod、polrootspadic、polsturm があります。少し試してみてください。

ではここで polsym(pol3,20) と入力してみてください。pol3 は上の例で入力したものと同じものです。これは $k = 20$ までの pol3 の根の k 乗の和を、もちろん polroots 機能ではなくニュートンの公式を使って計算します。べき数が奇数の項の和は零になり（これは多項式が偶数なので自明です）、符号は繰り返しパターンにしたがい、絶対値は 2 のべき乗になっているように見えます。この場合は実際そうなります。証明してみて、（非有理な）代数的な数を含まない k 次の対称的なべき乗の公式を見つけてみてください。

もうすこしべき級数を見てみましょう。はじめにも少しやってみましたが、

```
8*x + prod(n=1,39, if(n%4, 1 - x^n, 1), 1 + O(x^40))^8
```

と入力してみましょう。x のべき数が偶数の項のみのべき級数が得られます。これは意外ではありますが、モジュラー形式を使えば証明できます。ここで行った積の計算では、積の初期値を単に 1 にするのではなく 1 + O(x^40) にしました。そうしなければ計算は多項式そのままについて行われ、非常に時間がかかり、また計算を n=39 で止めてしまいたいときには x^40 以上の係数は不要なので、非実用的になってしまいます。

モジュラー形式（これは一般の関数のテイラー展開と並んで、べき級数が大活躍するところです）上での演算を行うとして、\ps 122 としてください（default(seriesprecision, 122) の代わりにこう入力できます）。そして d = x * eta(x)^24 とするとラマヌジャン（Ramanujan）のモジュラー判別式関数 Δ の（フーリエ）級数展開が 122 項出力されます。その係数はラマヌジャンの τ 関数の定義から与えられ、数々の美しい性質を持っています（詳しくはモジュラー形式の本を見てください）。ここではこれが 2 を法とするとしましょう。d%2 としてください。PARI はべ

[21] 訳者検証ではエラーになります。“incorrect polynomial in rnf function.”

き級数の係数（そういう意味では多項式の係数）をすぐには約してくれませんね[22]。小さなプログラムを書けばできるでしょうか？ そうです。代わりに `lift(Mod(1,2) * d)` と書けば魔法の呪文のようにうまくいきます。

結果に見られるパターンははっきりしています。もちろん証明はされていませんが（モジュラー形式については Antwarp III[23]を読むか、または近くのモジュラー形式の達人に聞いてください）。同様に `centerlift(Mod(1,3) * d)` とすると今度はすぐには分かりませんが、パターンがあります。Antwerp III を参照してください。

9 楕円曲線 - Working with Elliptic Curves

それではここで、もっと分かりにくいものに向かい合うことにしましょう。後で説明する数体を使うことになりますが、まずその初期化から始めます。これは、数多くのデータが繰り返し必要になることがあり、一度用意しておいて後でずっと使うのが便利だからです。それでは、`ellinit` で初期化を行いましょう（どんな数体を使うのか想像してみてください）。

`e = ellinit([6,-3,9,-16,-14])` と入力してください。これにより、以下のアフィン方程式で定義される楕円曲線のことをいろいろと計算します。

$$y^2 + 6xy + 9y = x^3 - 3x^2 - 16x - 14$$

この式に含まれる数字がいったい何なのかはよくわかりませんが、とにかく入力すべき係数だということははっきりしています。このよくわからないもの（数体の場合はもっとわかりにくいでしょう）から意味のある情報を得るには、メンバー関数と呼ばれる機能を使うといいでしょう。`?.` とするとその一覧が表示されます。一覧の右側に `ell` が表示されたものはどれも、`ellinit` で出力されたものに適用することができます（ほかの `init` 機能でもどうやって使うか、もうわかりますね？ しかし `clgpinit` は用意されていません[24]）。

それでは試してみましょう。判別式 `e.disc` が 37 になるのを見てみてください。するとこの曲線の導手は 37 ということになります。一般にはもちろんこのように自明ではありません。この場合、曲線の方程式は最小ではないので、`r = ellglobalred(e)` としてみましょう。一番目の要素 `r[1]` を見ると、導手の 37 が現れています。二番目の要素は 4 次元のベクトルで、これにより最小の方程式を得ることができます。単純に `e = ellchangecurve(e, r[2])` としてみましょう。`e` はこれで対応するデータによる最小の方程式になります。しばらくの間、三番目の要素は気にしないでください（気になる人のために説明すると、これは局所タマガワ数 c_p です）。

計算し直した `e` は最小の方程式が $y^2 + y = x^3 - x$ であることを示しています。ほかにも `e` について少し見てみましょう。`q = [0, 0]` として曲線上の点を指定します（`ellisoncurve(e, q)` として確認してみてください）。`q` は捩れ点（torsion point）のようにも思えますが、`ellheight(e, q)` として `q` の正規ネロン-テイト高さ（canonical Neron-Tate height）を計算してみましょう。これは零にならず、したがって `q` は捩れ点ではないことになります。他のもっといいやり方として、

```
for(k = 1, 20, print(ellpow(e, q,k)))
```

ともできます。この点の大きさが特性放物線にしたがって爆発的に大きくなるのがわかります。`ellheight(e, ellpow(e, q,20)) / ellheight(e, q)` とするとそれが $400 = 20^2$ になることが

[22]訳注：つまりエラーになるということです。‘‘`forbidden division t_SER % t_INT.`’’

[23]Volume 3 of the Proceedings of the International Summer School on “Modular functions of one variable and arithmetial applications” だと思います。

[24]訳注：clgp は class group 古典群だと思われます。

わかります。ellorder(e, q) とすると結果が 0 になり、これによっても q が捩れ点でないことがわかります。

名前に ell がついた機能が、最初に引数にユーザーが定義した楕円曲線をとることに注意してください。これは数体についても同じです。*ob* がついた機能を使うときには、 *ob*–init で初期化されたものを最初の引数にする必要があります。逆に言えば、初期化をちゃんとしておかないとこういった高度な機能は使えないということです。

では他の曲線も試してみましょう。e = ellinit([0,-1,1,0,0]) と入力してください。これは $y^2+y=x^3-x^2$ という楕円曲線です。判別式から導手が 11 であることがわかり、ellglobalred(e) とすれば今度は e が最小であることがわかります。q = [0, 0] で明らかに e 上にある点を作り、ellheight(e, q) としてみましょう。すると零にかなり近い値が得られ、q が捩れ点であろうことがわかります。そして for(k=1,5, print(ellpow(e, q,k))) とすると q が 5 次の点であることがわかります（無限遠での点が [0] として表示されます)。もっと簡単に ellorder(e, q) としてもかまいません。

また他の曲線を見てみましょう。e = ellinit([0,0,1,-7,6]) としてください。これは $y^2+y=x^3-7x+6$ です。ellglobalred(e) でこれは最小の方程式で、導手が判別式と同じで 5077 であることがわかります。この曲線状には自明な整数点がいくつかあるので、体系的に試してみましょう。

ここでまず、捩れ点について調べてみましょう。elltors(e) とすると捩れ点の部分群がすぐにわかります。これで捩れ点について思い悩まなくてすみそうです。つぎにメンバー関数 e.roots によって $\mathbf{C}$、たとえば $Y^2=X^3-7X+25/4$（$(X,Y)=(x,y+1/2)$ とします）上で最小の方程式の 3 個の根が得られます。するともし (x,y) が実数の点であれば、x は少なくとも、たとえば $x\geq -3$ といった最小の根と同じ値になるはずです。そして (x,y) が曲線上にあれば、それと対象な点は $(x,-y-1)$ になるのは明らかです。そこで ellordinate を使って以下のように入力して、結果を見てみます（point.gp というファイルに書いておいて、\r points で読み込んでもかまいません)。

```
{
  v=[];
  for (x = -3, 1000,
    s = ellordinate(e,x);
    if (#s,            \\ @com if cardinality of \kbd{s} is non-zero
      v = concat(v, [[x,s[1]]])
    )
  ); v
}
```

ちょっと話は変わりますが、スクリプト中にコメントを書くにはどうしたらいいかを示しておきます。\\のあとに続く記述はすべて（そこから最初の改行文字まで）無視されます。多くの行にわたるコメントを書きたいときは、/* .. */でくくるとできます。その間にはさまれたものは、同様にすべて無視されます。たとえば points.gp の最初に、ヘッダーとして以下のように書くことができます[25]。

```
/* 楕円曲線 e 上の有理点を、いま私が思いつく限りではもっとも素朴な方法
 * でみつけます（改良が必要)。
```

[25]訳者検証では、EUC で書かれた日本語コメントは OK でした。

```
 * e の係数は有理数でなければなりません。
 * 今後の予定：これをもっと使えるものにする。
 */
```

(こんなものまでキチンと入力したりするのは、時間のムダですよ。)

多くの（18 個）整数点が得られましたね。これらの対称点と無限遠をあわせて 37 個の整数点があり、これは導手がこれほど小さな曲線としては珍しいことです。とすると、この曲線のランクは高いに違いありません。それを調べてみましょう（もし知らなければ、です。この曲線はかなり有名なので)。位数（order）を別の値にしてみましょう（ここでは整数点のみを扱いますが、一般の有理点についても考えてみるといいでしょう)。

`hv = ellheight(e, v)` とすると正規高さ（canonical height）が得られます。では得られている点をこの高さの順に並べてみましょう。

```
iv = vecsort(hv,, 1);     \\ 間接的な並べ替え
hv = vecextract(hv, iv);
v  = vecextract( v, iv);
```

モーデル-ヴァイル群の生成元として高さが最も小さい数をとるのが合理的でしょう。ここでは最初の 4 個で試してみましょう。

```
m = ellheightmatrix(e, vecextract(v,[1,2,3,4])); matdet(m)
```

この曲線には捩れ点がないので、そのデターミナントがほとんど零になることから最初の 4 点は従属だろうと思われます。従属性を見るためには行列 `m` の核（kernel）を見れば十分でしょう。そこで、`matker(m)` としてみてください。非自明な核が得られ、整数であるべき係数は整数に近い値になります。ここで `elladd(e, v[1],v[3])` とすると、これが先ほど得られた `v[4]` と同じであることがわかります。

ほかの 4 点をでも従属性を見ることができます。見てみましょう。`vp = [v[1],v[2],v[3]]~;`、`m = ellheightmatrix(e,vp);`、`matdet(m)` としてみてみだされい。これは零にならず、最初の 3 点は一次独立で曲線のランクは少なくとも 3 であることがわかります（実際には 3 で、`e` はランクが 3 の曲線の中でもっとも導手が小さなものです)。ほかの点が独立かどうかも見てみましょう。これには `ellbil` を使います。ここでもし $\mathbf{Q}$ が `v[1]`、`v[2]` または `v[3]` と従属な点だとしたら、`matsolve(m, ellbil(e, vp,Q))` がその定義から、従属関係の係数を与えます。得られた係数が整数に近い値なら従属性があり、そうでなければ独立であろうと考えられます。`matker` を使うよりもこの方が安心でしょう。以下のように入力すると、

```
w = vector(18,k, matsolve(m, ellbil(e, vp,v[k])))
```

係数が整数に非常に近いことが「見え」ます。これを証明するには

```
wr = round(w); sqrt(norml2(w - wr))
```

として整数との差の最大値の上限を見るといいでしょう。この `wr` は `v` の最初の要素 3 上のすべての要素を表すベクトルです。ここまでくると、この曲線のランクは厳密に 3 で、そう証明できそうだと思ってもよさそうです。

それではもう少し、楕円曲線関係の機能を見てみましょう。ランク 3 の曲線 `e` をそのままにしておいて、以下を入力してください。

```
v1 = [1,0]; v2 = [2,0];
z1 = ellpointtoz(e, v1)
z2 = ellpointtoz(e, v2)
```

この曲線の複素パラメータ表示が得られます。これに点 v1 と v2 を加えるには elladd(e, v1,v2) とします。また ellztopoint(e, z1 + z2) としてもかまいませんが、これには複素数を与えねばならないという欠点があります。しかし $\mathbf{C}$ 上の加法則から e 上の加法群（group law one）がどうやって得られるかを見ることができます。

f = x * Ser(ellan(e, 30)) と入力すると $\Gamma_0(5077)$ の重みが 2 のモジュラー形式のフーリエ展開のべき級数が得られます（これは直接証明できますが、e が準安定なのでワイルズ（Wiles）の結果から追うことができます）。ここで modul = elltaniyama(e)、u=modul[1]; v=modul[2]; とするとこの曲線のモジュラーのパラメータ表示ができます。これに続けて

(v^2 + v) - (u^3 - 7*u + 6)

とします。

x * u' / (2*v + 1) としてみると、これが前述したモジュラー形式 f と同じであることがわかります（クオーテーションマーク ' を入力すると GP は主変数に関する導関数を返します）。関数 u と v は、上半平面上にあり（$x = e^{2i\pi\tau}$）、$\Gamma_0(5077)$ に対するモジュラー**関数**です。

最後に、点 q = [0,0] が 5 次だった例の曲線、e = ellinit([0,-1,1,0,0]) をもう一度見てみましょう。これをもう一度入力してください。この曲線の導手が 11 なのは以前見たとおりです（ellglobalred(e) でまた確認できます）。ここで、関数等式の符号が知りたいとします。それには

elllseries(e, 1,-11,1.1)、続けて elllseries(e, 1,-11,1)

と入力します[26]。

値がはっきりと異なるので、符号は $-$ にはなり得ません。符号を得るような代数的な計算法があり、導手が 6 の約数ではない場合は非常に簡単なのですが、まだその記述が完結していません。

今度は ls = elllseries(e, 1,11,1) として elllseries(e, 1,11,1.1) で確認してみましょう[27]。得られる値は（おおよそ）とるべき値をとり、1 における e の L 関数を与えます。バーチ（Birch）とスウィナートン-ダイナー (Swinnerton-Dyer）の予想（この曲線に関するものです）から、ls は以下の公式を与えます（この場合は、です）。

$$L(E,1) = \frac{\Omega \cdot c \cdot |\mathrm{III}|}{|E_{\mathrm{tors}}|^2}$$

ここで Ω は E の実周期、c は大域タマガワ数で導手を割り切る素数 p に対する局所 c_p の積、|III| はテイト-シャファレヴィッチ（Tate-Shafarevich）群の次数、E_{tors} は E の捩れ群（torsion group）です。

ここではたくさんの量が出てきました。Ω は e.omega[1] で得られます（もし e.roots で一つではなく 3 個の実数根がある場合は Ω は 2 * e.omega[1] で得られる値になるでしょう）。タマガワ数 c は ellglobalred(e) の最後の要素として得られ、ここでは 1 です。E の捩れ部分群は 5 次の点を含むことを見ましたが、torsell(e) とすると[28]その次数が厳密に 5 であることがわかります。そこで ls * 25/e[15] としてみましょう。|III| の値が 1 でなければならないことがわかります。

[26]訳者検証ではエラーになります。"expected character: ')' instead of: elllseries(e,1,-11,1.1)"
[27]これも同じエラーになります。
[28]訳注：これは非互換の昔の機能です。elltors(e) とするべきだ、という警告メッセージが表示されます。

10 二次数体 - Working in Quadratic Number Fields

Q 外では、最も簡単な数体は二次数体です。これはその判別式で定義されますが、または二次数体上でのある特定の次数の判別式が、4 を法として 0 または 1 と合同な平方数ではないような整数 D で定義する方がいいでしょう。この次数が最大であることは `isfundamental(D)` で家訓できます。ω を D が偶数のとき $\sqrt{D}/2$ で奇数のとき $(1+\sqrt{D})/2$ であるとすると、二次数体の元は $a+b\omega$ の形になります。二次数体上の演算を初期化するには、`w = quadgen(`D`)` とします。

これにより `w` は上で与えた ω と等しくなるように設定され、`w` と表示されます。異なる二次数体をいくつか使う場合は、表示される `w` はそれぞれ違う意味になります。たとえば

```
w1 = quadgen(-23); w2 = quadgen(-15);
```

とすると、`w1` と `w2` に関してそれぞれ `w` が表示されますが、それらはもちろん違うものです。複数の二次数体を同時に扱うときには気をつけてください。

ここで、二次数体の元に加えてその次数のイデアルも扱いたいとします。二次の場合はこれは二値の二次形式を扱うことと同じで、PARI ではこれが採用されています。判別式が負の値の場合、二次形式は $ax^2+bxy+cy^2$ を表す 3 つの数の組 (a,b,c) になります。こういった形式は `Qfb(`a`,`b`,`c`)` として表示、生成できます。

このような形式は、PARI のほかのものと同様に乗算、除算、べき乗が普通の演算子を使って行えます。また `qfbred` を使って既約化することができます（これらがどういう意味なのか全部説明することは、このチュートリアルでは避けます)。さらにシャンクス（Shanks）の NUCOMP アルゴリズムが実装されています（`qfbnumcomp` と `qfbnupow`)。こちらの方が少し早く計算できます。

最後に、**普通は**種数を与える `qfbclassno` があります。`qfbclassno` はフルヴィッツ（Hurwitz）の種数を与え、また種数と古典群（class group）構造を与える `quadclassunit` もあります。

ではこれらを全部、例で見てみましょう。

`qfbclassno(-10007)` と入力してください。GP は結果として 77 を表示します。しかし上記の説明で、その結果が必ずしも**正しいとは限らない**としていたことに注意してください。これはこのアルゴリズムを実装した人がなまけ者で、シャンクスのアルゴリズムを完全には PARI に組み込んでないからです。したがってごくたまにですが計算に失敗することがありますが、現実問題としてはほとんどの場合で正しく計算されます。少なくとも基礎判別式については完璧で、`qfbclassno` の結果の確認に使える `quadclassunit` よりは使えるものです（しかし現在は、一般化リーマン仮説 Generalized Riemann Hypothesis（GRH）が前提になっています)。

ということでここでは、種数についてすこし怪しい感じがします。チェックしてみましょう。まず判別式が -10007 の二次数体を求める必要があります。この判別式は 8 を法として 1 と合同なので、ノルムが 2 のイデアルがあることがわかります。たとえば二値の二次形式 (a,b,c) で $a=2$ ということです。これを得るには `f = qfbprimeform(-10007, 2)` とします。式が得られましたね。種数が正しければ少なくとも、この式を 77 乗すれば単位元になるはずです。確かめてみましょう。`f^77` とすると、1、つまり単位元ではじまる式が得られます。つまり確認できたと言うことです。式 `f` の 11 乗と 7 乗は単位元にはなりません。したがって `f` の次数は厳密に 77 であり、したがって種数は 77 の倍数であることがわかります。しかしどうやってそれが 77 であって、その倍数ではないことをはっきりさせられるでしょうか。以下を入力してみてください。

```
sqrt(10007)/Pi * prodeuler(p=2,500, 1./(1 - kronecker(-10007,p)/p))
```

これはディリクレの種数の公式の近似以外のなにものでもありません。`kronecker` はクロネッカー記号で、ここでは単にルジャンドルの記号になります。ここで、最初 1 に小数点をつけて

`1./(1 - \dots)` としたことに注意してください。そうしなければ PARI はすべての有理数について計算を行おうとし、非常に時間がかかって使い物にならなくなる、という恐れもありますよね？この場合は PARI はそんなことはしません（`prodeuler` は常に実数で計算します）が、そんなことを知らないこともあります。安全な方法をとるようにしましょう。

ここで 77.77 という、次数の 77 にかなり近い値を得ました。オイラー積の計算での素数の上限値は、上の証明を厳密にするために変更することができます。

では $D = -3299$ ではどうなるか見てみましょう。`qfbclassno` とオイラー積で、種数が 27 であることが確信できます。しかし上記の簡単な方法では、これを証明しようとするとつまづいてしまいます。そこで `f = qfbprimeform(-3299, 3)` としてみください（ここでは素数イデアルは 2 ではなく 3 です）。そして `f` を 9 乗すると単位元になるのを見てみてください。二次形式をほかのものに変えても同様です。では古典群が輪体ではないことを調べてみましょう。9 個の `f` の累乗を列挙してみると、`f = qfbprimeform(-3299, 3)` はその中に入っていません（3 乗だったらあるでしょう）。これは、古典群は次数 9 の輪体群と次数 3 の輪体群の積であろうことを示唆します。オイラー積での素数の上限を大きくすると証明できるでしょう。

ほかに、`quadclassunit` を使って、たとえば以下のように確認することができます。

```
quadclassunit(-3299)
```

この機能には少しごまかしがあり、GRH を仮定したとしても、間違った解を与えることがあります（古典群の全体ではなく、部分群を得ることがあります）。GRH の下ですでに証明された境界を用いるには、

```
quadclassunit(-3299,,[1,6])
```

とする必要があります。二つ続いているコンマ `,,` は打ち間違いではなく、第二引数を省いていることを示します（これは、ここでの例のような狭い古典群を計算するときに使います）。第三引数を指定したいときには、GP に第二引数が省かれていることを明示する必要があるということです。

今は GRH を真であると見なすと、この古典群がこの例で考えていたものということになります（この機能の詳しい説明は、ユーザーマニュアルの第 3 章を見てください）。

もっと一般的な機能 `bnfinit`（これは一般的な数体を扱い、もっと複雑な結果を返します）を使うと、上の結果を（GPH の仮定をせずに）証明できます。やってみましょう。以下を入力してください。

```
bnf = bnfinit(x^2 + 3299); bnfcertify(bnf)
```

零以外の結果（ここでは 1）は、すべてうまくいっていることを示します。しかし結局なにを証明したことになるのでしょう？この `bnf` を見てみましょう（このまま入力してください）。すごいですね。`init` 機能を思い出してください（`ellinit` については例で見ました）。これは、普通はあまり見たいとは思わない計算上の情報もすべて保存していますが、高機能な計算をするときには使いたい情報もあります。`\bnfcertify` が `quadclassunit` の出力には使えないのは、そのためです。もっとたくさんのデータが必要なのです。

こういった難しいものから、もっと使える情報を取り出すには、**メンバー関数**を使う必要があります（`?.` ですべてのリストが出てくるのを思い出してください）。この場合は `bnf.clgp` が古典群構造を取り出してくれます。先頭の 27 が GP の内部で使う、意味のないパラメータかそうでないかは `%.no` とするとわかります。`bnf.clgp.no` または `bnf.no` でもうまくいくでしょう。

最後の確認として、$\mathbf{Q}\sqrt{-3299}$ のヒルベルト（Hilbert）類体の相対方程式を見てみましょう。`quadhilbert(-3299)` とすると次数 27 になり、これまでのすべてが一致しています。

複素数体ではなく実数の二次数体、たとえば $D > 0$ では、これまでと大きくは変わりません。

二次数体を生成するのも同じ quadgen 機能を使います。イデアルはやはり二値の二次形式 (a,b,c) で表され、この場合は不定符号になります。この数体のアルキメデスの付値が意味を持ち始め（イデアルの代わりにイデールを考える場合はそうなりますね)、正値の判別式を持つ二次形式 4 つの組 (a,b,c,d) で表されます。ここで二次形式そのものは $ax^2+bxy+cy^2$ という形で a、b、c は整数、d はアルキメデスの要素（Archimedean component）で、実数です。こういった記法になれている人には、d はシャンクスとレンストラ（Lenstra）の「距離」を表すといった方がいいかもしれません。

こういった式を生成するには、これと同じ機能をその定義に使いますが、距離を初期化するのに第四引数を与えます。

`Qfb(`a`,`b`,`c`,`d`)`

(a,b,c) の判別式が負の場合、d は自動的に無視されます。これの指定を省いた場合は `0.` を指定したのと同じになります（実数値としての零を現在の精度の桁数で指定したのと同じです)。

これらの二次形式には乗算、除算、べき乗が適用でき、また qfbred を使って既約化できます。この機能は連続的な分数展開に相当する、初等の既約化の連続です。一回の既約化は、この機能を使うときに指定する引数にフラグを与えることで行われます（フラグは省略してもかまいません)。第四要素の d を扱うときには一般的に対数の計算が必要なので、その同じフラグがその第四要素を無視するために使われることがあります。判別式が正の式上でなんの既約化も行わなずに何かの演算をしたいときにはこの機能が便利です（負の場合はこの機能は使い物になりません)。qfbcompraw と qfbpowraw がこれを厳密に行います。

また、qfbprimeform は素形式を与えますが、素数ノルムのイデアルに対応して与えられるこの式は普通は既約化されていません。既約化したいときには qfbred が使えます。

最後に、種数を与える qfbclassno 機能（ここでは正しいことが保証されています)、単数基準を与える quadregulator 機能、それと同様ですが、さらに洗練されていて古典群の構造とその生成元を与える quadclassunit 機能があります。qfbclassno と quadregulator 機能は $O(\sqrt{D})$ のアルゴリズムを使うため、判別式が 10 桁以上の場合には計算に非常に時間がかかります。quadclassunit 機能はもっと広い範囲にまで使えます。

ではこれらの機能の実例と同時に、数論についても少し見てみましょう。まずは `d = 3 * 3299; qfbclassno(d)` と入力してください。種数が 3 であることがわかります（`d = -3299` とショルツ（Sholz）の定理からも、これが 3 で割り切れることがわかります)。そして式を生成しましょう。`f = qfbred(qfbprimeform(d,2), 2)` としてください（最後の 2 は qfbref にアルキメデスの要素を無視するように指示します)。すると素イデアルが 2 に等しいことがわかりますが、これが第一イデアルでしょうか？ これを確かめるには、もっとも効率が高いとは言えませんが、ここでは f に等しい既約化された形式の輪体をすべて見てみればいいでしょう。以下を入力してください。

```
g = f; for(i=1,20, g = qfbred(g, 3); print(g))
```

(ここで、3 は既約化を一回だけ、シャンクスの距離を使わずに行うことを指示します)。この輪体の第一形式（±1 で始まるもの）を持たない f に戻り、f に対応するイデアルが第一イデアルではないことがわかります。

種数が 3 なので、f^3 は第一イデアル $\alpha\mathbf{Z}_K$ になります。では α はどうやったら求められるでしょう？そのためには `f3 = qfbpowraw(f, 3)` とします。すると f の 3 乗が既約化されることなく計算されます。これは $8=2^3$ に等しいノルムのイデアルに対応し、α のノルムが ±8 になることがわかります。さらに、この形式の第四要素からほかにもわかることがあります。ノルムが 1 になるま

で（reach the unit form）式を既約化してみましょう（`qfbred(%,1)` と 6 回入力すればできます)。

アルキメデスが要素は `c = component(%, 4)` で得られます。この距離の定義から、

$$\frac{\alpha}{\sigma(\alpha)} = \pm e^{2c}$$

であることがわかります。ここで σ はここでの二次数体での実数共役です。そして

```
a = sqrt(8 * exp(2*c))
```

として `sa = 8 / a` とすると、符号とともに α と $\sigma(\alpha)$ の近似値が得られます。α は常に正の値をとるように選ぶことができ、また `a` と `sa` の和ではなく差が整数に近いことがわかります。したがって α のノルムは -8 で $\sigma(\alpha)$ に近似値は $-$`sa` になります。また

```
p = x^2 - round(a - sa)*x - 8
```

とすると α の特性多項式になります。この多項式の判別式が `d` の二乗積になり、ここ扱っている体上にあることが確認できます。二乗根の符号をはっきりさせるために α について解き、また近似値を使うと、$\alpha = (15221 + 153\sqrt{d})/2$ を得ます。これを自動的に行うには `polred` と `modreverse` を使います。これらは後で一般の数体の例と 2 変数の 2 連線形方程式の例で見ます。

練習： α が厳密に求められたところで、これが `f3` に対応するイデアルの生成元であることを確認してみましょう。

ここでもう少し輪体について見てみましょう。`D = 10^7 + 1` として以下を入力してください。

```
quadclassunit(D,,[1,6])
```

すると要素数が 5 のベクトルが得られ、(GRH を前提として）種数が 1 で基準（regulator）が 2641.5 であることがわかります。これを証明したいときは、`qfbclass` と `quadregulartor` を使ってください。`bnfinit` や `bnfcertify` を使うと途方もなく長い時間がかかります（精度の桁数の初期化を正しく行って、注意深くやれば 5 分くらいでしょうか)。`bnfcertify` は基礎単位（fundamental unit）を必要としますが、これが大きいため `bnfinit` が（比較的）計算に時間がかかります。これにはおおよそ $R/\log(10) \approx 1147$ 桁の精度が必要です。一方で `quadunit(D)` とするとどうですか？びっくりしますね（これが基準と等しいかどうかは、その対数を計算するとわかります)。

ここでたとえば 500 桁の基準を使いたいとします（最初に基礎単位を厳密に求めておくようなずるはしないものとします)。でも上で求めたままの近似を知っていれば、これは何の苦労もなく行えますね。

次は単位形式（unit form）の例を見てみましょう。`D` が奇数なので距離の初期値は 0 になります。以下の入力してください。

```
u = qfbred(Qfb(1,1,(1 - D)/4), 2)
```

距離の初期値を 0 にするために、`qfbred` には距離を与えずに入力します。

ここで `f = qfbred(u, 1)` としてください。これは第一輪体（principal cycle）に関する最初の形式です。ここでちょっと精度の桁数を少なく、たとえばデフォルトの初期値にしてください。`f` の単位元からのこの距離は約 4.253 です。2641.5 の距離を知ろうとすると、これはおおよそ `f` の距離の約 $2641.5/4.253 = 621$ 倍ににになります。したがってまず、`f^621` としてみます。結果をみると、どうやら行き過ぎのようです。距離が 3173.02 だからです。初期推定を見直して、`f` を $621 * 2641.5/3173$（5174 に近い値です）すると正しい距離が得られると考えてみましょう。ここで `f^517` を計算すると小数点の右側に第一形式（principal form）が得られます。これは別に運が良かったというわけではありません。この方法を使えばいつも必ず目的に近い値を得ます。またそれには簡約修正が一回必要です。輪体中の場所を得るには距離の要素を見るほかにはありません。

ここまでは精度の桁数を抑えて行いました。目的はこの未知の整数 517 を得ることでした。この数字が数学的に重要な意味を持っているわけではないことに注意してください。判別式が正の式の既約化の方法は、既約化には何通りもやり方があるために明確に定義された方法があるわけではありません。しかし PARI では、計算を行うために二次形式の距離ではなく係数で既約化の順序をはっきり決めていて、そのときの精度の桁数の影響はありません。

ここで精度を（たとえば）500 桁に戻してから、またやってみましょう。u の定義をもう一度入力し（なぜ必要になるのでしょうね?）、`f = qfbred(u, 1)`、続けて `f-517` としてみましょう。単位元にたどり着きますね。500 桁の基準が第四要素に何の苦労もなく得られます。

同様に**コンパクトな表現**（compact representaion）と呼ばれる基礎単位、または p 進数の基準を得ることができます。興味があれば試してみてください。

`quadhilbert` 機能もこの体上で試してみることができます。しかし種数が 1 なので、あまり面白い結果が得られるわけではありません。前出の $3 * 3299$ 上でやってみると、計算に 5 分ほどかかるでしょう（そろそろ一休みする時間かもしれませんね）。

11 一般の数体上の演算 - Working on General Number Fields

（訳注：頭がついて行かなくなってますが、ぼちぼちやります。）

12 GP を使ったプログラミング - GP Programming

そのうち書きます。

13 可視化 - Plotting

PARI は様々な出力デバイスについて、高次あるいは低次のグラフ描画機能を多数備えています。`X windows system`[29]、プリンタ用の PostScript ファイル、`gnuplot` 出力デバイス（デフォルトでは最初の二つが使えます）です。これらは様々な機能を備えていますが、どの機能を使うかを示すのに 2 の累乗で表される数値をフラグとして使います。しかし数値のままではわかりにくいので、まずフラグに名前を付けることにします。以下の通りに入力してください。

```
/* 一般的なフラグ：  */
parametric = 1;  no_x_axis =  8;  points       = 64;
recursive  = 2;  no_y_axis = 16;  points_lines = 128;
norescale  = 4;  no_frame  = 32;  splines      = 256;

/* 描画位置の相対的な場所： */
nw       = 0;  se       = 4;  relative = 1;
sw       = 2;  ne       = 6;

/* 文字列の場所：  */
/* 縦 */ bottom  =  0;   /* 横 */  left   = 0;   /* 微調整 */ hgap = 16;
         vcenter =  4;              center = 1;                vgap = 32;
```

[29]訳注：正しくは X window system（window の末尾の s はいりません）。

```
        top     =  8;              right  = 2;
```

そしてデフォルトの精度を、非常に低く設定しましょう。

```
\p 9
```

これは非常に重要で、グラフを描くためには非常に多くの点について関数の計算を行うため、相対精度が 28 桁では破滅的なことになります。出力デバイスの解像度が $10^{28} \times 10^{28}$ ピクセルになるようなことは、まずあり得ないでしょう。

まずは単純な例です。

```
ploth(X = -2, 2, sin(X^7))
```

これを見ると、「直接的な」グラフ描画の限界がわかるでしょう。sin の最初の数周期は -1 と 1 になりますが、左右の境界に近い後の方になってくるとそこまでいきません。PARI は $\sin(X^7)$ を多くの（等間隔の）点で計算しているので、この関数上の、そういった興味を引く点とは直接関係のない点で、関数値が計算されます。最大点や最小点に十分に近い値が計算されていないので、グラフ上の折り返し点での値が間違ってきてしまいます。

これを避けるためには、グラフ上の曲線が大きく曲がる点で、変数の刻み幅を小さくするように指定します。

```
ploth(X = -2, 2, sin(X^7),recursive)
```

グラフの両端の値は、精度が良くなっていますしかし再帰的な描画（PARI が内部でグラフがおおよそ直線になるまで間隔を分割するためにそう言います）には思わぬ落とし穴もあります。この例では、

```
ploth(X = -2, 2, sin(X*7), recursive)
```

グラフは正確に描けているように見えますが、原点やするどく曲がっている点の近くで直線になっています。これはグラフが原点に関して対象になっているためで、中心の 3 点が最初に区間 $[-2, 2]$ を再分割して計算されるときに、完全に直線上にのってしまうからです。PARI はもうこれ以上分割は必要ないと判断し、直線上にこれらの点をプロットします。

これを避ける方法はいくつもありますが、一つには、右側の境界を 2.1 にしてみましょう。または以下のように、分割の点数をデフォルトの 15 点ではなく 16 点にしてもいいでしょう。

```
ploth(X = -2, 2, sin(X*7), recursive, 16)
```

こういった方法は要するに、描画区間の分割の対照性をやぶることで問題を回避します。PARI はこういう病的な場合を自動的に判断できますが、現在は明示的な操作が必要になることもあります。

ploth には、関数の計算に非常に時間がかかるような場合でも描画できるようにする機能があります。$\zeta(\frac{1}{2} + it)$ をプロットしてみましょう。

```
ploth(t = 100, 110, real(zeta(0.5+I*t)), /*empty*/, 1000)
```

これには少し時間がかかるでしょう（多くの出力デバイスで 1000 がデフォルトになっていますが、結果が出力デバイスの種類によらないことを明示するためにわざわざ指定してあります）。再帰描画させてみましょう。

```
ploth(t = 100, 110, real(zeta(0.5+I*t)), recursive)
```

これもかかる時間はほとんど同じでしょう。では描画する点数を少なくして、なめらかな曲線で近似するようにしてみましょう。

```
ploth(t = 100, 110, real(zeta(0.5+I*t)), splines, 100)
```

これはそれほど時間がかかりませんが、出力はほとんど同じです。しかしそれは、どうやってら比べられるでしょうか？ 実は簡単に見られます。ζ 関数の実部と虚部を同じグラフ平面上で見てみましょう。

```
f(t) = z=zeta(0.5+I*t); [real(z),imag(z)]
ploth(t = 100, 110, f(t), , 1000)
```

実数根と虚根の半数が重なっていることがわかります。ここで、なぜ関数 `f(t)` を定義したかわかりますか？ある t の値に対して $\zeta(\frac{1}{2}+it)$ の値を二回計算してしまうのを避けるためです。同様にパラメトリックなプロットもできます。

```
ploth(t = 100, 110, f(t), parametric, 1000)
```

また計算を早くするのに、以下のようにもできます。

```
ploth(t = 100, 110, f(t), parametric+splines, 100)
```

もし出力するデバイスが対応していれば、PARI に線だけでなく点を描かせることもできます。

```
ploth(t = 100, 110, f(t), parametric+splines+points_lines, 100)
```

見てわかるように、グラフ上では点が非常にたくさんあります。もうすこし線を見たいときには、点の数を 30 点などに減らすことができます。しかし点数を 20 までに減らすと、描画するために行っている近似のため、零点を見つけられなくなります。以下のようにすると、スプライン補間を使って t が大きくなってもいいグラフが描けます。

```
ploth(t = 10000, 10004, f(t), parametric+splines+points_lines, 50)
```

二つの異なった方法で描いた同じ関数のグラフを比較するには、どうしたらいいでしょう？ ユーザーマニュアルを見ても、`ploth` にはそういう機能はないようです。しかし、そんなに難しくはありません。

ちょっと考え方を変える必要がありますが、PARI では高次および低次のさまざまな描画機能を組み合わせて使うことができ、それによっていろんなことができます。前出の $sin(X^7)$ をもう一度見てみましょう。グラフの周囲に長方形の枠を描きたいとします。

まず、低次の描画機能で描画するための仮想的な領域に “rectangles”（または “rectwindows”）とよぶ場所をとります（0 から 15 まで番号が振られます）。では中が空の “rectangles” を作って rectangle 2 と名前を付けましょう（0 から 15 の間なら何でもかまいません）。

```
plotinit(2)
plotscale(2, 0,1, 0,1)
```

これで大きさが出力デバイスにぴったり合う長方形の領域ができ、その中に（それぞれ x と y に対して）0 から 1 の座標型ができます。この長方形の領域の周囲に、枠を作りましょう。

```
plotmove(2, 0,0)
plotbox(2, 1,1)
```

この長方形の内側にグラフを描くときに、上部と左側に 0.10 の余白（長方形領域の幅の 10%になります)、下部と上部に 0.02 の余白を作ってみると面白そうです。これは 0.88×0.88 の長方形をもう一つ内側につくることになります（番号 3 を振りましょう)。最初の長方形の 0.88 倍の長方形を作り、グラフを描きます。

```
plotinit(3, 0.88, 0.88, relative)
plotrecth(3, X = -2, 2, sin(X^7), recursive)
```

(まだなにも描かれません。これらの命令はすべて PARI の仮想描画領域に対して行われます。）最後に rectangle 2 と 3 をいっしょに、必要なオフセットをつけて描きます（左上の角から数えます)。

```
plotdraw([2, 0,0,  3, 0.1,0.02], relative)
```

ploth の出力に比べると、なにか欠けているものがありますね。内側の長方形の頂点に座標が書いてありません。もし出力デバイスでマウスが使えるなら（使えるのは gnuplot デバイスだけですが)、グラフ上の任意の点での座標がわかりますが、印刷したハードコピーにも座標があるとよりいいですね。

グラフ上に x と y の境界値を書くのは簡単です。rectangle 2 の座標系では、それらの点は $(0.1, 0.1)$、$(0.1, 0.98)$、 $(0.98, 0.1)$、 $(0.98, 0.98)$ です。x の下界を書くには

```
plotmove(2, 0.1,0.1)
plotstring(2, "-2.000", left+top+vgap)
```

とします。描画されている範囲がわからないので（$\sin X^7$ では簡単に推測できますが)、y の最大、最小値を計算するのには少し工夫が必要です。幸いなことに、plotrecth が x と y の境界値を返してくれます。

以下がプログラム全体になります。

```
plotinit(3, 0.88, 0.88, relative)
lims = plotrecth(3, X = -2, 2, sin(X^7), recursive)
\p 3          \\ $3$桁もあれば十分です。
limits = vector(4,i, Str(lims[i]))
plotinit(2);      plotscale(2, 0,1, 0,1)
plotmove(2, 0,0); plotbox(2, 1,1)
plotmove(2, 0.1,0.1);
plotstring(2, limits[1], left+top+vgap)
plotstring(2, limits[3], bottom+vgap+right+hgap)
plotmove(2, 0.98,0.1); plotstring(2, limits[2], right+top+vgap)
plotmove(2, 0.1,0.98); plotstring(2, limits[4], right+hgap+top)
plotdraw([2, 0,0,  3, 0.1,0.02], relative)
```

これまではどう描くかがわかりやすい例でした。グラフのまわりに枠をつけ、どうなるかなどを見ました。もっとも難しい部分、つまりグラフそのものの描画は可能な限り PARI が行います。ではまた違うことをしてみましょう。$\zeta(1/2+it)$ の厳密なグラフを、スプライン補間と一緒に描きま

す。二つのグラフをそれぞれ別の長方形領域 rectangle 0 と rectangle 1 に描き、一つのプロットにします。または一つの長方形領域 rectangle 0 に二つのグラフを描くこともできます。

ここで問題となるのは、前出の例で rectangle 3 には座標系がないまま rectangle 2 の中にどうやって座標系を導入するかです。rectangle 3 にグラフを描くときには自動的に座標系が設定されます（出力デバイスでマウスが使えれば、座標系を見ることができます)。二つのグラフで描き方を変える場合、グラフの描画範囲は正確に同じにはならないので、長方形領域の出力の仕方に少し工夫が必要になります。二つ目のグラフを描くときに plotrecth として、先に描いたグラフの座標系を使うように指示します。たとえば、同じグラフを点の数を減らして描くときには以下のようにします。

```
plotinit(0, 0.9,0.9, relative)
plotrecth(0, t=100,110, f(t), parametric, 300)
plotrecth(0, t=100,110, f(t), parametric+splines+points_lines+norescale, 30);
plotrecth(0, t=100,110, f(t), parametric+splines+points_lines+norescale, 20);
plotdraw([0, 0.05,0.05], relative)
```

これで見たいものが見られます。グラフの描き方をいろいろ比較することはできますが、だんだんと混乱してきてどのグラフが何なんのか、わからなくなってきますね。少なくとも一つの出力デバイスについては、グラフの見せ方をいろいろと変えることができます。これで複数のグラフを見分けることができます。いくつも四角の箱が出てくるのをなんとかするのは、そう難しくありません。その四角の箱は、各グラフの外枠です。これは plotrect のオプションで描画を抑制することができます。そういった縁取りをつけることにすると上のスクリプトは以下のようになります。

```
plotinit(0, 0.9,0.9, relative)
plotpointtype(-1, 0)                \\ グラフの点の色を指定
plotpointsize(0, 0.4)               \\ 点を小さく描く（もし可能なとき）
plotrecth(0, t=100,110, f(t), parametric+points, 300)
plotpointsize(0, 1)                 \\ 点の大きさを普通にする
plotlinetype(-1, 1)                 \\ グラフの線の色を指定
plotpointtype(-1, 1)                \\ グラフの点の色を指定
curve_only = norescale + no_frame + no_x_axis + no_y_axis
plotrecth(0, t=100,110,f(t), parametric+splines+points_lines+curve_only, 30);
plotlinetype(-1, 2)                 \\ グラフの線の色を指定
plotpointtype(-1, 2)                \\ グラフの点の色を指定
plotrecth(0, t=100,110,f(t), parametric+splines+points_lines+curve_only, 20);
plotdraw([0, 0.05,0.05], relative)
```

二つ目と三つ目のグラフでも座標軸は欠けずに描かれますが、特に必要ないので無視しましょう。厳密に描いたグラフと 30 点のグラフを比べると違いがありますが、あまり大きな違いではありません。一方、20 点にまで減らしたものとでは無視できない違いがあります。

plotoinetype、plotpointtype、plotpointsize は、出力デバイスによっては機能しないものがあります。

高解像度のハードコピーがほしいときにはどうしたらいいでしょう？ いつくか方法がありますが、ディスプレイ出力デバイスを使っても高解像度ハードコピーができます。PM ディスプレイ（OS/2 の gnuplot 出力を使います）は 19500×12500 の解像度を要求しますが、すると PARI が

このデバイスにデータを送る時点ですでに高解像度になっているので、メニューバーから印刷することができます。または gnuplot 出力を使って出力デバイスをいろいろなハードコピーデバイス、`plotfile("plot.tex")` や `plotterm("texdraw")` に切り替えることができます。こうするとプロットはすべて `plot.tex` というファイルに gnuplot の `texdraw` 出力によるフォーマットで書き出されます。出力先を元に戻すためには、出力デバイスの端末を元に戻す必要があります。ファイルを初期設定に戻す必要があり、それは `plotfile("-")` でできます。

PARI のプログラミング機能を使って同時に複数のプロットをすることができます。

```
plotfile("manypl1.gif")        \\標準出力がバイナリモードになるのを避ける
plotterm("gif=300,200")
wpoints = plothsizes()[1]      \\$300 \times 200$は単なる一例
{
  for( k=1,6,
    plotfile("manypl" k ".gif");
    ploth(x = -1, 3, sin(x^k), , wpoints)
  )
}
```

これで $\sin x^k$, $k = 1 \ldots 6$ を 6 つの 300×200 の GIF ファイル `manypl1.gif`...`manypl6.gif` に出力できます[30]

さらに、PARI から PS ファイルに出力するようにもできます。これまでの例での `plotdraw` の代わりに `psdraw` を使います（または `ploth` の代わりに `psploth` を使う方法もあります)。この操作で出力するファイルを変える方法は、`default` コマンドの `psfile` オプションを見てください。PARI の PS 出力の精度は、そのときの出力デバイスと同じになることに注意してください。

たくさんの小さなグラフを一つの図にまとめたいとしましょう。上の $\zeta(1/2 + it)$ の例でやったような、一つの長方形領域にすべてのグラフを入れる方法は、各グラフがそれぞれ異なる位置に配置されるため、使えません。長方形領域は 16 個までしか使えないため、PARI ではあまり便利に使えるわけではありません。つまりこれは、一つの図に描けるグラフの数は 16 までということでしょうか？ありがたいことに、PARI の描画エンジンにはそんな制約はありません。`plotcopy` があります。

以下のスクリプトは 4 つの異なるグラフを、`A` と `T` の 2 つの長方形領域を使って一つの図にします。

```
A = 2;   \\ accumulator
T = 3;   \\ temporary target
plotinit(A);          plotscale(A, 0, 1, 0, 1)

plotinit(T, 0.42, 0.42, relative);
plotrecth(T, x= -5, 5, sin(x), recursive)
plotcopy(T, 2, 0.05, 0.05, relative + nw)

plotmove(A, 0.05 + 0.42/2, 1 - 0.05/2)
plotstring(A,"Graph", center + vcenter)
```

[30]訳注：`gnuplot` 出力デバイスを組み込んでいなければ、ファイルには出力されません。

```
plotinit(T, 0.42, 0.42, relative);
plotrecth(T, x= -5, 5, [sin(x),cos(2*x)], 0)
plotcopy(T, 2, 0.05, 0.05, relative + ne)

plotmove(A, 1 - 0.05 - 0.42/2, 1 - 0.05/2)
plotstring(A,"Multigraph", center + vcenter)

plotinit(T, 0.42, 0.42, relative);
plotrecth(T, x= -5, 5, [sin(3*x), cos(2*x)], parametric)
plotcopy(T, 2, 0.05, 0.05, relative + sw)

plotmove(A, 0.05 + 0.42/2, 0.5)
plotstring(A,"Parametric", center + vcenter)

plotinit(T, 0.42, 0.42, relative);
plotrecth(T, x= -5, 5, [sin(x), cos(x), sin(3*x),cos(2*x)], parametric)
plotcopy(T, 2, 0.05, 0.05, relative + se)

plotmove(A, 1 - 0.05 - 0.42/2, 0.5)
plotstring(A,"Multiparametric", center + vcenter)

plotlinetype(A, 3)
plotmove(A, 0, 0)
plotbox(A, 1, 1)

plotdraw([A, 0, 0])
\\ psdraw([A, 0, 0], relative)          \\ ハードコピーが必要なときはこちら
```

Rectangle `A` は累算器（accumulator)、rectangle `T` は `plotrecth` のターゲットとしてのみ使われます。Rectangle `T` にプロットが行われたらすぐ、そのプロットは累算器にコピーされます。この例にはいくつかの数字が出てきます。ここでは（プロットの大きさに比べて）間隔 0.06、外側の余白 0.05 で 4 つの長方形領域を描きたいとします。すると一つの長方形領域の大きさは 0.42 になります。そして各プロットには、余白の中央にキャプションをつけるとします。キャプションのための長方形領域を用意する必要はありません。これも累算器の中に入ります。

各長方形領域を単に置いていくのに、上の例では “geographic” 記法を使いました。`plotcopy` の最後の引数はグラフの角の点（北西、とも言います）を指定し、そこからのオフセットを指定できます（オフセットが正の値のときは、長方形領域の中になります)。

さらに別の便利な機能の例を見てみましょう。0 での $\sin x$ のテイラー級数のプロットするプログラムを作ってみます。簡単にするために、プログラムはどんな関数にもつけるようにしますが、関数は奇関数で、0 に関するテイラー級数の奇数項だけをプロットします。まずショートカットをいくつか定義しましょう。

```
xlim = 13;  ordlim = 25;  f(x) = sin(x);
```

```
default(seriesprecision,ordlim)
farray(t) = vector((ordlim+1)/2, k, truncate( f(1.*t + O(t^(2*k+1)) )))
FARRAY = farray('t);  \\ 't は変数 t を解放します
```

farray(x) は $f(x)$ のテイラー級数をベクトルとして返し、ここではこれを FARRAY に格納します。$f(x)$ を長方形領域に描くには、高さを 1.2 倍にしてその中に描きます。大きな長方形領域は最終的に 0.9 になるとします。

まず最初に、小さな長方形領域の大きさを測ります。

```
plotinit(3, 0.9, 0.9/1.2, 1);
curve_only = no_x_axis+no_y_axis+no_frame;
lims = plotrecth(3, x= -xlim, xlim, f(x), recursive+curve_only,16);
h = lims[4] - lims[3];
```

次は大きな長方形領域を作り、その中にテイラー級数をプロットします[31]。

```
plotinit(4, 0.9,0.9, relative);
plotscale(4, lims[1], lims[2], lims[3] - h/10, lims[4] + h/10)
plotrecth(4, x = -xlim, xlim, subst(FARRAY,t,x), norescale);
```

そしてこれが、この例での最も重要な命令です。

```
plotclip(4)
```

これはいったい何をするのでしょう？plotrecth(...norescale) コマンドは描こうとする長方形領域にあわせてグラフの大きさを調整しますが、ほかには何もしません。xlim が 13 なのでテイラー級数は区間-xlim...xlim で非常に大きな値になります。この場合、グラフの重要な部分が長方形領域の外にはみだしてしまいます。plotclip は、そのはみ出した部分を切り取ります。

```
plotinit(2)
plotscale(2, 0.0, 1.0, 0.0, 1.0)
plotmove(2,0.5,0.975)
plotstring(2,"Multiple Taylor Approximations",center+vcenter)
plotdraw([2, 0, 0,  3, 0.05, 0.05 + 0.9/12,  4, 0.05, 0.05], relative)
```

これによりキャプションが入り、3 つの長方形領域（キャプションと関数のグラフとテイラー級数のプロット）がつなげられます。

これで PARI の描画機能の簡単な例は終わりですが、いくつかの注意点がまだあります。まず、普通の出力デバイスでは、画像は小さな色の付いた四角い点（ピクセル）から成るということです (非常に大きなチェス盤のようなものです)。出力のための長方形領域はそういった小さな四角が集まったものです。そこに描くということは、それぞれの小さな四角を一つずつ塗りつぶしていくことになります。長方形領域は座標系を持っているので、これらの四角の境界がその座標系でどのようになっているのかを知ることが重要なこともあります。

plotscale コマンドは、長方形領域での x、y の範囲を表示します。その座標系での x と y の上限、下限は、四隅にある小さな四角の**中央**です。もし x と y の範囲が $[0,1]$ だったら、(0,0) と (0,1) をつなく線分は左の列の中央にきます。また、この線分の端点で計算に小さな誤差があった場合でも、線分の交点がどの四角の中にあるかは変わらないでしょう。したがって描かれる図は同

[31] 訳者検証では、ここの 3 行目 plotrecth が 40 分かかっても終わらず、これに続く例は試せませんでした。

じになってしまいます（PARI の計算には近似値を使うものも多いので、こういった細かい点も重要なのです)。

また他に、ここで使っている例では、長方形領域に相対的な位置、大きさを使っている点も重要です。こうすると、出力デバイスが変わってもほぼ同じような出力を得ることができ、出力デバイスの解像度などを気にする必要がなくなります。一方でまた、相対位置指定では図がいつも必ず同じになるとは限りません。これはなぜでしょう？ もし二つの異なる出力デバイスが同じ解像度を持っていても、図は変わってきます。各デバイスは異なるサイズのフォントを使うことがあり、また「単位長さ」も異なるからです。

PARI の出力デバイスについての情報は、6 個の数字で表されます。解像度、フォントの文字マスの大きさ、単位長さそれぞれについて、水平、垂直の値があります。これらの数値はピクセル数で表されます。`plotsizes` 機能でこれらの数値を見ることができます。「単位長さ」はグラフの長方形領域内で `ploth` の上部と右側の余白、`plotstring` の余白を計算するのに使われます。左側と下部の余白は単位長さと文字マスの大きさから計算されます（したがってグラフを印刷するときには十分な余裕があります)。

これで何が見えますか？ 相対的に大きさを指定することで、デバイスが異なっても**おおよそ**同じプロットが得られますが、「同じように見える」保証はありません。さらに「同じように見える」ことが望ましいのではなく、「出力デバイスに最適な見え方」が望まれます。そういった最適に調整されたプロットがほしい場合、相対的な指定をあきらめ、ピクセル単位でプロットする必要があります。そのためには大きさやオフセットを相対的に解釈するための `relative` フラグを上述の例から消します。`plothsizes` で小さな長方形領域の大きさと位置を、出力デバイスごとに自由に、詳細に決めることができます。

最適に調整できたかどうかを確認するのに、解像度があまり高くないプロットして（多くの出力デバイスがそうですが)、解像度が低いプロット（gnuplot の `plotterm("dumb")` など）を使ってみてください。

14 翻訳のための参考情報

以下を参考にしました。とくに ponpoko さんの web にある、このチュートリアルの先訳 [1] は、私に足りない数学知識を補うのに非常に有用でした。深く感謝します。

14.1 計算機環境

この文書の作成と本文中の例の実証は、以下の環境で行いました。

- Mac OS X v.10.3.2
- Apple / PowerBook G4(867MHz, 640MB メモリ)
- GP/PARI CALCULATOR ver. 2.2.8 (development CHANGES-1.907)
  (readline v4.3 enabled, extended help available)
  gnuplot は組み込んでません。
- pTeX Version 3.14159-p3.1.2 (sjis) (Web2C 7.4.5)
- pLaTeX2e ¡2001/09/04¿+0 (based on LaTeX2e ¡2001/06/01¿ patch level 0)

- dvipdfmx-20021230

以上 TEX 関係は pTeX package for Mac OS X[32] のものです。

14.2 最新版

14.2.1 PARI/GP の最新版

PARI/GP の最新版を CVS リポジトリからもらってくるには、Unix や Mac OS X でターミナル（またはターミナルエミュレータ、xterm、kterm、rxvt など）を起動し、以下のようにします (%はシェルのプロンプトです)。

```
% cvs -d :pserver:cvs@pari.math.u-bordeaux.fr:/home/cvs login
% cvs -z3 -d :pserver:cvs@pari.math.u-bordeaux.fr:/home/cvs checkout pari
```

一行目を入力するとパスワードの入力を求められますが、リターンキーだけを入力します。二行目を入力すると、カレントディレクトリに pari というディレクトリが作られ、その中にソースが展開されます。

14.2.2 GNUPLOT の最新版と組み込み

同様に、グラフィクス用に gnuplot を CVS からもらってきて、PARI/GP に組み込む準備をするには以下のようにするのが、安直ですが楽です。1 行目でパスワードを聞かれたらやはりリターンだけでかまいません。configure には環境と必要に応じて、適当な引数を与えます（なにも与えなくてもいいかもしれません)。make install の必要はありません（%はシェルのプロンプトです)。

```
% cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/gnuplot login
% cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/gnuplot co gnuplot
% cd gnuplot
% ./configure
% make
% cd src
% ar cr libgnuplot.a version.o util.o term.o bitmap.o stdfn.o
% sudo cp libgnuplot.a /usr/local/lib
```

この後に PARI/GP のソースツリーのディレクトリに移動して、Configure & make します。Configure が libgnuplot.a を見つけて、その旨を表示するはずです。しかし CVS からもらってきた PARI/GP の開発中バージョンでは、gnuplot の ver.3.7.3、3.8j ともに組み込みに失敗しました（どちらも最後にリンクするときです)。ので、それ以上は試していません。

14.2.3 この文書の最新版

この翻訳文の最新版は、http://www.cbrc.jp/~tominaga/translations/に置いています。

[32] http://www.r.dendai.ac.jp/kiriki/tex/

参考文献

[1] http://www.bekkoame.ne.jp/~ponpoko/Math/pari/pari.html

[2] フェルマーの大定理が解けた！(足立著、ブルーバックス)