単位： 実習課題の達成度によって評価します。

数式処理とは： 数学の研究や勉強において、ソフトウェア（プログラミング言語）を利用する目的は、おもに次の 3 点です。

- 数値計算
  - 多くの自然現象は微分方程式で記述することができます。ところが、ほとんどの関数が明示的に積分できないのと同じように、ほとんどの微分方程式は厳密に解くことができません。したがって厳密解を求めるのは諦めて、コンピュータに頼って近似的に解くことになります。このような研究分野を数値解析といいます。数値解析には C 言語を使う人が多いようです。また、行列の数値計算に強いソフトウェアとして Matlab が有名です。
- 記号計算（数式処理）
  - 数式処理とは、私たちがふだん紙の上で行っている式の計算、たとえば多項式の加減乗除や因数分解、あるいは関数の微分積分などをコンピュータにさせようとするものです。数式処理は数値計算とちがって計算に誤差が生じません。したがって計算の精度という問題から開放され、厳密な議論ができるという強みがあります。数式処理を行う代表的なソフトウェアとして Mathematica, Maple, Maxima, GAP, Sage, SINGULAR, REDUCE, Risa/Asir などがあります。それぞれのソフトウェアに得意分野があります。
- グラフィクス
  - 数学を理解する上で、可視化（図を描くこと）は大切な作業です。

また本実習では、数式処理とは無関係ながら、TEX（テフ）の使い方についても学びます。今後のスケジュールは Mathematica を 12 回、TEX を 3 回の予定です。

実習の準備： まず本実習用のフォルダ（ディレクトリ）を作っておき、プログラムはすべてそこに保存するようにします。フォルダを作る場所とフォルダの名前はたとえば `H:¥mathematica` とすればよいでしょう。作成手順は、マウスで「スタート」→「コンピューター」→「ホームフォルダ (H:)」と辿り、右クリックで「新規作成」→「フォルダー」を選び、最後にキーボードから「`mathematica`」と入力します。

起動： 「スタート」→「すべてのプログラム」→「Wolfram Mathematica」→「Wolfram Mathematica 9」

実行： Mathematica を起動するとウィンドウが開きます。ここにプログラムを書きます。書いたプログラムコードを実行するには、そのコード中の任意の場所で「`Shift+Enter`」(`Shift` キーを押しながら `Enter` キーを押す）します。

- 実際のプログラムコードを書き始めるまえに、まず先にファイル名を決めて、先ほどの場所（`H:¥R`）に保存する習慣をつけましょう。また、プログラムを書いたらこまめに保存することが大切です。これにより、不慮の事故（Mathematica が固まってファイルが保存できなくなっちゃった）からある程度身を守ることができます。ファイルの保存は、Mathematica のメニューから「ファイル」→「保存」を選びます。保存するファイルの種類は「Mathematica ノートブック (`*.nb`)」にします。

終了： Mathematica のメニューから「ファイル」→「終了」

基本的な使い方（その 1）： まずは簡単な四則演算から始めます。電卓的な使い方から始めて、代数方程式を解く方法までを説明します。

- 加減乗除にはそれぞれ `+`, `-`, `*`, `/` を用います。ただし、たとえば `1/3` を小数として出力したいときは `N[1/3]` と入力します。このコマンド `N[]` は、数値 (numeric) の意味です。
  - 表示する桁数を `k` 桁にしたいときは `N[1/3,k]` とします。詳しくは `?N` してみましょう。
- 累乗は `^` で表します。たとえば 2 の 10 乗は `2^10` です。
- 初等関数として `Sqrt[]`, `Exp[]`, `Log[]`, `Cos[]`, `Sin[]`, `Tan[]`, `ArcTan[]` などが用意されています。
- 定数として、円周率 `Pi` や虚数単位 `I` や自然対数の底 `E` など。
- 変数への代入は `=` を用います。たとえば `a=1` と入力すれば変数 `a` に `1` が代入されます。変数の型を宣言しておく必要はありません。
- 関数を定義するには、たとえば $f(x) = x^2$ ならば `f[x_]=x^2` または `f[x_]:=x^2` とします。前者の `=` は即時評価を意味し、後者の `:=` は遅延評価を意味しますが、両者のちがいについては後日説明します。
- 方程式 $x^2 = 1$ を解くには `Solve[x^2==1,x]` とします。等号（両辺が等しい）には記号 `==` を使います。代入や関数の定義につかう記号 `=` とは意味が全く違いますので、はっきり区別しましょう。
  - 方程式を解くには `Reduce[]` を使うこともできます。とくに方程式がパラメータを含む場合は、正確に解くためには `Solve[]` でなく `Reduce[]` を使う必要があります。たとえば `Solve[a*x+b==0,x]` と `Reduce[a*x+b==0,x]` の出力を比べてみましょう。
  - 積には明示的に`*`を付けるようにしましょう。たとえば `a` と `b` の積は `a*b` です。これを省略して `a b` と書くこともできますが、これは、しばしば `ab` と入力してしまいがち（`a` と `b` の間のスペースを入力し忘れた）なのでおすすめしません。このように `ab` と書くと、これは `ab` という名前のひとつの変数であると解釈されます。したがってエラーが起こらず、プログラムが長くなった場合にバグに気付きにくくなります。

基本的な使い方（その 2）： 上で説明した基本事項を応用して、簡単な数学の問題を解いてみます。

- 典型的には次のような使い方をします。例として、放物線 $f(x) = x^2$ と直線 $g(x) = x + 1$ で囲まれる図形の面積を求めてみます。この面積は、2 次方程式 $f(x) = g(x)$ の解を a, b $(a < b)$ としたとき、定積分

$$\int_a^b (g(x) - f(x))\, dx$$

で計算できます。この計算を Mathematica にさせてみましょう。

```
f[x_]=x^2
g[x_]=x+1
Plot[{f[x],g[x]},{x,-2,2}]    (*f と g のグラフを描く*)
sol=Solve[f[x]==g[x],x]       (*交点の x 座標を求める*)
a=x/.sol[[1]]                 (*小さいほうを a とする*)
b=x/.sol[[2]]                 (*大きいほうを b とする*)
Integrate[g[x]-f[x],{x,a,b}]
```

  - リスト `sol` の成分は `sol[[1]]` や `sol[[2]]` などと取り出します。
  - ひとつ大切な使い方を覚えましょう。数値の代入の仕方です。これは「式 `/.{代入のルール}`」という形で使います。例えば、式 $2x + 1$ に $x = 3$ を代入するには

```
2*x+1/.{x->3}
```

と書きます。このとき出力はもちろん `7` です。上記プログラム 5 行目と 6 行目の右辺は、この代入の操作を意味しています。

数式処理実習課題（1 回目）　　学籍番号の下 2 桁 □□　名前＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

1 π^e と e^π はどちらが大きいか。

2 $a=\sqrt{3}+\sqrt{-2},\ b=\sqrt{3}-\sqrt{-2}$ のとき $\dfrac{a}{b}+\dfrac{b}{a}$ と a^4-2a^2 の値を求めよ。ヒント：`?Simplify`

3 $f(x)=x^3-2x+1$ とする。方程式 $f(x)=0$ を解け。また $y=f(x)$ のグラフを描け。ただし x 軸との交点がわかるように適当な描画範囲を指定すること。さらに曲線 $y=f(x)$ と x 軸で囲まれる部分の面積を求めよ。

4 （難）$a=\sqrt[3]{7+5\sqrt{2}},\ b=\sqrt[3]{7-5\sqrt{2}}$ のとき $a+b$ と ab の値を求めよ。
ヒント①：根号の入力は Mathematica のメニューから「パレット」→「基本数学アシスタント」。あるいは $\sqrt[3]{x}=x^{1/3}$ で。
ヒント②：定数 `b` が定義できたら `N[b]` してみましょう。
ヒント③（数値 `N[b]` を見て「おかしいな」と思った人のためのヒント）：定数 b は方程式 $b^3=7-5\sqrt{2}$ の解のひとつです。

こんな質問がありました：　前回の実習時にでた質問のうちいくつかを紹介します。

- ゼロになるはずのない定積分がゼロになりました。
  - 被積分関数の定義を見直してください。場合によっては `Clear[f]` を実行していったん被積分関数 `f` の定義をクリアしてから、もういちど `f` の定義式を実行し直したほうがよいかもしれません。
  - ここで重要な注意をします。例えば

```
f[x]=x
f[x_]=x*x
```

    という 2 つの定義が同時に存在する場合、`f[4]` の値は `16` となり `f[y]` の値は `y*y` となりますが、`f[x]` の値は `x*x` でなく `x` となります。このように Mathematica では `f[x]` のような、より特殊な定義がある場合、そちらが優先的に適用されます。また Mathematica はいちど評価した式はずっと記憶しているので、たとえいったん `f[x]=x` の行をプログラムから消去し、再度 `f[x_]=x*x` を評価し直したとしても、`f[x]` の値は消えずに `x` のまま残ります。Mathematica のこの仕様はしばしば混乱の原因となります。
  - そこで、すべてのプログラムには、その 1 行目に

```
ClearAll["Global`*"]
```

    と書く習慣をつけましょう。これを書いておけば、ノートブック全体を再評価することによって、すべての定義がすっかり消去され、Mathematica を起動したばかりのような状態で動作させることができます。ノートブック全体を再評価するには、Mathematica のメニューから「評価」→「ノートブックを評価」を選択します。
  - 命令 `ClearAll["Global`*"]` の意味を理解するには、まずコンテキストについて知っておく必要があります。コンテキストとは、Mathematica が変数や関数の名前などのシンボルを記憶しておく領域（のようなもの）です。ノートブックで普通に関数や変数を使うとその名前（シンボル）は `Global` コンテキストに登録されます。Mathematica が扱うシンボルの正式名は、コンテキストと簡略名を記号 ` で結合したものです。例えばノートブックで普通に `a=1` とすると `Global`a` というシンボルが定義され、これに `1` という値が結び付けられます。通常はコンテキスト部分は省略され、簡略名だけで表現されます。次に、関数 `ClearAll[]` の使い方ですが、もし `ClearAll[symb]` と書けば記号 `symb` の値や定義をすべてクリアします。またもし `ClearAll["expr"]` と書けばパターン `expr` にマッチする記号名をすべてクリアします。記号 `*` は任意の文字列にマッチするワイルドカードです。
- 方程式 $x^2-x-1=0$ のふたつの解 a,b に対して a^3+b^3 の値を計算する場合

```
sol=Solve[x^2-x-1==0,x]
a=x/.sol[[1]]
b=x/.sol[[2]]
a^3+b^3
```

  と入力すればよいことは何とか理解できました。しかし、この代入の仕方がわかりづらいので、次のようにしてはいけませんか。

```
Solve[x^2-x-1==0,x]
(* 答えが {{x->(1-Sqrt[5])/2},{x->(1+Sqrt[5])/2}} だと表示されるので、キーボードから直接次のように入力する *)
a=(1-Sqrt[5])/2
b=(1+Sqrt[5])/2
a^3+b^3
```

  - 確かにそう書いたほうが直接的で理解しやすいのはわかります。初心者のうちは後者の書き方でオーケーです。しかし、ある程度慣れてきたらなるべく後者の書き方は卒業しましょう。というのも、プログラムは「1 回書いたら終わり」ではなくて、ふつうは試行錯誤を繰り返しながら何度も書き換えていくものだからです。そのとき、出発点の方程式を変更することもよくあります。たとえば方程式が $x^2+x+1=0$ に変わったら当然ながら a と b の値も変わります。このとき、もし後者の書き方をしていると、いちいち手作業で `a` と `b` の値を更新しなくてはいけません。ところが、前者の書き方をしていれば書き換えるのは方程式だけで済み、`a` と `b` の値は自動的に更新されます。つまり、前者のほうがプログラムを修正する作業がずっと楽になるため、後者にくらべて上手な書き方なのです。
- たとえば「$a=2$ のとき $a+1$ の値を求めよ」という場合

```
a=2
a+1
```

  と書くのと

```
a+1/.{a->2}
```

  と書くのはどういう違いがあるのですか。
  - それは変数への「大域的な代入」と「局所的な代入」との違いです。前者の書き方では `a` の値は常に `2` として固定されており、式 `a+1` の計算が終わった後も `a` の値は `2` です。後者の書き方では式 `a+1` の中にあらわれる `a` に値 `2` を代入しているだけなので、いったん式 `a+1` の評価が終わってしまえば `a` の値は不定の状態にもどります。つまり入力「`a=2; a+1; a`」に対する出力は `2` となりますが、入力「`a+1/.{a->2}; a`」に対する出力は `a` となります。状況に応じて便利なほうの書き方を選びます。
- 実習課題の 4 番は、どこが難しいのか、いまいちわかりません。
  - 冪根（n 乗根）の計算には十分注意しましょう、という趣旨の問題です。実習課題では $a=\sqrt[3]{7+5\sqrt{2}}$ と $b=\sqrt[3]{7-5\sqrt{2}}$ がどちらも実数なのに、和 $a+b$ や積 ab の値が複素数になってしまった人が多くいました。その理由をいちばん簡単な例で説明します。
  - たとえば $c=\sqrt[3]{-1}$ としましょう。右辺の記号 $\sqrt[3]{-1}$ は、ふつう数学の文脈では「方程式 $x^3=-1$ の実数解」を意味します。すなわち $\sqrt[3]{-1}=-1$ です。ところが Mathematica は、記号 $\sqrt[3]{-1}$ あるいは記号 $(-1)^{1/3}$ を、単に「3 乗したら -1 になる数」と理解します。そうすると原理的には 3 通りの可能性があります。すなわち

$$(-1)^{1/3}=\cos\frac{\pi}{3}+\sqrt{-1}\sin\frac{\pi}{3}$$
$$(-1)^{1/3}=\cos\pi+\sqrt{-1}\sin\pi$$
$$(-1)^{1/3}=\cos\frac{5\pi}{3}+\sqrt{-1}\sin\frac{5\pi}{3}$$

    のうちのどれかです。この 3 つのうちどれを採用するか決めないといけませんが、Mathematica においては、自然数 n に対して

$$\sqrt[n]{-1}=(-1)^{1/n}=\cos\frac{\pi}{n}+\sqrt{-1}\sin\frac{\pi}{n}$$

    と定められています。これは実際 `ComplexExpand[(-1)^(1/n)]` を実行してみるとわかります。したがって Mathematica では `N[(-1)^(1/3)]` の出力は `0.5+0.866025I` という複素数です。この数値は $\cos(\pi/3)+\sqrt{-1}\sin(\pi/3)$ の近似値です。
  - このように、冪根を含む計算では、きちんと自分が意図した値となっているかどうか、注意深くチェックする必要があります。なお、この面倒を避けるため、3 乗根 $\sqrt[3]{x}$ に対しては実数の値を返す `CubeRoot[x]` という命令が用意されています。より高次の冪根 $\sqrt[n]{x}$ については実数を返す `Surd[x,n]` という命令があります。

今日の内容に入りましょう。行列とベクトルについて実習します。

行列の作成：　行列を作る方法は、目的や場面に応じて、複数の手段が提供されています。

- 成分をすべて手入力したい場合。たとえば行列

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

  を作るには `A={{1,2,3},{4,5,6}}` とします。
- – 行列を入力するためのユーザインタフェースも用意されています。メニューから「挿入」→「表・行列」→「新規作成」を選びます。ノートブックに行列が挿入されるので、プレースホルダをクリックして入力します。次のプレースホルダへは、タブで移動します。
- 成分がパラメータとして与えられている場合。たとえば行列

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

  を作るには `A=Table[a[i,j],{i,1,2},{j,1,3}]` とします。
- – とくに成分がより具体的な関数で指定されている場合、たとえば

$$a_{ij} = \frac{1}{i+j-1}$$

  の場合は、はじめから `A=Table[1/(i+j-1),{i,1,2},{j,1,3}]` と書けます。

特別な行列の作成：　対角行列や三角行列のような特別な n 次正方行列については、簡単な作成方法が用意されています。

- 対角行列 $\mathrm{diag}\,(a,b,c)$ は `DiagonalMatrix[{a,b,c}]` とします。また特に単位行列 $\mathrm{diag}\,(1,1,1)$ は `IdentityMatrix[3]` としても作ることができます。
- 三角行列を作る（下三角部分や上三角部分を 0 で置換する）ための命令として `UpperTriangularize[]` や `LowerTriangularize[]` があります。詳しくは `?UpperTriangularize` で。

行列の部分を取り出す方法：

- 行をとりだす。たとえば行列 `A` の第 1 行目を取り出すには `A[[1]]` とします。
- 成分をとりだす。たとえば行列 `A` の第 1 行第 2 列目の成分を取り出すには `A[[1,2]]` とします。
- 列をとりだす。まず `All` ですべての行を選び、列を指定します。たとえば行列 `A` の第 1 列目を取り出すには `A[[All,1]]` とします。
- 小行列をつくる。たとえば行列 `A` の第 i 行と第 j 列を削除するには `Drop[A,{i},{j}]` とします。また行や列だけを削除する場合は `Drop[A,{i},None]` や `Drop[A,None,{j}]` とします。

ベクトルの作成：　ベクトルを作るにはすこし注意が必要です。

- `{a,b,c}` はベクトルです。
- – これが基本的な作り方です。通常はこの方法で作ります。
- – ただし、このようにして作ったベクトルは、縦ベクトルと横ベクトルの区別がありません。これはソフトウェアの挙動を柔軟にするための措置ですが、数学を専攻する私たちにとっては、縦ベクトルと横ベクトルの区別がないのはむしろ分かりづらく、混乱の原因となる場合があります。そのような場合は、縦ベクトルと横ベクトルをはっきり区別するために、ベクトルを次のように行列の特別な場合として構成します。
- 横ベクトルを作る。たとえば横ベクトル

$$v = (a,b,c)$$

  を作るには `v={{a,b,c}}` とします。これは 1×3 の行列です。
- – しかし、日本の数学のテキスト（たとえば線形代数）は、そのほとんどが縦ベクトルを基本とする流儀で書かれています。したがって横ベクトルを使うことは少なく、むしろ次に述べる縦ベクトルの作成方法が基本的となります。
- 縦ベクトルを作る。たとえば縦ベクトル

$$v = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

  は `v={{a},{b},{c}}` と作ります。これは 3×1 の行列です。
- – カッコが多くて入力が面倒です。もうすこし楽に入力するためには、いったん横ベクトルを作ってから転置すればよいでしょう。すなわち `v=Transpose[{{a,b,c}}]` とします。あるいは、リストを分割するためのコマンド `Partition[]` を利用して `v=Partition[{a,b,c},1]` とすることもできます。

行列とベクトルに対する演算：　まず行列に対する演算としては、次のものが基本的です。

- 和と差はそれぞれ `+` と `-` です。
- 積は `.` です。もし記号 `*` で積をとると、これは各成分どうしの積（アダマール積）を計算してしまいますので注意しましょう。
- 逆行列は `Inverse[]` です。
- 転置行列は `Transpose[]` です。
- 行列式は `Det[]` です。
- 固有値と固有ベクトルを列挙するには `Eigensystem[]` です。

また、ベクトルに対しては次の演算があります。

- 外積 $v \times w$ は `Cross[v,w]` です。
- 内積 $\langle v, w \rangle$ は `v.w` です。これは行列どうしの積に使う記号 `.` と同じです。また、行列とベクトルの積にも同じ記号 `.` を使います。
- – とくにノルム $|v| = \sqrt{\langle v, v \rangle}$ は `Norm[v]` としても求められます。

.....

実習課題の提出方法：　実習課題のプログラムは、以下に述べる手順で FU ポータルから提出してください。

1. 実習課題のプログラムを書きます。プログラムの一行目はかならず `ClearAll["Global`*"]` です。
2. 不要な部分は削除して、他人が読みやすいように、プログラムを整形します。適宜コメントも挿入してください。記号 `(*` と記号 `*)` で囲まれた領域がコメント部分になります。そのあとで Mathematica のメニューから「評価」→「ノートブックを評価」を選択し、整形後のプログラムが正常に動くことを確認します。
3. 提出用のテキストファイルを作ります。Mathematica のメニューから「ファイル」→「別名で保存」を選択します。「フォーマット」は「テキスト `(*.txt)`」を選びます。ファイルの名前は任意です。ただしファイルの名前に空白およびドットを含めてはいけません。ドットはファイル名と拡張子の区切りを示すものです。
4. テキストファイルを FU ポータルから提出します。左側に表示されるメニューから「課題提出」を選び、さきほど保存したテキストファイルをアップロードします。提出の締め切り時刻は土曜日の 24 時 00 分です。それ以降は受け取りません。

評価基準はつぎの 4 点です。

- きちんと動くプログラムであること。
- 実習課題で要求された内容が達成されていること。
- 適宜コメント文があるなど、読みやすいコードであること。
- 指示されたとおりの手順で提出ファイルが作成されていること。

数式処理実習課題（2 回目）

1 $f_{ij} = \det \begin{pmatrix} a_i & a_j \\ b_i & b_j \end{pmatrix}$ とおく。このとき

$$f_{01}f_{23} - f_{02}f_{13} + f_{03}f_{12} = 0$$

であることを示せ。これをプリュッカーの関係式という。
ヒント：小さな添え字を使って f_{01} などと書いていますが、これは Mathematica で `f[0,1]` と入力して構いません。すなわち f を整数 i と整数 j の 2 変数関数と思うことにします。同様に a_i は `a[i]` と書けば十分です。

2 3 次元の実ベクトル $\boldsymbol{x}, \boldsymbol{y}$ に対して

$$|\boldsymbol{x}|^2 |\boldsymbol{y}|^2 = \langle \boldsymbol{x}, \boldsymbol{y} \rangle^2 + |\boldsymbol{x} \times \boldsymbol{y}|^2$$

が成立することを証明せよ。
ヒント①:「左辺 − 右辺」を計算してそれがゼロになればよいでしょう。
ヒント②：コマンド `Norm[]` は使わないほうが便利な場合もあります。
ヒント②の解説：ベクトル `x={a,b,c}` に対して `Norm[x]^2` と `x.x` は等しくありません。これは Mathematica が複素ベクトルのノルムを理解するからです。ただし、もし `x` が実ベクトルであることを Mathematica に教えた場合には `Norm[x]^2` と `x.x` は等しくなります。すなわち、`Simplify[Norm[x]^2, Element[x, Reals]]` と `x.x` は等価です。

3 実対称行列

$$A = \begin{pmatrix} a & b & c \\ b & a & b \\ c & b & a \end{pmatrix}$$

の固有値 $\lambda_1, \lambda_2, \lambda_3$ に属する固有ベクトルをそれぞれ $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$ とする。このとき $P = (\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$ とおけば

$$P^{-1}AP = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$$

となることを示せ。
ヒント①：`Eigensystem[A]` の出力がどういう形式か理解しておくことが必要です。たとえば実対称行列 `B={{1,2},{2,1}}` に対して `es=Eigensystem[B]` とすると、出力は `{{3,-1},{{1,1},{-1,1}}}` です。これは固有値 `3` に属する固有ベクトルが `{1,1}` であり、固有値 `-1` に属する固有ベクトルが `{-1,1}` であることを意味しています。また、固有値を並べたリスト `{3,-1}` は、リスト `es` の第 1 番目の要素なので `es[[1]]` とすれば取り出すことができます。同様に、固有ベクトルを並べたリスト `{{1,1},{-1,1}}` は、リスト `es` の第 2 番目の要素なので `es[[2]]` として取り出せます。
ヒント②：行列 P を入力したら、それが意図した行列になっているかどうか確かめましょう。きちんと $P = (\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$ になっていますか。もしかして P が

$$\begin{pmatrix} {}^{\mathrm{t}}\boldsymbol{v}_1 \\ {}^{\mathrm{t}}\boldsymbol{v}_2 \\ {}^{\mathrm{t}}\boldsymbol{v}_3 \end{pmatrix}$$

となっていませんか。

今日のテーマは因数分解です。

多項式の因数分解：

- 整数 n の素因数分解は `FactorInteger[n]` です。
- 多項式の因数分解には `Factor[]` を使います。
  - たとえば、多項式 x^2-1 を因数分解するには `Factor[x^2-1]` とします。出力は `(-1+x)*(1+x)` です。
  - つぎに、多項式 x^2-2 を因数分解してみます。答えは当然 $\left(x+\sqrt{2}\right)\left(x-\sqrt{2}\right)$ のはずですが、入力 `Factor[x^2-2]` に対する出力は `-2+x^2` です。つまり何もしてくれません。Mathematica は、デフォルトでは、係数が有理数となるような範囲でしか因数分解しないのです。これを 1 次式の積に因数分解するには

    ```
    Factor[x^2-2,Extension->Sqrt[2]]
    ```

    として、係数体である有理数体 $\mathbb{Q}$ に $\sqrt{2}$ を付け加え、係数体を拡大しておく必要があります。この「体（たい）」とか「体の拡大」という言葉の意味は、そのうち代数学の講義で勉強するでしょう。
  - 「体の拡大」というと何やら面倒そうですが、1 変数の複素係数 n 次多項式は、その根をすべて求めさえすれば常に 1 次式の積に分解できる（代数学の基本定理）ので、あまり気にする必要はありません。コマンド `Factor[]` が真にその威力を発揮するのは、多変数の多項式を因数分解する場合です。実習課題で取り組みましょう。

リスト操作：　いままでの実習課題をやってみて、おそらく多くの人が「リストの操作が上手にできれば作業が楽になる」ということに気づいたと思います。リストは Mathematica において最も重要なオブジェクトです。その使い方に習熟すれば、いろいろな作業が非常に効率よく進められるようになります。ここでは、もっとも基本的なリスト操作をまとめておきます。

- 1 次元リスト（ベクトル）`v={a,b,c,d}` があるとします。
  - `v[[3]]` とすると、第 3 番目の要素を返します。この場合 `c` です。
  - `Length[v]` はリスト `v` の長さを返します。この場合 `4` です。ここでいう「長さ」は、リストに含まれる要素の個数であり、ベクトルの大きさ（ノルム）とはまったく違いますので、混乱なきよう。
- 2 次元リスト（行列）`A={{a,b},{c,d}}` があるとします。
  - 行列は “リストのリスト” なので見づらいです。これを通常通り

    $$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

    とディスプレイに表示したいときは `MatrixForm[A]` とします。これに関して重要な注意をしておきます。リスト `A` を定義するときに、始めから `A=MatrixForm[{{a,b},{c,d}}]` としてはいけません。このように定義すると、見かけ上は `A` は行列ですが、計算上では `A` は行列として扱われません。たとえば

    ```
    A=MatrixForm[{{a,b},{c,d}}]
    A.A
    ```

    と

    ```
    B={{a,b},{c,d}}
    B.B
    ```

    の出力を比べてください。後者は行列の積が正しく計算されます。
  - `A[[2]]` は第 2 番目の要素を返すので、この場合は `{c,d}` です。
  - `A[[2]][[1]]` は第 2 番目の要素のなかの第 1 番目の要素を返すので、この場合は `c` です。これは短く `A[[2,1]]` とも書けます。
  - `Flatten[A]` とすると `A` を平坦化して `{a,b,c,d}` を返します。この命令はしばしば便利に使えますので覚えておきましょう。
- リスト `L={{a,b},{{x1,x2},{y1,y2}}}` があるとします。
  - `L[[1]]` は `{a,b}` であり、`L[[2]]` は `{{x1,x2},{y1,y2}}` です。したがって、たとえばリスト `{x1,x2}` を取り出したいときは `L[[2,1]]` とします。
  - `Flatten[L]` の出力は `{a,b,x1,x2,y1,y2}` です。もし、ここまで平坦化せずに、平坦化のレベルを深さ 1 で止めておきたい場合は `Flatten[L,1]` とします。出力は `{a,b,{x1,x2},{y1,y2}}` です。

実習課題

1 自然数 n に対して、n 次交代行列 $X=(x_{ij})$ の行列式は完全平方式となることが知られている。すなわち, n 次正方行列 X がもし $X+{}^{\mathrm{t}}X=0$ をみたすならばある多項式 p が存在して $\det X=p^2$ と書ける。この多項式 p を n 次のパフィアンという。6 次のパフィアンを求めよ。ただしパフィアンの符号は好きなように決めてよい。
ヒント①：6 次交代行列を楽に入力するには、どうすればよいですか。たとえば、まず行列 `X0=Table[x[i,j],{i,1,6},{j,1,6}]` を作っておいて、それを `X1=UpperTriangularize[X0]` と上三角化したものを用意しておくと便利です。
ヒント②：入力 `(a+b)^2/.{c_^2->c}` に対する出力は `a+b` です。

2 n 個の変数 $x_1, x_2, \ldots, x_n$ があるとする。このとき、n 次正方行列 A の第 i 行第 j 列目の成分を ${x_j}^{i-1}$ で与えると

$$\det A=\prod_{1\le i<j\le n}(x_j-x_i)$$

となる。これをファンデルモンドの行列式という。この公式が正しいことを $n=7$ のときに確かめよ。
ヒント①：右辺の積（これを差積といいます）は、まず添え字 j が $2\le j\le n$ の範囲を自由に動き、各 j の値に応じて添え字 i が $1\le i\le j-1$ の範囲をすべて動きます。いきなり差積を考えるのは大変なので、まずは差積に現れる因子をぜんぶ書き出してみましょう。たとえば `Table[x[j]-x[i],{j,2,n},{i,1,j-1}]` とします。
ヒント②：たとえば `v={a,b,c}` というリストがあったとき、すべての要素の積 `a*b*c` は `Product[v[[i]],{i,1,3}]` で計算できます。あるいは `Fold[Times,1,v]` とも書けます。後者のほうがスマート。

3 3 次実正方行列 Y が $Y=\left({}^{\mathrm{t}}Y\right)^{-1}$ かつ $\det Y=1$ をみたすとする。このとき、等式

$$Y=R_1(a)\,R_3(b)\,R_1(c)$$

をみたす実数 $a,b,c\in(-\pi,\pi]$ を行列 Y のオイラー角という。ただし

$$R_1(\theta)=\begin{pmatrix}1&0&0\\0&\cos\theta&-\sin\theta\\0&\sin\theta&\cos\theta\end{pmatrix},\ R_3(\theta)=\begin{pmatrix}\cos\theta&-\sin\theta&0\\\sin\theta&\cos\theta&0\\0&0&1\end{pmatrix}$$

である。次の問いに答えよ。
(1) すべての成分が整数、かつ行列式の値が正であるような 3 次正方行列 Z をひとつ与えよ。ただし単位行列は除く。
ヒント：たとえば `Z=RandomInteger[{0,3},{3,3}]` など。そのうえで行列式 `Det[Z]` が正かどうかチェック。もし正でなければ再度生成。
(2) 行列 Z の各列ベクトルを正規直交化して得られる行列を Y とおく。行列 Y が $Y=\left({}^{\mathrm{t}}Y\right)^{-1}$ かつ $\det Y=1$ をみたすことを確かめよ。
ヒント：`Y=Orthogonalize[Z]`
(3) 行列 Y のオイラー角を求めよ。
ヒント：入力 `MapThread[Equal,{{x,y},{p,q}}]` に対する出力は `{x==p,y==q}` です。

今日のテーマは冪級数です。

冪級数：

- 関数 $f(x)$ を $x=a$ において冪級数に展開すると

$$f(x)=f(a)+f'(a)(x-a)+\frac{f''(a)}{2}(x-a)^2+\mathrm{O}\left((x-a)^3\right)$$

となります。右辺の級数は `Series[f[x],{x,a,2}]` と書きます。

– `Series[]` の出力にはランダウの記号 O が現れます。このあと何らかの処理を続ける（たとえばグラフを描く）ために `Series[]` の出力を利用しようと思っても、ランダウの記号が邪魔してエラーになってしまいます。そこで、ランダウの記号を削除する（すなわち級数展開を途中で打ち切って多項式として出力する）ための命令 `Normal[]` が用意されています。例をあげましょう。たとえば、関数 $\exp(x)$ の $x=0$ における冪級数展開

$$1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}+\mathrm{O}\left(x^6\right)$$

を 5 次で打ち切れば、多項式

$$1+x+\frac{x^2}{2}+\frac{x^3}{6}+\frac{x^4}{24}+\frac{x^5}{120}$$

が得られます。これを便宜的に、関数 $\exp(x)$ の $x=0$ における 5 次の「テーラー多項式」と呼ぶことにします。ここに挙げたテーラー多項式は、冪級数 `ps=Series[Exp[x],{x,0,5}]` を多項式化したもの、すなわち `Normal[ps]` です。

恒等式：　恒等式を解くための命令として `SolveAlways[]` が用意されています。

– たとえば方程式 $ax+b=0$ がどんな x についても成立するためには $a=b=0$ であることが必要かつ十分です。これを Mathematica に解かせるには `SolveAlways[a*x+b==0,x]` とします。出力は `{{a->0,b->0}}` です。

.....

実習課題（今回はノートブック (`*.nb`) をそのまま提出します）

1 関数 $y=\sin(x)$ の $x=0$ における n 次のテーラー多項式を $s_n(x)$ とする。奇数 $n=2k+1$ $(0\le k\le 11)$ のそれぞれに対して、関数 $y=\sin(x)$ と関数 $y=s_n(x)$ のグラフを 1 枚のグラフ用紙に同時に描け。ただし描画範囲は $-4\pi\le x\le 4\pi$ とし、$y=\sin(x)$ のグラフは黒で、$y=s_n(x)$ のグラフは赤で表示すること。
ヒント①：`s[n_,x_]:=Normal[Series[Sin[x],{x,0,n}]]`
ヒント②：`Table[a[n],{n,1,2}]` の出力は `{a[1],a[2]}` です。もし `Table[{a[n],b},{n,1,2}]` なら `{{a[1],b},{a[2],b}}` です。
ヒント③：関数 $y=f(x)$ のグラフを黒で、関数 $y=g(x)$ のグラフを赤で描画するには

```
Plot[{f[x],g[x]},{x,-1,1},PlotStyle->{Black,Red}]
```

とします。次に、たとえば `list={{f[x],g[x]},{p[x],q[x]}}` という関数の組のリストがあるとします。このとき $y=f(x)$ のグラフを黒で、$y=g(x)$ のグラフを赤で表示しようと思って

```
Plot[list[[1]],{x,-1,1},PlotStyle->{Black,Red}]
```

としても、グラフは色分けされません。正しくは

```
Plot[Evaluate[list[[1]]],{x,-1,1},PlotStyle->{Black,Red}]
```

とします。このように、関数の組がリスト `list` で与えられているような場合は、いったん `list[[1]]` の内容を強制的に評価してからコマンド `Plot[]` に渡すことで、きちんと色分けされます。

2 開区間 $(-1,1)$ 上の連続関数 f に対して、ふたつの数列 a_n, b_n を

$$a_n=\frac{1}{\pi}\int_{-\pi}^{\pi}f(t)\cos(nt)\,dt,\quad b_n=\frac{1}{\pi}\int_{-\pi}^{\pi}f(t)\sin(nt)\,dt$$

と定める。このとき、各 $x\in(-1,1)$ について

$$f(x)=\frac{a_0}{2}+\sum_{n=1}^{\infty}\left(a_n\cos(nx)+b_n\sin(nx)\right)$$

と級数展開できることが知られている。これを関数 f のフーリエ展開という。また、フーリエ展開を途中で打ち切った式

$$f_m(x)=\frac{a_0}{2}+\sum_{n=1}^{m}\left(a_n\cos(nx)+b_n\sin(nx)\right)$$

を、便宜的に m 次のフーリエ三角多項式と呼ぶことにする。関数 $f(x)=x$ の場合に、そのフーリエ三角多項式の列 $f_1(x)$, $f_2(x)$, ... が関数 $f(x)$ に収束する様子をグラフィクスでわかりやすく表示せよ。
ヒント①：問 1 ができた人はこの問題もすぐにできます。たとえば、フーリエ三角多項式 $f_1(x),\dots,f_6(x)$ のそれぞれについて、そのグラフを関数 $f(x)$ のグラフと一緒に描けば、収束の様子がわかります。描画範囲やグラフの色は見やすいように適当に指定します。
ヒント②：フーリエ展開のための命令 `FourierTrigSeries[]` が用意されているので、数列 a_n, b_n を自分で定義する必要はありません。たとえば `FourierTrigSeries[f[x],x,m]` とすると、これは関数 `f[x]` の `m` 次のフーリエ三角多項式を返します。

3 多項式の列 p_0, p_1, ... を次のように定める。まず $p_0=1$, $p_1=x$ とし、$n\ge 2$ に対しては

$$p_n=\frac{2n-1}{n}xp_{n-1}-\frac{n-1}{n}p_{n-2}$$

とする。多項式 p_n を n 次のルジャンドル多項式という。次の問いに答えよ。
(1) ルジャンドル多項式 p_0, ..., p_{10} をリストとして表示せよ。
ヒント：ルジャンドル多項式を定義するには

```
p[0]=1;
p[1]=x;
p[n_]:=(2*n-1)*x*p[n-1]/n-(n-1)*p[n-2]/n
```

とします。最後の式は `=` でなく `:=` で定義する必要があります。実際、最後の式を `=` で定義するとエラーが出ることを確かめてください。両者の違いについては来週説明します。
(2) ルジャンドル多項式は、等式

$$\frac{1}{\sqrt{1-2xt+t^2}}=\sum_{n=0}^{\infty}p_nt^n$$

をみたす。これをルジャンドル多項式の母関数表示という。この母関数表示を利用して p_0, ..., p_{10} を求め、それが (1) で求めたものと一致することを確かめよ。
ヒント①：母関数表示の意味を説明します。これは変数 t についての恒等式で、左辺を変数 t について冪級数展開したとき、その係数たちがルジャンドル多項式となっている、ということです。したがって、左辺については $t=0$ における 10 次のテーラー多項式を求めます。右辺の和は、高次の項を打ち切って $0\le n\le 10$ の範囲で考えればよいでしょう。和は、たとえば `Sum[a[n],{n,1,2}]` とすれば `a[1]+a[2]` を計算します。
ヒント②：入力 `{a,b}-{c,d}` に対する出力は `{a-c,b-d}` です。

実習課題の提出要領を再掲します。

実習課題の提出要領：　実習課題のプログラムは、以下に述べる手順で FU ポータルから提出してください。

1. 実習課題のプログラムを書きます。プログラムの一行目はかならず `ClearAll["Global`*"]` です。
2. 不要な部分は削除して、他人が読みやすいように、プログラムを整形します。適宜コメントも挿入してください。記号 `(*` と記号 `*)` で囲まれた領域がコメント部分になります。そのあとで Mathematica のメニューから「評価」→「ノートブックを評価」を選択し、整形後のプログラムが正常に動くことを確認します。
3. 提出用のテキストファイルを作ります。Mathematica のメニューから「ファイル」→「別名で保存」を選択します。「フォーマット」は「テキスト (*.txt)」を選びます。ファイルの名前は任意です。ただしファイルの名前に空白およびドットを含めてはいけません。ドットはファイル名と拡張子の区切りを示すものです。
4. テキストファイルを FU ポータルから提出します。左側に表示されるメニューから「課題提出」を選び、さきほど保存したテキストファイルをアップロードします。提出の締め切り時刻は土曜日の 24 時 00 分です。それ以降は受け取りません。

評価基準はつぎの 4 点です。

- きちんと動くプログラムであること。
- 実習課題で要求された内容が達成されていること。
- 適宜コメント文があるなど、読みやすいコードであること。
- 指示されたとおりの手順で提出ファイルが作成されていること。

注意：　実習課題を採点してみて、いくつか気になったことを注意します。まずは全員向けの注意から。

– プログラムを提出するときは、不要な部分を削除して、読みやすいコードにしてください。また、出力をとくに必要としない部分には、行末にセミコロン ; を付けて出力を抑制してください。
– 上記「実習課題の提出要領」で作成したテキストファイルをエディタで編集してはいけません。そのまま提出してください。とくに `In[]` と `Out[]` を削除してはいけません。書かれたプログラムが入力なのか出力なのか区別できなくなります。
– ファイル名に空白を含めるのはやめましょう。これは思わぬトラブルの原因となります。どうしてもファイル名に空白を使いたい場合でも、なるべくアンダースコアやハイフンで代用します。

つぎに少数の人向けの注意を。

– コメントの領域は、記号 `(*` と記号 `*)` で囲みます。これを全角文字で （* と *） にしたり、あるいはカッコを変えて <* と *> で入力するなど、そのほか勝手な記号に変更することはできません。
– プログラムの一行目はかならず `ClearAll["Global`*"]` です。ここで使われている逆引用符の記号 ` は、ふつうの引用符の記号 ' とは違うものです。また `Global` を `global` と書くことはできません。Mathematica のコマンドは常に大文字で始まります。
– 代入のルールを記述するとき、たとえば `x/.{x->1}` と書きますが、この記号 `->` を全角文字の矢印 → で置き換えて `x/.{x→1}` と入力してはいけません。このように書くと、当然ながらプログラムがエラーを起こします。
– 行列の積は `A.B` です。これを `A*B` と書いてはいけません。これはアダマール積といって、成分毎に積をとるだけの演算です。行列の積とはまったく違うものです。
– 「行列式」と「逆行列」の用語が混乱している人がいます。これは数学科として恥ずかしすぎます。数学の内容をきちんと理解している人は決して用語を間違えません。

提出ファイルの例：　提出ファイルのサンプルを載せておきます。たとえば前回の実習課題については、次のようなプログラムが書ければオーケーです。

```
In[1]:= (*---------------第 1 問---------------*)
In[2]:= ClearAll["Global`*"]
In[3]:= (*6 次交代行列 X を作る*)
In[4]:= X0=Table[x[i,j],{i,1,6},{j,1,6}];
In[5]:= X1=UpperTriangularize[X0];
In[6]:= X=X1-Transpose[X1];
In[7]:= (*X の行列式を因数分解する*)
In[8]:= q=Factor[Det[X]];
In[9]:= (*パフィアンをとりだす*)
In[10]:= q/.{x_^2->x}
Out[10]= x[1,6] x[2,5] x[3,4]-x[1,5] x[2,6] x[3,4]
-x[1,6] x[2,4] x[3,5]+x[1,4] x[2,6] x[3,5]+x[1,5] x[2,4] x[3,6]
-x[1,4] x[2,5] x[3,6]+x[1,6] x[2,3] x[4,5]-x[1,3] x[2,6] x[4,5]
+x[1,2] x[3,6] x[4,5]-x[1,5] x[2,3] x[4,6]+x[1,3] x[2,5] x[4,6]
-x[1,2] x[3,5] x[4,6]+x[1,4] x[2,3] x[5,6]-x[1,3] x[2,4] x[5,6]
+x[1,2] x[3,4] x[5,6]
In[11]:= (*---------------第 2 問---------------*)
In[12]:= ClearAll["Global`*"]
In[13]:= n=7;
In[14]:= (*差積の因子を列挙する*)
In[15]:= d=Flatten[Table[x[j]-x[i],{j,2,n},{i,1,j-1}]];
In[16]:= (*差積を計算する*)
In[17]:= pd=Fold[Times,1,d];
In[18]:= (*ファンデルモンド行列を作る*)
In[19]:= A=Table[x[j]^(i-1),{i,1,n},{j,1,n}];
In[20]:= (*ファンデルモンド行列の行列式が差積に等しいことをチェック*)
In[21]:= Simplify[Det[A]-pd]
Out[21]= 0
In[22]:= (*---------------第 3 問---------------*)
In[23]:= ClearAll["Global`*"]
In[24]:= (*行列式が正の整数行列 Z を与える*)
In[25]:= Z={{1,2,3},{2,1,2},{3,2,1}};
In[26]:= (*実際 Z の行列式が正であることをチェック*)
In[27]:= Det[Z]>0
Out[27]= True
In[28]:= (*Z を正規直交化した行列を Y とする*)
In[29]:= Y=Orthogonalize[Z];
In[30]:= (*Y が直交行列であることをチェック*)
In[31]:= Y-Inverse[Transpose[Y]]
Out[31]= {{0,0,0},{0,0,0},{0,0,0}}
In[32]:= (*Y の行列式が 1 であることをチェック*)
In[33]:= Det[Y]-1
Out[33]= 0
In[34]:= (*回転行列を定義*)
R1[x_]={{1,0,0},{0,Cos[x],-Sin[x]},{0,Sin[x],Cos[x]}};
R3[x_]={{Cos[x],-Sin[x],0},{Sin[x],Cos[x],0},{0,0,1}};
In[36]:= (*オイラー角をもとめるための方程式を準備*)
In[37]:= y=Flatten[Y];
r=Flatten[R1[a].R3[b].R1[c]];
eq=MapThread[Equal,{y,r}];
In[40]:= (*オイラー角をもとめる*)
In[41]:= sol=Solve[eq,{a,b,c}];
In[42]:= (*-Pi から Pi の間にあるような角を選ぶ*)
In[43]:= euler=sol/.{C[n_]->0}
Out[43]= {{a->ArcTan[Sqrt[7/2]/9],b->ArcTan[Sqrt[13]],
c->π-ArcTan[3/2]},{a->-π +ArcTan[Sqrt[7/2]/9],
b->-ArcTan[Sqrt[13]],c->-ArcTan[3/2]}}
In[44]:= (*-Pi から Pi の間にあることをチェック*)
In[45]:= IntervalMemberQ[Interval[{-Pi,Pi}],{a,b,c}/.euler]
Out[45]= {{True,True,True},{True,True,True}}
```

今日のテーマは微分です。

微分：

- `D[f[x],x]` とすれば、これは 1 階導関数 $f'(x)$ を意味します。
- 高階の導関数は、たとえば 3 階なら `D[f[x],{x,3}]` とします。
- 偏微分をする場合にも同じ記号 `D` を使います。たとえば $f_x(x,y)$ なら `D[f[x,y],x]` とします。高階の偏導関数の場合は、たとえば $f_{xyy}(x,y)$ なら `D[f[x,y],x,{y,2}]` などと書きます。
- Mathematica は「偏微分の順番」を気にしません。したがって解析学のデリケートな問題に関して、しばしば嘘をつきますので注意しましょう。たとえば関数 $f\colon \mathbb{R}^2 \to \mathbb{R}$ を

$$f(x,y) = \begin{cases} \dfrac{xy\left(x^2-y^2\right)}{x^2+y^2} & (x,y) \neq (0,0) \text{ のとき} \\ 0 & (x,y) = (0,0) \text{ のとき} \end{cases}$$

で定めると、これは $f_{xy}(0,0) \neq f_{yx}(0,0)$ となる有名な例ですが、Mathematica で計算すると $f_{xy}(0,0) = f_{yx}(0,0)$ となってしまいます。

即時評価と遅延評価：　ここで即時評価と遅延評価について説明します。たとえば関数 $f(x) = x^2$ を定義するときには `f[x_]=x^2` あるいは `f[x_]:=x^2` と書きますが、このように、Mathematica は `=` と `:=` という 2 種類の代入演算子をもっています。両者の違いは、右辺を評価するタイミングにあります。演算子 `=` は即時型の代入を表し、右辺は定義時にすぐに評価されます。これに対し、演算子 `:=` は遅延型の代入を表し、右辺は、左辺の値が要求されるごとに毎回評価されます。例をみましょう。

- 即時評価と遅延評価の違い。たとえば

```
a=RandomReal[]
b:=RandomReal[]
```

とします。コマンド `RandomReal[]` は 0 と 1 の間の実数をランダムに生成します。このとき `{a,a}` は同じ値が並びますが、`{b,b}` は異なる値が並びます。また `{a,a}` は何度評価しなおしても常に同じ値しか表示されませんが、`{b,b}` は評価するたびに毎回違う値が表示されます。

関数 $f(x) = x^2$ を定義する場合には、即時的に `f[x_]=x^2` と書いても遅延的に `f[x_]:=x^2` と書いても、両者のあいだに大した差はありません。ところが、次のような場合には、即時評価と遅延評価を明確に使い分ける必要があります。

- 即時評価を使わなければならない例。たとえば

```
f[x_]=D[Sin[x],x] (*これは f[x_]=Cos[x] と書くのと同じ*)
g[x_]:=D[Sin[x],x]
```

とします。このとき `f[0]` は正常に評価され、出力が `1` となりますが、`g[0]` を評価すると「`0` は有効な変数ではありません」というエラーが出ます。なぜなら `g[0]` の値が要求されたとき Mathematica は式 `D[Sin[0],0]` を評価しますが、`Sin[0]` を '変数' `0` で微分することはできないため先のようなエラーが出るのです。また関数 `f` は `Plot[f[x],{x,-Pi,Pi}]` でグラフが描けますが、関数 `g` のグラフは `Plot[g[x],{x,-Pi,Pi}]` としても描けません。ただし無理矢理 `Plot[Evaluate[g[x]],{x,-Pi,Pi}]` とすれば `g` のグラフも描けます。

- 遅延評価を使わなければならない例。たとえば数列を再帰的に定義するような場合は、遅延評価を使います。例として、自然数の階乗を定義してみましょう。数列 $a_n = n!$ が、漸化式 $a_n = na_{n-1}$ をみたすことから

```
a[1]=1
a[n_]:=n*a[n-1]
```

と定義します。もしこの 2 行目を即時的に `a[n_]=n*a[n-1]` と書いてしまうと「最大再帰回数 1024 を超えています」というエラーがでます。右辺の `a[n-1]` の評価がいつまでたっても終わらないからです。

実習課題

1 ルジャンドル多項式（定義は前回の実習課題参照）について次の問に答えよ。
(1) n 次のルジャンドル多項式は

$$p_n = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left(\left(x^2-1\right)^n\right)$$

と書ける。これをロドリーグの公式という。この公式が正しいことを $0 \leq n \leq 10$ について確かめよ。
ヒント：まずは右辺を `n` の関数として定義しましょう。`?Factorial`
(2) n 次のルジャンドル多項式は微分方程式

$$\left(1-x^2\right)\frac{d^2}{dx^2}p_n - 2x\frac{d}{dx}p_n + n\left(n+1\right)p_n = 0$$

をみたす。これを $0 \leq n \leq 10$ について確かめよ。
ヒント：こんどは左辺を `n` の関数として定義します。定義は即時的にすべきですか、あるいは遅延的にすべきですか。

2 関数 f に対して、関数 S_f を

$$S_f(x) = \frac{f'''(x)}{f'(x)} - \frac{3}{2}\left(\frac{f''(x)}{f'(x)}\right)^2$$

と定める。関数 S_f のことを関数 f のシュワルツ微分という。
(1) 関数 $h(x) = x/\left(x-1\right)^2$ のシュワルツ微分 S_h を求めよ。
(2) シュワルツ微分は 1 次分数変換で不変であることを証明せよ。すなわち、関数 f と定数 a,b,c,d に対して

$$h(x) = \frac{af(x)+b}{cf(x)+d}$$

とおくと、$S_h(x) = S_f(x)$ となることを示せ。
(3) 合成関数 $f \circ g$ のシュワルツ微分が

$$S_{f\circ g}(x) = S_f\left(g(x)\right)\left(g'(x)\right)^2 + S_g(x)$$

となることを示せ。ヒント：`?Composition`

3 a を x,y の関数とする。関数列 τ_{-1}, τ_0, $\tau_1, \tau_2, \ldots$ を次のように定める。まず $\tau_{-1} = 0$, $\tau_0 = 1$ とし、自然数 n に対しては

$$\tau_n = \det\left(\frac{\partial^{i+j-2}}{\partial x^{i-1}\partial y^{j-1}}a\left(x,y\right)\right)_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$$

とする。このとき、すべての非負整数 m に対して偏微分方程式

$$\tau_m\frac{\partial^2}{\partial x\partial y}\tau_m - \frac{\partial}{\partial x}\tau_m \cdot \frac{\partial}{\partial y}\tau_m - \tau_{m+1}\tau_{m-1} = 0$$

が成り立つことが知られている。これを戸田分子方程式という。戸田分子方程式が成り立つことを $0 \leq m \leq 5$ について示せ。

来週（5 月 15 日）は休講です。7 月 10 日と 7 月 17 日も休講の予定。

今日のテーマはベクトル場です。

ベクトル場：　平面内の各点 p において、その点における速度ベクトル $v(p)$ が定まっているとき、この対応 $v\colon \mathbb{R}^2 \to \mathbb{R}^2$ をベクトル場（正確には速度ベクトル場）といいます。次に、質点がベクトル場に沿って流れていく様子をイメージしましょう。その質点の軌跡を積分曲線といいます。ベクトル場や積分曲線を理解するには絵を描いてみるのがいちばんです。そのためのコマンドが `VectorPlot[]` と `StreamPlot[]` です。

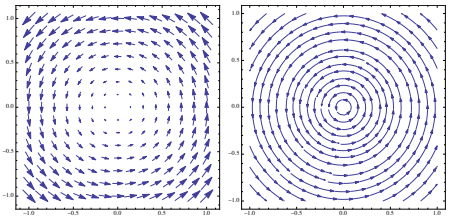
– たとえば点 ${}^{\mathrm{t}}(x,y) \in \mathbb{R}^2$ における速度ベクトル $v\,(x,y)$ が

$$v\,(x,y) = \begin{pmatrix} -y \\ x \end{pmatrix}$$

と与えられているとします。このとき

```
VectorPlot[{-y,x},{x,-1,1},{y,-1,1}]
StreamPlot[{-y,x},{x,-1,1},{y,-1,1}]
```

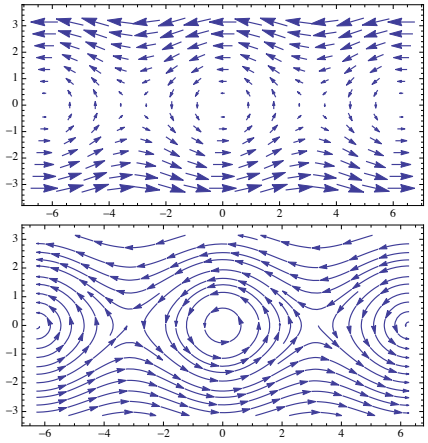
とすれば、次のような図が出力されます。左がベクトル場で、右がその積分曲線です。水の流れをイメージしてください。



– ベクトル場を変えれば、水の流れも変わります。たとえば

$$v\,(x,y) = \begin{pmatrix} -y \\ \sin x \end{pmatrix}$$

なら、ベクトル場と積分曲線は次のようになります。



これらの図の意味をもう少し考えましょう。平面内のどこか好きな位置に質点をおいてみます。平面内には、ベクトル場にしたがって水が流れているので、質点はその流れにのって動いていきます。このことを数式で書いてみます。はじめの「好きな位置」を ${}^{\mathrm{t}}(a,b)$ とします。また、時刻 t における質点の位置を ${}^{\mathrm{t}}(x(t),y(t))$ とします。とくに

$$\begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

です。さて、時刻 t における質点の速度ベクトル ${}^{\mathrm{t}}(x'(t),y'(t))$ は、あらかじめ与えられたベクトル場 v によって決まっています。すなわち

$$\begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = v\,(x(t),y(t))$$

です。これらを連立した式

$$\begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = v\,(x(t),y(t)), \quad \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

を「1 階常微分方程式系の初期値問題」といいます。もっと具体的に書くと、たとえば、先に挙げたひとつめの例ならば初期値問題は

$$\begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = \begin{pmatrix} -y(t) \\ x(t) \end{pmatrix}, \quad \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

となります。あるいは、ふたつめの例ならば

$$\begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = \begin{pmatrix} -y(t) \\ \sin x(t) \end{pmatrix}, \quad \begin{pmatrix} x(0) \\ y(0) \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} \tag{1}$$

です。このように、ベクトル場は 1 階常微分方程式系（の初期値問題）を定めます。逆に、1 階の常微分方程式系からただちにベクトル場が決まります。したがって、ベクトル場を考えることと 1 階の常微分方程式系を考えることは等価であり、両者は同じものです。言い方を変えます。1 階の常微分方程式系（の初期値問題）は、わざわざ解かなくても、ベクトル場さえ描ければ、解の振る舞いがすべて視覚的にわかってしまうのです。

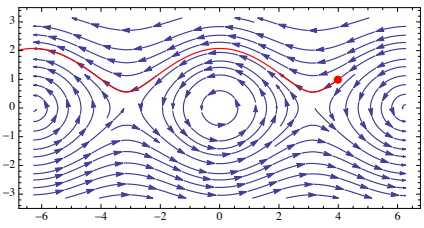
解曲線：　とは言いながら、やはり、答えを「式」として知りたい場合もあります。そのようなときは、初期値問題を解かなければなりません。そのためのコマンドが `NDSolve[]` です。たとえば、初期値問題 (1) を $a=4$, $b=1$ として時間 $0 \le t \le 10$ の範囲で解くには

```
eq={x'[t]==-y[t],y'[t]==Sin[x[t]]};
iv={x[0]==4,y[0]==1};
sol=NDSolve[Flatten[{eq,iv}],{x[t],y[t]},{t,0,10}]
```

とします。また、初期値問題の積分曲線（これをとくに解曲線や解軌道ともいいます）を描くには、次のようにします。

```
p[t_]={x[t],y[t]}/.sol[[1]];
ParametricPlot[p[t],{t,0,10}]
```

積分曲線（青）と初期値問題の解軌道（赤）を同時に描いたのが次の図です。赤色の点が初期値です。赤色の点が流れにのって移動していくようすがわかります。これが「初期値問題を解く」ということです。



実習課題：　時刻 t におけるウサギとキツネの個体数を、それぞれ $x(t)$, $y(t)$ とする。ウサギは被食者でありキツネは捕食者である。両者の個体数の変化は、おおむね微分方程式

$$\begin{cases} x'(t) = x(t)\,(a - by(t)) \\ y'(t) = -y(t)\,(c - dx(t)) \end{cases}$$

でモデル化できることが知られている。これをロトカ・ヴォルテラ方程式という。定数を $a=c=1$, $b=0.04$, $d=0.02$ とし、初期値を $x(0)=100$, $y(0)=10$ とする。

(1) ロトカ・ヴォルテラ方程式のベクトル場と積分曲線を描け。

(2) 時間 $0 \le t \le 20$ のあいだでウサギとキツネの個体数がどのように変化するか、初期値問題を解いて解軌道を図示せよ。

(3) その結果（解軌道が閉曲線になる）が何を意味するか、言葉で述べよ。

今日のテーマは微分方程式の解法です。今後の予定は、次回「曲面」、その次「アニメーション」、その次「複雑な処理」（これは変更するかもしれません）、最終回「TEX」です。

微分方程式の解析解：　微分方程式の解を求めるためのコマンドが `DSolve[]` です。たとえば、初期値問題

$$x'(t) = x(t), \quad x(0) = 1$$

を解くには `DSolve[{x'[t]==x[t],x[0]==1},x[t],t]` とします。出力はもちろん `{{x[t]->Exp[t]}}` です。

微分方程式の数値解法：　ところが、ほとんどすべての微分方程式はその解を初等関数で表すことができません。これは努力が足りないからできないのではなく、不可能であることが証明されています。そこで、解析解（微分方程式をみたす関数）を求めるのは諦めて、かわりに数値解（微分方程式を近似的にみたす数値の列）を求めます。いったん数値解を求めることができれば、それをグラフィクスで可視化するなどして、解の振る舞いを理解することができます。このようにして微分方程式を調べる方法を、微分方程式の数値解法といいます。さて、数値解法の手順は

1. まず、微分方程式を離散化して適当な差分方程式を導出し、
2. 次に、その差分方程式の数値解を求めます。

差分方程式を数値的に解くこと自体は簡単なので、数値解法の肝は手順 1 にあります。微分方程式の離散化の仕方はいろいろなテクニックが知られています。ここではロトカ・ヴォルテラ方程式

$$\begin{cases} x'(t) = x(t)\,(2 - y(t)) \\ y'(t) = -y(t)\,(3 - 2x(t)) \end{cases}$$

を例にとって、もっともよく知られた 2 種類の離散化のスキーム（オイラー法とルンゲ・クッタ法）を紹介します。初期値は

$$x(0) = 2, \quad y(0) = 1$$

としておきましょう。差分間隔を表す正定数を ϵ とし、それをあらかじめ適当な値に固定しておきます（たとえば $\epsilon = 0.1$ など）。

- オイラー法。これは最も単純な離散化のスキームです。微分を前進差分におきかえることによって、差分方程式を導出する方法です。すなわち、ロトカ・ヴォルテラ方程式に対して差分方程式系

$$\begin{cases} \dfrac{x_{n+1} - x_n}{\epsilon} = x_n\,(2 - y_n) \\ \dfrac{y_{n+1} - y_n}{\epsilon} = -y_n\,(3 - 2x_n) \end{cases}$$

を考えます。整理して

$$\begin{cases} x_{n+1} = x_n + \epsilon x_n\,(2 - y_n) \\ y_{n+1} = y_n - \epsilon y_n\,(3 - 2x_n) \end{cases}$$

です。この差分方程式系は初期値 $x_0 = 2$, $y_0 = 1$ から次々に値が決まっていきます。プログラムはたとえば次のように書けばよいでしょう。

```
e=0.1;
max=37;
x[n_]:=x[n-1]+e*x[n-1]*(2-y[n-1]);
y[n_]:=y[n-1]-e*y[n-1]*(3-2*x[n-1]);
x[0]=2;
y[0]=1;
list=Table[{x[n],y[n]},{n,0,max}];
ListPlot[list,Joined->True,PlotMarkers->Automatic]
```

ところが、実行してみると分かりますが、このプログラムは計算にとても時間がかかってしまうため、実用的ではありません。再帰的な計算をするのに無駄が多いせいです。そこで数列の再帰計算に特化したコマンド `RecurrenceTable` を使います。このコマンドを利用すると、瞬時に計算が終わります。

```
e=0.1;
max=37;
system={x[n+1]==x[n]+e*x[n]*(2-y[n]),
        y[n+1]==y[n]-e*y[n]*(3-2*x[n]),
        x[0]==2,
        y[0]==1};
list=RecurrenceTable[system,{x,y},{n,0,max}];
ListPlot[list,Joined->True,PlotMarkers->Automatic]
```

こうした離散的な計算は Mathematica の得意分野です。関連する話題については Mathematica のメニューから「ヘルプ」→「ドキュメントセンター」をたどり、検索窓に「離散微積分」または「DiscreteCalculus」を入力してみてください。

- ルンゲ・クッタ法。数値解法というのは近似解法なので、どのような数値解法も誤差は免れません。そこで近似の精度を上げるために、いろいろの数値解法が提案されています。なかでもルンゲ・クッタ法はオイラー法にくらべて精度がずっと高く、汎用的な離散化手法として有名です。ルンゲ・クッタ法は、ロトカ・ヴォルテラ方程式を

$$\begin{cases} \dfrac{x_{n+1} - x_n}{\epsilon} = \dfrac{F_1 + 2F_2 + 2F_3 + F_4}{6} \\ \dfrac{y_{n+1} - y_n}{\epsilon} = \dfrac{G_1 + 2G_2 + 2G_3 + G_4}{6} \end{cases}$$

のように離散化します。左辺の微分は、オイラー法と同様、前進差分によって素直に離散化していますが、右辺の離散化がテクニカルです。ここに現れる F_1, F_2, F_3, F_4 と G_1, G_2, G_3, G_4 はそれぞれ次の手順によって定められる数列です。記述を簡単にするために

$$f(x, y) = x\,(2 - y)$$
$$g(x, y) = -y\,(3 - 2x)$$

とします。このとき、まず

$$F_1 = f\,(x_n, y_n)$$
$$G_1 = g\,(x_n, y_n)$$

とおいて、次に

$$F_2 = f\left(x_n + \frac{\epsilon}{2}F_1,\ y_n + \frac{\epsilon}{2}G_1\right)$$
$$G_2 = g\left(x_n + \frac{\epsilon}{2}F_1,\ y_n + \frac{\epsilon}{2}G_1\right)$$

とし、その次に

$$F_3 = f\left(x_n + \frac{\epsilon}{2}F_2,\ y_n + \frac{\epsilon}{2}G_2\right)$$
$$G_3 = g\left(x_n + \frac{\epsilon}{2}F_2,\ y_n + \frac{\epsilon}{2}G_2\right)$$

とおいて、最後に

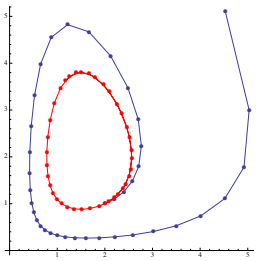
$$F_4 = f\,(x_n + \epsilon F_3,\ y_n + \epsilon G_3)$$
$$G_4 = g\,(x_n + \epsilon F_3,\ y_n + \epsilon G_3)$$

とします。手順がやや複雑ですが F_1, F_2, F_3, F_4 と G_1, G_2, G_3, G_4 はすべて x_n と y_n だけから決まっている（x_{n+1} や y_{n+1} は現れな

い）ことに注意しましょう。したがって、ルンゲ・クッタ法によって導出した差分方程式系は、初期値 $x_0 = 2$, $y_0 = 1$ から次々に値が決まっていきます。ルンゲ・クッタ法のプログラムはたとえば次のように書けばよいでしょう。

```
e=0.1;
max=37;
f[1,x_,y_]=x*(2-y);
f[2,x_,y_]=-y*(3-2*x);
initx=2;
inity=1;
F1[m_,x_,y_]=f[m,x,y];
F2[m_,x_,y_]=f[m,x+e*F1[1,x,y]/2,y+e*F1[2,x,y]/2];
F3[m_,x_,y_]=f[m,x+e*F2[1,x,y]/2,y+e*F2[2,x,y]/2];
F4[m_,x_,y_]=f[m,x+e*F3[1,x,y],y+e*F3[2,x,y]];
system={x[n+1]==x[n]+e*(F1[1,x[n],y[n]]
                        +2*F2[1,x[n],y[n]]
                        +2*F3[1,x[n],y[n]]
                        +F4[1,x[n],y[n]])/6,
        y[n+1]==y[n]+e*(F1[2,x[n],y[n]]
                        +2*F2[2,x[n],y[n]]
                        +2*F3[2,x[n],y[n]]
                        +F4[2,x[n],y[n]])/6,
        x[0]==initx,
        y[0]==inity};
list=RecurrenceTable[system,{x,y},{n,0,max}];
ListPlot[list,Joined->True,PlotMarkers->Automatic]
```

- 数値解の比較。次のグラフは、オイラー法による数値解（青色）とルンゲ・クッタ法による数値解（赤色）を示したものです。



– 同一の初期値 $x_0 = 2$, $y_0 = 1$ から始めていますが、ふたつの曲線はかなり異なった様子を示しています。青色の曲線と赤色の曲線、どちらが正しい答えでしょうか。以下に述べるように、ロトカ・ヴォルテラ方程式の解は常に周期的になる（すなわち解軌道が閉曲線となる）ことが証明できるので、正しい答えは赤色の曲線です。

– ロトカ・ヴォルテラ方程式を変形して

$$(\log x(t))' = 2 - y(t), \quad (\log y(t))' = -3 + 2x(t)$$

なので、とくに

$$(-3 + 2x(t))\,(\log x(t))' - (2 - y(t))\,(\log y(t))' = 0$$

が成りたちます。この両辺は t について積分できて

$$-3\log x(t) + 2x(t) - 2\log y(t) + y(t) = C$$

です。したがって解軌道は、初期値で決まる C の値を保つような閉曲線

$$\begin{aligned} &-3\log x(t) + 2x(t) - 2\log y(t) + y(t) \\ &\quad = -3\log x(0) + 2x(0) - 2\log y(0) + y(0) \end{aligned}$$

となります。実際これが閉曲線であることを確かめるには、定数 C の値をひとつ固定したときに、集合

$$S = \left\{ {}^{\mathrm{t}}(x, y) \in \mathbb{R}^2 \;\middle|\; -3\log x + 2x - 2\log y + y = C \right\}$$

が有界であることを証明すればよいでしょう（省略）。この議論ではロトカ・ヴォルテラ方程式をまったく解いていないにもかかわらず、解の性質（周期解になる）がわかってしまったことに注意します。このように、微分方程式を解かずに微分方程式の解の挙動を調べる研究分野を力学系といいます。

– 一般にルンゲ・クッタ法はオイラー法に比べてよい結果を示します。詳しくは誤差の評価をしなければなりませんが、オイラー法が 1 次の精度しかないのに対し、ルンゲ・クッタ法は 4 次の精度をもっています。

以上、数値解法の基礎を学びました。プログラムを書くのが大変だと感じた人もいるでしょう。もっと手軽に数値解法を利用したいひとのために、Mathematica では `NDSolve[]` という命令が用意されています。この命令を使うと、ユーザーは離散化の手続きを一切気にすることなく、微分方程式の数値解を求めることができます。

実習課題：　次のどちらかの問を選択せよ。おすすめは B。

A ロジスティック方程式

$$x'(t) = x(t)\,(1 - x(t))\,, \quad x(0) = \frac{1}{2}$$

について次の問に答えよ。
(1) 解析解を求めよ。また $0 \le t \le 30$ の範囲でグラフを描け。ヒント：グラフは `Plot` のオプションに `PlotRange->{0,1}` をつける。
(2) ロジスティック方程式をオイラー法によって

$$x_{n+1} = x_n + \epsilon\, x_n\,(1 - x_n)\,, \quad x_0 = 0.5$$

と離散化する。定数 ϵ の値によって数値解の性質が変化することをみよう。
(i) $\epsilon = 0.1$ のとき数値解 x_n を $0 \le n \le 1000$ の範囲でプロットせよ。数値解がロジスティック方程式の解析解とよく似た挙動をみせることがわかる。
(ii) $\epsilon = 3$ のとき数値解 x_n を $0 \le n \le 1000$ の範囲でプロットせよ。数値解はカオス的になり、ロジスティック方程式の解析解とはまったく異なる挙動をみせることがわかる。
(3) ロジスティック方程式をルンゲ・クッタ法によって離散化し、数値解をプロットせよ。また、差分間隔 ϵ の値によって数値解の性質が変化するかどうか確かめよ。

B メトロノームのシミュレーションをしてみましょう。
(1) 単振動（水平ばね振り子）の微分方程式

$$m\,x''(t) = -k\,x(t)$$

について次の問に答えよ。ただし $k/m = 1/2$ とする。
(i) 解析解を求めよ。
(ii) 位置 $x(t)$ に対して速度 $y(t) = x'(t)$ を考える。このとき位置と速度の組 $(x(t), y(t))$ をプロットした平面を相平面という。相平面内のベクトル場とその積分曲線を描け。ヒント：位置 $x(t)$ と速度 $y(t)$ は、微分方程式系

$$\begin{cases} x'(t) = y(t) \\ y'(t) = -(k/m)\,x(t) \end{cases}$$

をみたす。ベクトル場と積分曲線の描きかたは前回のプリント参照。
(2) 単振動（粘性抵抗のある水平ばね振り子）の微分方程式

$$m\,x''(t) = -k\,x(t) - 2c\,x'(t), \quad k/m = 1/2, \quad 2c/m = 1/\sqrt{2}$$

に対して、その相平面内のベクトル場と積分曲線を描け。また、初期値を

$$(x(0), y(0)) = (0, -2)\,,\ (0, -1)\,,\ (0, 1)\,,\ (0, 2)$$

としたときの解軌道を描け。解軌道が急激に原点に吸い込まれる様子がわかる。ヒント：まず位置と速度がみたす微分方程式系を手で求める（簡単）。解軌道の描き方は前回のプリント参照。
(3) メトロノームは、基本的には減衰振動子だが、針が中心付近を通過するときに、その運動方向に力が加わるような仕組みがある。これを記述する微分方程式として次のものが考えられる。

$$m\,x''(t) = -k\,x(t) - 2c\,x'(t) + p\left(x(t), x'(t)\right)$$

ただし関数 p は $x = 0$ 付近で弾く力を表す。以下では $m = 1$, $k = 1$, $c = 0.2$ とし、関数 p を

$$p\left(x, x'\right) = \begin{cases} q(x) & (\text{ただし } x' > 0 \text{ のとき}) \\ -q(-x) & (\text{ただし } x' < 0 \text{ のとき}) \end{cases}$$

$$q(x) = \begin{cases} 0 & (\text{ただし } x < 0.01 \text{ のとき}) \\ 8x - 2/25 & (\text{ただし } 0.01 \le x < 0.06 \text{ のとき}) \\ -8x + 22/25 & (\text{ただし } 0.06 \le x < 0.11 \text{ のとき}) \\ 0 & (\text{ただし } x \ge 0.11 \text{ のとき}) \end{cases}$$

とする。相平面内のベクトル場と積分曲線を描け。また、初期値を

$$(x(0), y(0)) = (0.6, 0)\,,\ (0.1, 0)$$

としたときの解軌道を描け。前問の減衰振動の場合と違って、解軌道は原点に吸い込まれずに、ある閉曲線に引き寄せられていく様子が分かる。ヒント：まず位置と速度がみたす微分方程式系を手で求める（簡単）。つぎに関数 p の定義が面倒だが、定義をよく眺めると

$$p(x, y) = \mathrm{sign}(y)\max\left\{0,\ \frac{2}{5} - 8\left|x - \mathrm{sign}(y)\frac{3}{50}\right|\right\}$$

と書けることがわかる。ただし $\mathrm{sign}(y)$ は y の符号。これならすぐにプログラム可。

今日のテーマは曲面です。2 変数関数と複素関数を題材にします。

グラフィクス：　いままでも度々グラフィクスのコマンドを利用してきました。平面図形ならば `Plot[]` や `ParametricPlot[]` が代表的で、空間図形ならば `Plot3D[]` や `ParametricPlot3D[]` をよく使います。今回は、曲面上に等高線を描くコマンド `ContourPlot3D[]` を紹介します。等高線を描くと、場合によっては、空間図形の形状をよりよく把握することができます。例として関数

$$f(x,y)=\begin{cases}\dfrac{xy\left(x^2-y^2\right)}{x^2+y^2} & (x,y)\neq(0,0)\text{ のとき}\\ 0 & (x,y)=(0,0)\text{ のとき}\end{cases}$$

を考えます。このように場合分けのある関数は `Piecewise[]` で

```
f[x_,y_]:=Piecewise[{{0,x==0&&y==0}},
          x*y*(x^2-y^2)/(x^2+y^2)]
```

と定義します。関数 f は原点において偏微分の順序が交換できない、すなわち $f_{xy}(0,0)\neq f_{yx}(0,0)$ となる病的な関数ですが、グラフを `Plot3D[]` で描くと、形状はいたってふつうで、いったいどこが病的なのかわかりません。そこで等高線を描いてみます。

```
ContourPlot3D[z==f[x,y],{x,-1,1},{y,-1,1},{z,-1,1},
              MeshFunctions->{#3&},Mesh->20]
```

わかりますか。原点の周辺に所々「空白」があることに注目します。範囲を `{z,-0.01,0.01}` に変えて、さらに原点付近をマウスで拡大（`Ctrl` を押しながらマウスを動かす）してみましょう。

複素関数：　皆さんが慣れ親しんでいる関数は、実数を変数とする実数値の関数です。これに対し、複素数を変数とする複素数値の関数を複素関数といいます。たとえば、複素数 z に対して複素数 z^2 を対応させる関数を f としましょう。すなわち $f(z)=z^2$ です。このとき、複素数を実部と虚部にわけて考えると、$z=x+\sqrt{-1}y$ に対して

$$f(z)=z^2=\left(x+\sqrt{-1}y\right)^2=x^2-y^2+\sqrt{-1}\,(2xy)$$

となります。したがって

$$u(x,y)=x^2-y^2,\quad v(x,y)=2xy$$

とおくと $f(z)=u(x,y)+\sqrt{-1}v(x,y)$ と書けます。このように、複素関数をひとつ与えることは、ふたつの 2 変数関数 u,v を与えることと同等です。

– 複素関数についての理論を「関数論」といいます。関数論の主役は微分可能な複素関数であり、これを正則関数といいます。正則関数は、実部と虚部がたがいに独立ではなく、一方を与えると、他方が定数倍の差を除いて決まってしまいます。

さて、実関数の場合は、その性質を調べる際、グラフを利用すると便利であることは周知の通りです。しかし複素関数に対して同じことをしようとすると 4 次元空間が必要となり、これをそのまま可視化することは不可能です。では、まったくお手上げかというと、必ずしもそうではありません。以下では、複素関数をいかにして可視化するか、という問題を考えてみましょう。

- まずは素朴に、複素関数の実部と虚部を並べてみます。

```
f[z_]=z^2;
a=Plot3D[Re[f[x+I*y]],{x,-3,3},{y,-3,3}];
b=Plot3D[Im[f[x+I*y]],{x,-3,3},{y,-3,3}];
GraphicsRow[{a,b}]
```

さきほど計算したように実部は x^2-y^2 で虚部は $2xy$ なので、これらふたつのグラフが並んで表示されます。しかし、このふたつのグラフがなぜ「複素数 z を 2 乗すること」を表現しているのか、実感がわきません。

- つぎに複素関数をひとつのグラフで表現することを考えます。

– 複素数はその絶対値と偏角で決まります。これを復習しましょう。複素数 $z=x+\sqrt{-1}y$ に対して平面の点 ${}^t(x,y)$ を対応させます。このときの平面を複素平面と呼びます。複素平面内の点 ${}^t(x,y)$ を極座標で表示すれば $x=r\cos\theta$, $y=r\sin\theta$ と書けます。値 r を複素数 z の絶対値（あるいは大きさ）といい、記号で $|z|$ と書きます。また、値 θ を複素数 z の偏角といい、記号で $\arg(z)$ と書きます。

– この事実を利用して、複素関数をひとつのグラフで表現します。絶対値は通常どおりグラフで表示して、偏角は色で表現します。たとえば偏角をその値に応じて

```
list=Table[Style[1,Hue[h]],{h,0.5,-0.5,-0.005}];
PieChart[list,ChartBaseStyle->EdgeForm[None]]
```

と色分けすることにします。このとき

```
f[z_]=z^2;
myColor={ColorFunction->Function[{x,y,z},
         Hue[Rescale[Arg[f[x+I*y]],{0,2*Pi}]]],
         ColorFunctionScaling->False};
Plot3D[Abs[f[x+I*y]],{x,-2,2},{y,-2,2},
       Evaluate[myColor],Mesh->False]
```

のように関数 $f(z)=z^2$ をプロットすると、偏角が 2 倍のスピードで変化していることが見てとれます。

- 次に定義域と値域を別々に描き、座標系がどのように写像されるか、その変化のしかたを観察します。複素平面をふたつ用意して、ひとつを変数 z が動く平面とし、他方を関数値 $f(z)$ が動く平面とします。

– まずは直交座標系で考えます。定義域の複素平面において、実軸および虚軸に平行な直線をそれぞれ青と赤で表示し、それぞれがどのような曲線に写像されるのか、対応関係を見やすくしておきます。

```
f[z_]=z^2; x0=0.5; y0=0.5;
a=Show[ParametricPlot[{x,y},{x,-1,1},{y,-1,1}],
  ParametricPlot[{x0,y},{y,-1,1},PlotStyle->Red],
  ParametricPlot[{x,y0},{x,-1,1},PlotStyle->Blue]];
g[x_,y_]=Through[{Re,Im}[f[x+I*y]]];
b=Show[ParametricPlot[g[x,y],{x,-1,1},{y,-1,1}],
  ParametricPlot[g[x0,y],{y,-1,1},PlotStyle->Red],
  ParametricPlot[g[x,y0],{x,-1,1},PlotStyle->Blue]];
GraphicsRow[{a,b}]
```

左側の図では青と赤の交点がひとつしかないのに、なぜ右側の図では交点がふたつに増えるのですか。対応関係をよく観察すればわかります。

– つぎに極座標系で考えます。ここでも対応を見やすくするために、定義域の複素平面において、原点からの距離が一定の円を青で表示し、偏角が一定の直線を赤で表示しておきます。

```
f[z_]=z^2; r0=0.5; t0=0.5; tmin=0.05*Pi; tmax=0.95*Pi;
p[r_,t_]={r*Cos[t],r*Sin[t]};
a=Show[ParametricPlot[p[r,t],{r,0,1},{t,tmin,tmax}],
  ParametricPlot[p[r0,t],{t,tmin,tmax},PlotStyle->Red],
  ParametricPlot[p[r,t0],{r,0,1},PlotStyle->Blue]];
```

```
g[r_,t_]=Through[{Re,Im}[f[r*Cos[t]+I*r*Sin[t]]]];
b=Show[ParametricPlot[g[r,t],{r,0,1},{t,tmin,tmax}],
  ParametricPlot[g[r0,t],{t,tmin,tmax},PlotStyle->Red],
  ParametricPlot[g[r,t0],{r,0,1},PlotStyle->Blue]];
GraphicsRow[{a,b}]
```

– このように、定義域内で直交する 2 直線は、複素関数 f で写像されたあと、値域内でもやはり直交することが分かります。この性質は等角性と呼ばれ、正則関数がもっている特徴のひとつです。

実習課題

1 集合 $X = \left\{ {}^{\mathrm{t}}(x,y,z) \in \mathbb{R}^3 \;\middle|\; z = x^2 - y^2 \right\}$ がどのような図形をなすか描いてみよう。記号の便利のため $f(x,y) = x^2 - y^2$ とおく。集合

$$X_1 = \left\{ \begin{pmatrix} x \\ y \\ f(x,y) \end{pmatrix} \;\middle|\; x,y \in \mathbb{R} \right\}$$

$$X_2 = \left\{ \begin{pmatrix} u+v \\ u-v \\ f(u+v,u-v) \end{pmatrix} \;\middle|\; u,v \in \mathbb{R} \right\}$$

$$X_3 = \left\{ \begin{pmatrix} r\cos\theta \\ r\sin\theta \\ f(r\cos\theta, r\sin\theta) \end{pmatrix} \;\middle|\; r \geq 0,\ 0 \leq \theta < 2\pi \right\}$$

はいずれも X と等しく $X = X_1 = X_2 = X_3$ である。

(1) 集合 X_1, X_2, X_3 が表す図形をそれぞれ次の範囲でプロットせよ。

X_1 は $-1 \leq x \leq 1,\ -1 \leq y \leq 1$ の範囲で。
X_2 は $-1/2 \leq u \leq 1/2,\ -1/2 \leq v \leq 1/2$ の範囲で。
X_3 は $0 \leq r \leq 1,\ 0 \leq \theta < 2\pi$ の範囲で。

ヒント：`?ParametricPlot3D`

(2) 前問で描いた 3 つの図形はそれぞれ違う形をしているが、これらはいずれも図形 X の一部分を切り取ったものであり、曲面としては同一のものを表現している。実際 X_1 と X_2 を同時に表示すればぴったりと重なることがわかる。これを確かめよ。また X_1 と X_3 および X_2 と X_3 についても確認せよ。ヒント：`?Show`

2 複素数について次の問に答えよ。

(1) 複素数 z に対して、級数

$$1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots$$

は収束することが知られている。このことを視覚的に確認しよう。級数の部分和を

$$\begin{aligned} S_0 &= 1 \\ S_1 &= 1 + z \\ S_2 &= 1 + z + z^2/2! \\ S_3 &= 1 + z + z^2/2! + z^3/3! \\ &\vdots \end{aligned}$$

とおく。各 S_n は複素数だから複素平面内の点を表している。複素数 $z = 2 + \sqrt{-1}/2$ に対して、点 $S_n(z)$ を $0 \leq n \leq 10$ の範囲でプロットし、折れ線で結べ。点列 S_n がある点に収束していく様子がわかる。ヒント：`ListPlot` のオプションに `Joined->True` をつける。

(2) 前問の級数 $\sum_{n=0}^{\infty} z^n/n!$ の収束値を e^z と表す。このとき、複素数 z を $z = x + \sqrt{-1}y$ と表示しておくと、等式

$$e^z = e^x \left(\cos y + \sqrt{-1} \sin y \right)$$

が成りたつことが知られている。これをオイラーの公式という。さらに一般に、正の実数 t に対してその複素数乗 t^z を

$$t^z = e^{\log t^z} = e^{z \log t}$$

と定める。たとえば $t = 2$ とすれば、複素数 $z = x + \sqrt{-1}y$ に対して

$$2^z = 2^x \left(\cos(y \log 2) + \sqrt{-1} \sin(y \log 2) \right)$$

である。これを確かめよ。ヒント：`?ComplexExpand`

3 リーマン予想というの、知ってますか。「ゼータ関数の非自明な零点の実部は $1/2$ だろう」という予想です。リーマン予想は正しそうだ、という雰囲気を味わってみましょう。ゼータ関数 ζ は

$$\zeta(z) = -\frac{\Gamma(1-z)}{2\pi\sqrt{-1}} \int_C \frac{(-w)^{z-1}}{e^w - 1} dw$$

と定義される複素関数で（右辺の意味はわからなくて結構。この実習課題をするぶんには問題ありません。ただしもし、積分の意味、積分路 C の取りかた、およびガンマ関数 Γ の定義について知りたい場合は、適当な関数論のテキスト、たとえばアールフォルスの「複素解析」など、を参照のこと）、とくに $\Re(z) > 1$ なる複素数 z に対しては

$$\zeta(z) = 1 + \frac{1}{2^z} + \frac{1}{3^z} + \frac{1}{4^z} + \cdots$$

という表示をもちます。また、ゼータ関数の零点というのは $\zeta(z) = 0$ をみたす複素数 z のことです。負の偶数はすべてゼータ関数の零点となることが知られていて、これを自明な零点といいます。負の偶数以外の零点を非自明な零点と呼びます。非自明な零点の実部は 0 以上 1 以下であることが知られています。

(1) まずは $\Re(z) = 1/2$ であるような複素数 z に対して、ゼータ関数の値 $\zeta(z)$ がいつ 0 になるかを調べてみよう。複素数 $\zeta(z)$ がゼロになることとその絶対値 $|\zeta(z)|$ がゼロになることは同値であることに注意。そこで、実関数 f を

$$f(x,y) = \left| \zeta\left(x + \sqrt{-1}y\right) \right|$$

と定める。関数 $f(1/2, y)$ のグラフを $0 \leq y \leq 60$ の範囲で描け。たしかに零点がところどころに存在するのがわかる。ヒント：`?Zeta`

(2) つぎに $0 \leq \Re(z) \leq 1$ であるような複素数 z に対して、ゼータ関数の値 $\zeta(z)$ がいつ 0 になるかを調べてみよう。リーマン予想によれば $\Re(z) = 1/2$ のときに限ってゼータ関数の値が 0 になりうるはずである。そこで、値 $x_0 \in [0,1]$ をいくつか固定したうえで関数 $f(x_0, y)$ のグラフを $0 \leq y \leq 60$ の範囲で描き、それらをつなげてアニメーション表示させてみよう。ヒント：基本的には `Animate[Plot[f[x,y],{y,0,60}],{x,0,1}]` とすればアニメーションが作れるが、各コマの描画範囲をそろえて見やすくするには `Plot` のオプションとして `PlotRange` を適当に指定する必要あり。

(3) ゼータ関数のグラフを描き、リーマン予想の主張を確認せよ。ただしゼータ関数の絶対値を高さとするグラフを考え、偏角に応じてグラフを色分けすること。

ヒント①：問 (1) で定義した関数 f のグラフを描けばよい。偏角による色分けの仕方は実習資料で説明済み（`myColor`）。

ヒント②：描画範囲が広いと大変なので、範囲を適当に分割する。たとえば、関数 `f[x,y]` のグラフを `{x,0,1},{y,10*n,10*(n+1)}` の範囲で描く関数を `g[n_]` とする。ただし `Plot3D` のオプションとして `PlotPoints` の値を大きめに指定しないとリーマン予想が見えない。たとえば `PlotPoints->{50,500}` など。より見やすくするために、オプションとして `PlotRange->{0,1}` と `BoxRatios->{1,5,1}` も追加。

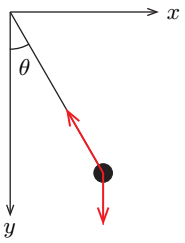
ヒント③：`g[1]`, `g[10]`, `g[-100]`, `g[12345]` などいくつか好きな数値を代入してグラフを鑑賞する。零点の実部がいつでも $1/2$ になっていそうなことが確認できる。

今日のテーマはアニメーションです。振り子の運動を表す微分方程式

$$\theta'' = -\omega^2 \sin\theta \tag{1}$$

を題材にします。

振り子の運動方程式：　長さ l の軽い棒に質量 m のおもりをつけ、天井から吊します。図のように座標軸および角 θ をとります。



重力加速度 $g = 9.80665$ を用いて $\omega = \sqrt{g/l}$ とおくと、おもりの運動は微分方程式 (1) で表現できます。以下このことを説明します。

– ニュートンの運動方程式で考える方法。まず、おもりの位置は

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = l \begin{pmatrix} \sin\theta(t) \\ \cos\theta(t) \end{pmatrix}$$

です。以後面倒なので変数 t を書くのは省略します。微分して

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = l\theta' \begin{pmatrix} \cos\theta \\ -\sin\theta \end{pmatrix}$$

です。さらに微分して、おもりの加速度 a は

$$a = \begin{pmatrix} x'' \\ y'' \end{pmatrix} = l\theta'' \begin{pmatrix} \cos\theta \\ -\sin\theta \end{pmatrix} - l\left(\theta'\right)^2 \begin{pmatrix} \sin\theta \\ \cos\theta \end{pmatrix}$$

です。一方、おもりには、鉛直下向きに大きさ mg の力と、棒の方向に棒からの張力がかかっています。張力の大きさを T としましょう。式で書けば、おもりに働く力 F は

$$F = mg \begin{pmatrix} 0 \\ 1 \end{pmatrix} + T \begin{pmatrix} -\sin\theta \\ -\cos\theta \end{pmatrix}$$

です。これらをニュートンの運動方程式 $ma = F$ に代入して

$$ml\left(\theta'' \begin{pmatrix} \cos\theta \\ -\sin\theta \end{pmatrix} - \left(\theta'\right)^2 \begin{pmatrix} \sin\theta \\ \cos\theta \end{pmatrix}\right) = mg \begin{pmatrix} 0 \\ 1 \end{pmatrix} - T \begin{pmatrix} \sin\theta \\ \cos\theta \end{pmatrix}$$

です。両辺に $(\cos\theta, -\sin\theta)$ を内積して T を消去したものが、振り子の微分方程式 (1) です。

– 大学の物理の講義でラグランジュの運動方程式を習った人は次のように考えましょう。ラグランジアン L は

$$\begin{aligned} L &= \text{「運動エネルギー」} - \text{「位置エネルギー」} \\ &= \frac{1}{2} m \left(l\theta'\right)^2 - mg\left(-l\cos\theta\right) \end{aligned}$$

です。したがってラグランジュの運動方程式

$$\frac{d}{dt}\frac{\partial L}{\partial \theta'} = \frac{\partial L}{\partial \theta}$$

に代入すれば、ただちに振り子の微分方程式 (1) を得ます。ニュートンの運動方程式では張力が出てきたのに、なぜラグランジュの運動方程式では張力が出てこないのだろう、と疑問をもった人がいるかもしれません。これは次のような事情です。いま説明した方法では、動径方向の拘束条件を予め解いてしまったために張力が登場していないのです。張力を求めるためには、極座標 (r, θ) を導入し、動径方向の仮想的な運動を考える必要があります。すなわち、拘束条件 $r - l = 0$ のもと「ラグランジュの未定乗数法」の考え方にしたがってラグランジアン

$$L = \text{「運動エネルギー」} - \text{「位置エネルギー」} - \lambda\left(r - l\right)$$

を導入し、ラグランジュの運動方程式

$$\frac{d}{dt}\frac{\partial L}{\partial \theta'} = \frac{\partial L}{\partial \theta}, \quad \frac{d}{dt}\frac{\partial L}{\partial r'} = \frac{\partial L}{\partial r}$$

を計算します。実際に計算するとわかりますが、ラグランジュ乗数 λ がちょうど張力の大きさを表しています。

振り子の周期：　アニメーションをきれいに作る（始まりのコマと終わりのコマをぴったり重ねる）ために、振り子の周期 P について計算しておきます。最大角を θ_0 とすると、エネルギー保存の法則から

$$\frac{1}{2} m \left(l\theta'\right)^2 + mg\left(-l\cos\theta\right) = mg\left(-l\cos\theta_0\right)$$

が成りたちます。すなわち

$$\left(\frac{d\theta}{dt}\right)^2 = 2\omega^2\left(\cos\theta - \cos\theta_0\right)$$

です。振り子が $\theta = \theta_0$ から $\theta = 0$ まで振れるのに要する時間は $P/4$ なので、この区間で積分すると

$$\int_{\theta_0}^{0} \frac{1}{\sqrt{\cos\theta - \cos\theta_0}} d\theta = -\int_0^{P/4} \sqrt{2}\,\omega dt = -\sqrt{2}\,\omega \frac{P}{4}$$

です。以上より、周期は

$$P = \frac{4}{\sqrt{2}\,\omega} \int_0^{\theta_0} \frac{1}{\sqrt{\cos\theta - \cos\theta_0}} d\theta$$

となります。もうすこし右辺の定積分を計算するために、新しい変数 ϕ を

$$\sin\frac{\theta}{2} = \sin\frac{\theta_0}{2}\sin\phi$$

と導入します。この変数変換によって、周期は

$$P = \frac{4}{\omega} \int_0^{\pi/2} \frac{1}{\sqrt{1 - \sin^2\left(\theta_0/2\right)\sin^2\phi}} d\phi$$

と表示できます。右辺は、第一種完全楕円積分と呼ばれる定積分

$$K\left(k\right) = \int_0^{\pi/2} \frac{1}{\sqrt{1 - k^2\sin^2\phi}} d\phi \tag{2}$$

を用いて

$$P = \frac{4}{\omega} K\left(\sin\frac{\theta_0}{2}\right) \tag{3}$$

と書けます。実際に第一種完全楕円積分の値を計算するには Mathematica のコマンド `EllipticK[]` を使います。ただしひとつ注意。式 (2) の右辺の定積分の値は、数学ではふつう $K\left(k\right)$ と書くことが多いのですが、Mathematica では `EllipticK[k^2]` と書きます。これを `EllipticK[k]` と書かないように注意します。

アニメーション：　微分方程式 (1) を数値的に解いて、振り子のアニメーションを作ってみます。いま計算したように、振り子の周期 P (1 往復するのに何秒かかるか) は棒の長さ l と角 θ_0 の値で決まります。角 θ_0 はたとえば $\pi/6$ としましょう。棒の長さ l は、60 秒間に振り子がちょうど整数回（たとえば 20 回）往復するように決めておくことにします。つまり P は $20P = 60$ をみたすものとし、これを式 (3) を代入して l の値を決めます。プログラムはたとえば次のように書きます。

```
(*振り子が 1 分間に n 往復するような棒の長さを求める*)
n=20;
tMax=60;
initAngle=Pi/6;
g=9.80665;
eq=4*Sqrt[x/g]*EllipticK[(Sin[initAngle/2])^2]==tMax/n;
length=Solve[eq,x];
l=x/.length[[1]];
(*運動方程式を数値的に解く*)
ode=theta''[t]==-(g/l)*Sin[theta[t]];
init={theta[0]==initAngle,theta'[0]==0};
ivp=Flatten[{ode,init}];
sol=NDSolve[ivp,theta,{t,0,tMax}];
(*2 次元アニメーション*)
position={l*Sin[theta[t]],-l*Cos[theta[t]]}/.sol[[1]];
bob=Disk[position,0.05];
rod=Line[{{0,0},position}];
l2=1.1*l;
Animate[Graphics[{Black,bob,Gray,rod}/.t->time,
 PlotRange->{{-l2,l2},{-l2,l2}}],{time,0,tMax},
 DefaultDuration->tMax,SaveDefinitions->True]
```

オプション SaveDefinitions->True の意味を知るには、このオプションを False に変更してみてください。ファイルを保存してからいったん終了し、再度ファイルを開いてアニメーションを実行するとどうなりますか。

次に振り子の数を増やします。15 個の振り子が動く様子をひとつの画面に同時に描いてみます。上記プログラムをすこしだけ変更すればできます。

```
(*k 個の振り子が 1 分間にそれぞれ n,n+1,...,n+k-1 往復するような棒の長さを求める*)
k=15;
n=20;
tMax=60;
initAngle=Pi/6;
g=9.80665;
eq=Table[4*Sqrt[x/g]*EllipticK[(Sin[initAngle/2])^2]
         ==tMax/(n+i-1),{i,1,k}];
length=Table[Solve[eq[[i]],x],{i,1,k}];
l=Table[x/.length[[i,1]],{i,1,k}];
(*k 個の独立な運動方程式を数値的に解く*)
ode=Table[theta''[t]==-(g/l[[i]])*Sin[theta[t]],{i,1,k}];
init={theta[0]==initAngle,theta'[0]==0};
ivp=Table[Flatten[{ode[[i]],init}],{i,1,k}];
sol=Table[NDSolve[ivp[[i]],theta,{t,0,tMax}],{i,1,k}];
(*2 次元アニメーション*)
position=Table[{l[[i]]*Sin[theta[t]],
             -l[[i]]*Cos[theta[t]]}/.sol[[i,1]],{i,1,k}];
bob=Table[Disk[position[[i]],0.05],{i,1,k}];
rod=Table[Line[{{0,0},position[[i]]}],{i,1,k}];
l2=1.1*l[[1]];
Animate[Graphics[{Black,bob,Gray,rod}/.t->time,
 PlotRange->{{-l2,l2},{-l2,l2}}],{time,0,tMax},
 DefaultDuration->tMax,SaveDefinitions->True]
```

15 個の振り子が互いにぶつからないように、きちんと 3 次元空間の中に配置してみましょう。次のコードを末尾に追加します。

```
(*3 次元アニメーション*)
position3D=Table[{i*0.2,l[[i]]*Sin[theta[t]],
             -l[[i]]*Cos[theta[t]]}/.sol[[i,1]],{i,1,k}];
bob3D=Table[Sphere[position3D[[i]],0.05],{i,1,k}];
rod3D=Table[Line[{{position3D[[i,1]],0,0},
                  position3D[[i]]}],{i,1,k}];
Animate[Graphics3D[{Black,bob3D,Gray,rod3D}/.t->time,
PlotRange->{{0,(k+1)*0.2},{-l2,l2},{-l2,l2}}],
{time,0,tMax},DefaultDuration->tMax,SaveDefinitions->True]
```

.....

実習課題

振り子の運動をあらわす微分方程式 (1) の親戚に、サイン・ゴルドン方程式とよばれる微分方程式がある。サイン・ゴルドン方程式は 2 変数関数 $\theta = \theta(x,t)$ についての偏微分方程式

$$\left(\frac{\partial^2}{\partial t^2} - \frac{\partial^2}{\partial x^2}\right)\theta = -\sin\theta$$

であり、その解析解としてたとえば次のものが知られている。

$$\theta_1(x,t) = 4\arctan\left(\frac{c}{\sqrt{c^2-1}}\frac{\sinh\left(\sqrt{c^2-1}\,t\right)}{\cosh\left(c\,x\right)}\right)$$

$$\theta_2(x,t) = 4\arctan\left(\frac{\sqrt{c^2-1}}{c}\frac{\sinh\left(c\,x\right)}{\cosh\left(\sqrt{c^2-1}\,t\right)}\right)$$

$$\theta_3(x,t) = 4\arctan\left(\frac{q}{\sqrt{1-q^2}}\frac{\sin\left(\sqrt{1-q^2}\,t\right)}{\cosh\left(q\,x\right)}\right)$$

ただし c, q はパラメータである。以下の問いに答えよ。
(1) 関数 θ_1, θ_2, θ_3 はいずれもサイン・ゴルドン方程式の解であることを示せ。ヒント：?FullSimplify
以下ではパラメータの値を $c = 2$, $q = 1/2$ とする。
(2) 関数 θ_1 のグラフをアニメーションで表示せよ。ただしアニメーションのパラメータを t ($-10 \le t \le 10$) とし、各 t の値について関数 $\theta_1(x,t)$ のグラフを $-10 \le x \le 10$ の範囲で描くこと。ヒント：アニメーションを見やすくするためには Plot のオプションとして PlotRange を適当に指定する必要あり。
(3) 関数 θ_2 と θ_3 についても同様にアニメーション表示せよ。
(4) サイン・ゴルドン方程式にしたがって動くような振り子のアニメーションを作ろう。3 次元空間の中に 51 個の振り子を一列にならべる。ただし、時刻 t における第 i 番目 ($i = 1, \ldots, 51$) の振り子のおもりの位置 $p_i(t)$ を、サイン・ゴルドン方程式の解 $\theta\,(x,t)$ を用いて

$$p_i(t) = \frac{1}{2}\begin{pmatrix} -5 + (i-1)/5 \\ \sin\theta\,(x_i,t) \\ -\cos\theta\,(x_i,t) \end{pmatrix}, \quad x_i = -13 + \frac{i}{2}$$

と定める。関数 θ_1, θ_2, θ_3 のそれぞれについて $-10 \le t \le 10$ の範囲で振り子のアニメーションを作れ。ヒント：プログラムの概略は次のとおり。

```
k=51;
x[i_]=-13+i/2;
p[theta_,i_,t_]:=1/2*{略,Sin[theta[x[i],t]],略};
bob[theta_,i_,t_]:=Sphere[p[theta,i,t],0.05];
rod[theta_,i_,t_]:=Line[略];
animation[theta_]:=Animate[略];
```

このとき、たとえば関数 θ_1 に theta1 と名前をつけている場合は、animation[theta1] を評価すると解 θ_1 による振り子のアニメーションが表示される。

今日のテーマは手続き型プログラミングです。

手続き型プログラミング： Mathematica では C 言語のような通常の手続き型プログラミングを行うことができます。そのとき使うのが `Module[]` という命令です。これを使うことのメリットは「いくつかの処理をまとめてひとつの関数としてパッケージ化できる」点にあります。例を挙げましょう。たとえば、実数 a, b, c が与えられたとき「2 次方程式 $ax^2 + bx + c = 0$ の解 x_1, x_2 を複素平面上にプロットする」ことを考えます。この作業をするには、まず 2 次方程式を解いて解 x_1, x_2 を求め、そのうえで平面の点 $(\Re x_1, \Im x_1)$ および $(\Re x_2, \Im x_2)$ をプロットする、という手順を踏む必要があります。この 2 つの手順をひとつの関数としてまとめてしまいましょう。次のように書きます。

```
plotSol[a_,b_,c_]:=Module[{sol,p1,p2},
  sol=Solve[a*x^2+b*x+c==0,x];
  p1={Re[x],Im[x]}/.sol[[1]];
  p2={Re[x],Im[x]}/.sol[[2]];
  ListPlot[{p1,p2}]]
```

書式は

```
Module[{局所変数のリスト},
  処理 1;
  処理 2;
  処理 3]
```

です。ここでもうひとつ新しいことを覚えます。アンダースコアの使い方です。アンダースコアを 3 つ並べると、これは「複数あってもよいし、まったくなくてもよい」という状況を想定したパターンの記述方法になります。たとえば

```
plotSol[a_,b_,c_,option___]:=Module[{sol,p1,p2},
  sol=Solve[a*x^2+b*x+c==0,x];
  p1={Re[x],Im[x]}/.sol[[1]];
  p2={Re[x],Im[x]}/.sol[[2]];
  ListPlot[{p1,p2},option]]
```

と書くことで、プロットのオプションが楽に指定できるようになります。すなわち `plotSol[1,2,3]` と書けば、点はデフォルトの設定でプロットされ、`plotSol[1,2,3,PlotStyle->Red]` と書けば赤丸で、`plotSol[1,2,3,PlotStyle->Red,PlotMarkers->{Automatic,10}]` と書けば大きな赤丸でプロットされます。アンダースコアの詳細については `?Blank`（アンダースコア 1 つ）や `?BlankSequence`（アンダースコア 2 つ）や `?BlankNullSequence`（アンダースコア 3 つ）。

また、複雑な処理をするときのテクニックとして基本的なのは、条件分岐と反復処理です。プログラミング I（C 言語）を履修しているひとにとってはお馴染みの内容でしょう。

- 条件分岐を行うには `If[]` を使います。書式は `If[cond,t,f]` です。`cond` の評価の結果が `True` ならば `t` を評価し、`False` ならば `f` を評価します。たとえば

  ```
  abs[x_]:=If[x<0,-x,x]
  ```

  のように使います。
- 反復処理を行うには `For[]` または `While[]` を使います。反復処理の基本構造には 2 つの種類があります。ひとつは、繰り返しの実行回数を定めておくものであり、これには `For[]` を用います。もうひとつは、繰り返しのたびに条件判断を行って続行や打ち切りを決定するものであり、これには `While[]` を用います。それぞれについて説明します。
  - コマンド `For[]` の書式は `For[start,test,incr,body]` です。これは、まず `start` を実行し `test` が `True` を返さなくなるまで `body` と `incr` を繰り返し評価します。評価の順番は `test`, `body`, `incr` の順に行われます。たとえば

    ```
    fact[n_]:=Module[{x=1,i},
      For[i=1,i<=n,i++,x=i*x];
      Return[x]]
    ```

    のように使います。また、このように、`Module` 内部で用いる局所変数は、宣言と同時に初期化することができます。
  - コマンド `While[]` の書式は `While[test,body]` です。これは `test` が `True` を返さなくなるまで `test` と `body` を繰り返し評価します。たとえば `n` 以下の自然数値を当てさせるゲームを作りましょう。正解するまで入力を要求します。いつ正解するのか分からないので `For[]` ではなく `While[]` を使います。

    ```
    numbers[n_]:=Module[{atari=RandomInteger[{1,n}],m},
      m=Input[];
      While[m!=atari,
        If[m>atari,
          Print["もっと小さい数"],
          Print["もっと大きい数"]];
        m=Input[];];
      Print["あたり"]]
    ```

実習課題：

1 自然数 m, n に対して、方程式

$$am + bn = \gcd(m, n) \tag{1}$$

をみたす整数 a, b の組が無数に存在することが知られている。ただし $\gcd(m, n)$ は m, n の最大公約数である。式 (1) をベズーの恒等式という。入力 m, n に対して、式 (1) の解 a, b を一組見つけたうえでベクトル $(a, b, \gcd(m, n))$ を返すような関数 `bezout` を作れ。

ヒント①：ベズーの恒等式をみたす整数 a, b を具体的に一組求めるためには「ユークリッド互除法」が有効です。ユークリッド互除法のアルゴリズムのひとつに次のものがあります。

- 開始：ベクトル $\boldsymbol{u} = (u_1, u_2, u_3)$, $\boldsymbol{v} = (v_1, v_2, v_3)$, $\boldsymbol{t} = (t_1, t_2, t_3)$ および変数 q を用意する。ただしベクトル $\boldsymbol{u}, \boldsymbol{v}$ の初期値は $\boldsymbol{u} = (1, 0, m)$, $\boldsymbol{v} = (0, 1, n)$ とする。
- ループ：v_3 が 0 でない限りつぎの操作を続ける。
  1. u_3 を v_3 で割ったときの商を q とおく。
  2. $\boldsymbol{u} - q\boldsymbol{v}$ を $\boldsymbol{t}$ とおく。
  3. $\boldsymbol{v}$ を改めて $\boldsymbol{u}$ とおく。
  4. $\boldsymbol{t}$ を改めて $\boldsymbol{v}$ とおく。
- 終了：ベクトル $\boldsymbol{u}$ を返す。このとき $u_3 = \gcd(m, n)$ でしかも $u_1 m + u_2 n = u_3$ が成り立つ。

ヒント②：このアルゴリズムを素直にプログラミングすればよいでしょう。そのためには `Module` を使うのが自然です。たとえば

```
bezout[m_,n_]:=Module[{u={1,0,m},v={0,1,n},t,q},
  省略;
  Return[u]]
```

などと書きます。ヒント③：`?Quotient`

2「6 以上の全ての偶数は、ふたつの奇素数の和で表すことができるだろう」という予想（ゴールドバッハ予想）があります。奇素数というのは 3 以上の素数のことです。実際にいくつかの偶数についてゴールドバッハ予想を確かめてみましょう。以下では n と書けばそれは 6 以上の偶数であるとします。

(1) $n = p_1 + p_2$ となる奇素数 p_1, p_2 のすべての組み合わせを出力するような関数 `goldbach1` を作れ。ただし、たとえば `goldbach1[26]` を評価したら

```
26=3+23
26=7+19
26=13+13
```

と表示するようにせよ。ヒント①：関数の定義には `Module` を使い、その内部では 3 以上 $n/2$ 以下のすべての素数 p について順に $n-p$ が素数かどうかを判定し、もし素数だったら p と $n-p$ を `Print` 文で出力するようにする。ヒント②：`?PrimeQ`, `?Print`, `?Prime`

(2) $n = p_1 + p_2$ となる奇素数 p_1, p_2 の組み合わせの総数を a_n とする。入力 n に対して a_n の値を返すような関数 `goldbach2` を作れ。たとえば `goldbach2[26]` の値は `3` である。ヒント：ほぼ (1) と同じ。反復処理が何回行われたかを数えるためのカウンタを局所変数として用意するだけ。

(3) 前問 (1), (2) の関数を統合して関数 `goldbach3` を作れ。ただし `goldbach3[26]` を評価したら

```
26=3+23
26=7+19
26=13+13
```

を表示し、`goldbach3[26,length]` を評価したら `3` を返すようにすること。ヒント①：アンダースコア 3 つ。ヒント②：`?UnsameQ`

(4) 前問 (3) の関数を改良して、次のような動作をする関数 `goldbach` を作れ。`goldbach[26]` を評価したら

```
26=3+23
26=7+19
26=13+13
```

を表示し、`goldbach[26,length]` を評価したら `3` を返し、さらに `goldbach[26,graph]` を評価したら平面内に点列 $(6, a_6)$, $(8, a_8)$, ..., $(24, a_{24})$, $(26, a_{26})$ をプロットする。関数が定義できたら、いくつかの m に対して（たとえば $m = 1000, 2000, 3000, 4000$ など）グラフ `goldbach[m,graph]` を描いてみよ。ゴールドバッハ予想がたしかに正しそうであることが分かる。

.....

今後の予定：来週 6/26 は休講です。その後 7/3 と 7/10 は TEX について実習します。最終週 7/17 は休講。

今週と来週は TEX について実習します。

TEX： TEX（テフ）は組版（くみはん）を行うためのソフトウェアです。最初の TEX が使われてから三十数年がたちました。これだけ長い間安定して使われているソフトはほかに例を見ません。数式がきれいに出力できたり、目次や文献表や索引の作成が楽に行えるため、数学者のあいだでは論文執筆のときの標準的なツールとなっています。皆さんにとっては、セミナーのレジュメを作ったり、卒業研究発表会の予稿を書いたり、塾のアルバイトや教育実習で数学の試験問題をつくったり、あるいは大学院で修士論文を書いたりするときに役立つでしょう。Microsoft WORD に代表されるワープロとの違いはまず第一に、ウィジウィグか否か、ということにあります。ウィジウィグ (WYSIWYG) とは What You See Is What You Get の略で、画面表示と印刷イメージが同じことをいいます。ワープロでは入力したものがそのまま印刷されます（すなわち WYSIWYG です）が、TEX では、以下に見るように、入力原稿と印刷出力には差異があります。

ウェブ上で TEX を試す： TEX を使うにはパソコンに TEX のシステムをインストールする必要がありますが、試しに使ってみるだけなら、インストールせずにウェブ上で体験することができます。奥村晴彦さんのページ `http://oku.edu.mie-u.ac.jp/~okumura/texonweb/` を開いてください。『ここに何か書いてください。』という行を消して、好きな文章を書き込みます。それ以外の行は消さずにそのまま残しておきます。たとえば

```
方程式$x^2 = 2$の解は$x = \pm \sqrt{2}$である。
```

と書いてみます。「PDF 生成」ボタンを押せば PDF ファイルができます。出力は

方程式 $x^2 = 2$ の解は $x = \pm\sqrt{2}$ である。

となります。このように、TEX では、入力した文字列がそのまま出力されるのではなく、コマンドに相当する部分をコンパイルしてから、出力のファイルを生成します。入力と出力が “ずれている” ので、これが我慢できない人は TEX が嫌いになってしまうようです。もうひとつ、ワープロと違うところを見ておきましょう。改行の扱いです。

- 先ほどの原稿に改行を入れて

  ```
  方程式$x^2 = 2$の解は
  $x = \pm \sqrt{2}$である。
  ```

  と入力してみます。出力はどうなりますか。
- 今度は空行を入れて

  ```
  方程式$x^2 = 2$の解は

  $x = \pm \sqrt{2}$である。
  ```

  と入力してみます。出力はどう変わりますか。
- 空行が 2 行以上（3 行でも 4 行でも）続くとどうでしょうか。

  ```
  方程式$x^2 = 2$の解は


  $x = \pm \sqrt{2}$である。
  ```

つまり TEX では原稿の改行は無視され、空行が 1 行以上連続する場合には段落がひとつだけ改まります。文章を入力するときには TEX のこの流儀を生かして、好きなところで改行を入れながら、文章を書きます。一行の長さを揃える必要はまったくありません。キーボードを打っているときのリズムや気分に応じて、あるいは原稿を見やすくするための工夫として、気ままに改行を入れてください。

数式モード： もうすこしウェブ上で TEX を試してみましょう。数式の入力について基本的なことを練習します。すでに見たように、数式を入力するときは数式を `$` マークで囲みます。この `$` マークで囲まれた領域を数式モードといいます。数式モードで使えるコマンドのうち、代表的なものをいくつか挙げます。たとえば

`\pm`	$\pm$	`\times`	$\times$	`\div`	$\div$
`\leq`	$\leq$	`\geq`	$\geq$	`\neq`	$\neq$
`\in`	$\in$	`\notin`	$\notin$	`\subset`	$\subset$
`\cap`	$\cap$	`\cup`	$\cup$	`\emptyset`	$\emptyset$
`\angle`	$\angle$	`\perp`	$\perp$	`\parallel`	$\parallel$
`\to`	$\to$	`\mapsto`	$\mapsto$	`\infty`	∞
`\sqrt{x}`	$\sqrt{x}$	`x^{10}`	x^{10}	`x_{10}`	x_{10}
`\partial`	∂	`\cos x`	$\cos x$	`\sin x`	$\sin x$

や

`\frac{a}{b}`	$\frac{a}{b}$
`\{1, 2, \dots, n\}`	$\{1, 2, \dots, n\}$
`\sum_{k = 0}^n`	$\sum_{k=0}^{n}$
`\lim_{x \to 0}`	$\lim_{x \to 0}$
`\int_0^1`	$\int_0^1$
`\langle x, y\rangle`	$\langle x, y\rangle$

などです。TEX では記号 `{` と記号 `}` には特別な役割があるので、これらの記号そのものを出力するには `\{` と `\}` のように書かなければなりません。また、数学ではギリシア文字もよく使いますが、ギリシア文字の小文字は次のように書きます。

`\alpha`	α	`\beta`	β	`\gamma`	γ	`\delta`	δ
`\epsilon`	ϵ	`\zeta`	ζ	`\eta`	η	`\theta`	θ
`\iota`	ι	`\kappa`	κ	`\lambda`	λ	`\mu`	μ
`\nu`	ν	`\xi`	ξ	`\pi`	π	`\rho`	ρ
`\sigma`	σ	`\tau`	τ	`\upsilon`	υ	`\phi`	ϕ
`\chi`	χ	`\psi`	ψ	`\omega`	ω		

たとえば

- $e^{\sqrt{-1}\theta} = \cos\theta + \sqrt{-1}\sin\theta$
- $\{1, 2\} \cap \{3, 4\} = \emptyset$
- $\sum_{k=0}^{10} a_k = a_0 + a_1 + \cdots + a_{10}$
- $\frac{d}{dx}\sin x = \cos x$

などの数式を出力するには、それぞれどのように入力しますか。

TEX の実際： それでは本式に TEX を使ってみます。とりあえず Windows 上で説明しますが、Linux が好きな人は Linux を使ってください。まず TEX 用のフォルダを作っておきましょう。フォルダの場所と名前はたとえば `H:¥tex` とします。このフォルダは Linux 上では `~/windows/tex/` で参照できます。TEX を扱うときの基本的な手順は

1. 適当なエディタで TEX のソースを書く。
2. コマンドプロンプト上でコンパイルする。
3. 生成された DVI ファイルを見て、仕上がりをチェックする。
4. もし必要なら DVI ファイルを PDF ファイルに変換する。

という順番です。先ほど試したウェブ上の TEX サービスは、手順 2 と手順 3 が省略されており（というより、初心者が意識しなくて済むように裏で処理されており）、表に見えるのは、手順 1 と手順 4 のふたつだけでした。それぞれの手順について、もうすこし詳しく見てみましょう。

1. 適当なエディタで TEX のソースを書く。エディタは好みのもの、たとえば Meadow（Windows 上の Emacs 風エディタのひとつ）など、を使います。次のファイルが TEX のテンプレートです。このファイルの内容を `template.tex` として保存します。

```
\documentclass[a4paper,11pt]{jsarticle}
\usepackage[top=25truemm,bottom=25truemm,left=25truemm,right=25truemm]{geometry} % マージンの指定
\usepackage{amsmath,amssymb} % 数学
\usepackage[dvipdfmx]{graphicx} % グラフィクス
\begin{document}
本文。
\end{document}
```

– 1 行目は、文書のクラスを指定しています。この宣言の意味は「用紙は A4 版を用い、文字の大きさは欧文文字サイズが 11 ポイントとなるように印字するが、文書のレイアウトはすべてクラスファイル `jsarticle` に任せる」ということです。ポイント `pt` は印刷業界でよく使われる長さの単位で、72.27 ポイント = 1 インチ = 2.54 センチメートル、すなわち 1 ポイント = 0.3514 ミリメートル です。
– 2 行目は、用紙の上下左右のマージン（余白）を好みの大きさに設定しています。長さの単位の `truemm` は「本当のミリメートル」の意味です。単に `mm` と書くと、さきに指定した文字の大きさに応じて単位長が変わってしまうので、ポイント値に依存しない本当の長さを指定します。また、コメントを書くときは、コメント文の先頭に記号 `%` をつけます。もしコメント文が複数行にわたるなら各行ごとに `%` が必要です。
– 3 行目は、数学のパッケージを読み込んでいます。使えるコマンドやフォントが増え、便利になります。
– 4 行目は、グラフィクス用のパッケージを読み込んでいます。文書中に画像を挿入するにはこのパッケージが必要です。
– ここで言葉をひとつ覚えましょう。1 行目の `\documentclass[a4paper,11pt]{jsarticle}` と 5 行目の `\begin{document}` で挟まれた領域（つまり 2 行目から 4 行目まで）をプリアンブルと呼びます。プリアンブルは前置きとか序文というような意味で、文書についてのいろいろな設定を書き込む場所です。

● それでは実際に TEX の原稿を書いてみます。いまから書く原稿のファイル名を `hellotex.tex` とします。TEX の原稿を書くときは、まず `templatex.tex` の内容をコピーすることから始めます。
– エディタが Meadow なら、最初に「`Ctrl-x Ctrl-f`」でファイル名 `H:/tex/hellotex.tex` を決めて、そのあと「`Ctrl-x i`」でテンプレートのファイル `H:/tex/templatex.tex` を挿入します。ファイルを編集したあと保存するには「`Ctrl-x Ctrl-s`」です。キーボードからのコマンド操作が嫌な人は、ウィンドウ上部に表示されているアイコンをマウスでクリックしても構いません。

● 「本文。」という行を消して、好きな文章を書き込みます。それ以外の行は消さずにそのまま残しておきます。適当な文章が思いつかない人は、たとえば「方程式`$x^2 = 2$`の解は`$x = \pm \sqrt{2}$`である。」と書いてみましょう。

2. コマンドプロンプト上でコンパイルする。いま作った TEX のソースファイル（`hellotex.tex`）を DVI という形式のバイナリファイルに変換します。この変換作業をコンパイル（またはタイプセット）と呼びます。コマンドプロンプトを開き（スタート→すべてのプログラム→アクセサリ→コマンドプロンプト）、目的の場所に移動（`cd H:¥tex`）してから、次のように入力します。

```
platex hellotex.tex
```

エラーが出る場合もあります。もしエラーが出たら `hellotex.tex` の内容を適当に修正し、そのあとでもう一度コンパイルします。
– Windows と Linux を併用する人のための注意。両 OS 間でファイルを共有するときは文字コードに注意が必要です。Windows 上でつくった TEX ファイルの文字コードが SJIS の場合、そのファイルを Linux 上でふたたびコンパイルしたいならば、Linux 上では

```
platex -kanji=sjis hellotex.tex
```

のようにオプションをつける必要があります。もしこのオプションをつけないとコンパイルが正常に行われないため、DVI ファイルにおいて日本語フォントがまったく表示されない状態になります。なお、最近の主流では TEX ファイルは UTF-8 で書くことが推奨されています。Windows で作るファイルをはじめから UTF-8 にしておけば、このような文字コード指定の面倒は起こりません。ファイルをはじめから UTF-8 で作るにはエディタ Meadow をカスタマイズする必要がありますが、その方法（簡単です）については割愛します。Meadow のカスタマイズ方法を知りたい人は `.emacs`（ドットイーマックス）をキーワードに検索してください。

3. 生成された DVI ファイルを見て、仕上がりをチェックする。手順 2 のコンパイルが正常終了したならば、同じディレクトリに DVI ファイルが生成されているはずです。そのファイルをビューア（dviout.exe）で開いて仕上がりをチェックします。
– マウス操作で簡単にファイルが開けるように、はじめに一度だけファイルの関連づけをしておきましょう。DVI ファイル `hellotex.dvi` のアイコンを右クリックし、「プロパティ」→プログラムの「変更」→「参照」→ローカルディスク (C:) → Software → dviout → dviout.exe →「開く」を選びます。今後は DVI ファイルのアイコンをクリックすれば自動的にビューア dviout が立ち上がります。
– Linux ならばビューアは `xdvi` を使います。

4. もし必要なら DVI ファイルを PDF ファイルに変換する。コマンドプロンプト上で

```
dvipdfmx hellotex.dvi
```

とします。これにより `hellotex.pdf` が生成されます。PDF ファイルのアイコンをクリックすればビューア Acrobat Reader が自動的に立ち上がります。
– Linux ならばビューアは `acroread` です。

TEX について、いくつかのトピックを取り上げて説明します。

画像の挿入：　文書中に関数のグラフを挿入したいとします。このような場合、グラフを直接 TEX で描くことはせず、グラフの描画はそれ専用のソフトウェアに任せます。たとえば Mathematica でグラフを描き、グラフの画像を PDF ファイルに書き出してから、その PDF ファイルを TEX 文書中に挿入する、という手順を踏みます。書き出した PDF ファイルの名前が `graph.pdf` ならば

```
\begin{center}
\indludegraphics[width=30mm]{graph.pdf}
\end{center}
```

と書くと、画像が幅 30mm にスケール（拡大または縮小）されて、左右中央に出力されます。幅 `width` のかわりに高さ `height` の値を指定することもできます。もし幅と高さの値を同時に指定した場合には、画像の縦横比が変わります。縦横比を保ちたいときは、オプションを

```
[width=30mm,height=40mm,keepaspectratio]
```

と指定すると、縦横比を変えずに、指定した幅と高さに収まるように画像がスケールされます。

別行立ての数式：　数式を書くときは、数式部分を `$` マークで囲まなければいけないことをすでに説明しました。たとえば

```
方程式$a x^2 + b x + c = 0$の解は
$x = \frac{- b \pm \sqrt{b^2 - 4 a c}}{2 a}$である。
```

と書けば、出力は

方程式 $ax^2 + bx + c = 0$ の解は $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ である。

となります。ところが数式中に分数が登場する場合には、このように分数の部分だけが小さなフォントになってしまうため、出力が見苦しくなります。本来ならば、分数は別の行に大きく書いて

```
方程式 $ax^2 + bx + c = 0$ の解は
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
である。
```

方程式 $ax^2 + bx + c = 0$ の解は

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

である。

と出力すべきでしょう。このように別の行に大きく書かれた数式を、別行立ての数式（あるいはディスプレイスタイルの数式）といいます。ディスプレイスタイルの数式を書くには、数式を `$` で囲むのではなく、記号 `\[` と記号 `\]` で囲みます。すなわち、上掲の出力を得るには

```
方程式$a x^2 + b x + c = 0$の解は
\[
x = \frac{- b \pm \sqrt{b^2 - 4 a c}}{2 a}
\]
である。
```

と入力します。これはもちろん、すべてを繋げて 1 行で

```
方程式$a x^2 + b x + c = 0$の解は\[x = \frac{- b \pm
\sqrt{b^2 - 4 a c}}{2 a}\] である。
```

と書いても構いませんが、前者のように書いたほうが「いま別行立ての数式を書いてるのだ」という雰囲気がでて、ソースファイルが読みやすくなります。

以上、分数はディスプレイスタイルで書くべし、という話をしました。ディスプレイスタイルの数式に対し、文章中に通常の大きさで書かれた数式を、テキストスタイルの数式といいます。いま見たように、分数は、テキストスタイルとディスプレイスタイルで出力の見栄えが異なるのですが、これ以外にも、テキストスタイルとディスプレイスタイルでデザインの変わる記号がいくつかあります。和や積分、極限の記号などです。たとえば

```
和$\sum_{k = 0}^n a_k$と極限$\lim_{x \to 0} f (x)$
```

と入力すると、出力はテキストスタイルで

和 $\sum_{k=0}^{n} a_k$ と極限 $\lim_{x \to 0} f(x)$

となります。このような添え字の付き方が気に入らない場合には、ディスプレイスタイルで

```
和
\[\sum_{k = 0}^n a_k\]
と極限
\[\lim_{x \to 0} f (x)\]
```

と入力すれば

和

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

と極限

$$\lim_{x \to 0} f(x)$$

と、見慣れたデザインのものが出力されます。また、わざわざ別行立てにしたくない場合は `\limits` をつけて

```
和$\sum\limits_{k = 0}^n a_k$と
極限$\lim\limits_{x \to 0} f (x)$
```

と書けば、出力はいちおうテキストスタイルで

和 $\sum\limits_{k=0}^{n} a_k$ と極限 $\lim\limits_{x \to 0} f(x)$

となります。ただし `\limits` を使うと行間が乱れる（この部分だけ前後の行間が広くなってしまう）ので、あまりおすすめしません。

行列：　行列を書くには `pmatrix` 環境を使います。たとえば

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

を出力するには

```
\[
\begin{pmatrix}
1 & 2\\
3 & 4
\end{pmatrix}
\]
```

と書きます。列の区切りが `&` で、行の区切りが `\\` です。

括弧の大きさ：　たとえば「1/2 と 1/3 を足して 6 倍すると 5 になる」ことを数式で書けば

$$6\left(\frac{1}{2}+\frac{1}{3}\right)=5$$

ですが、これを

```
\[6 (\frac{1}{2} + \frac{1}{3}) = 5\]
```

と書くと、出力は

$$6(\frac{1}{2}+\frac{1}{3})=5$$

となってしまいます。括弧の大きさがおかしいでしょう。正しくは

```
\[6 \left(\frac{1}{2} + \frac{1}{3}\right) = 5\]
```

と書きます。このように `\left` と `\right` を使えば、数式に合わせて括弧の大きさが自動的に変わります。

連立方程式：　`\left` と `\right` の使い方をもうひとつ見ます。連立方程式です。たとえば

$$\left\{\begin{array}{l} x^2-xy+y^2=1 \\ xy=1 \end{array}\right.$$

を書くには、`array` 環境をつかって

```
\[
\left\{
\begin{array}{l}
x^2 - x y + y^2 = 1\\
x y = 1
\end{array}
\right.
\]
```

とします。`\left` と `\right` は必ずペアで使います。一方にだけ記号を付けたい場合は、もう片方をドット `.` にします。この場合のドットは出力されません。

場合分け：　関数を場合分けして定義したいような場合、たとえば

$$f(x)=\begin{cases} 0 & \text{ただし } x \text{ が有理数のとき} \\ 1 & \text{ただし } x \text{ が無理数のとき} \end{cases}$$

と書きたいときは `cases` 環境をつかって

```
\[
f (x) =
\begin{cases}
0 & \text{ただし$x$が有理数のとき}\\
1 & \text{ただし$x$が無理数のとき}
\end{cases}
\]
```

と入力します。

複数の数式を並べる：　たとえば複数の等式を並べる場合は

$$\begin{aligned} S_0 &= 1 \\ S_1 &= 1+x \\ S_2 &= 1+x+x^2/2 \end{aligned}$$

のように、等号の位置で揃えると見栄えがよくなります。これは

```
\begin{align*}
S_0 &= 1\\
S_1 &= 1 + x\\
S_2 &= 1 + x + x^2/2
\end{align*}
```

と書きます。揃えたい位置に `&` を置きます。`align` 環境自体が数式モードとなるため、記号 `\[` と `\]` で囲む必要はない（というより囲んではいけない）ことに注意しましょう。これ以外にも、別段位置をそろえる必要はないけれども複数の数式を列挙したい、というような場合には `gather` 環境なども便利に使えます。さらにひとつ注意。ウェブで検索するとよく「複数の数式を並べるには `eqnarray` 環境を使う」と出てきますが、`eqnarray` 環境は今となっては古くさい（不具合がある）ので、使わないほうがよいとされています。

空白の微調整：　文書や数式のスペーシングは基本的には TEX に任せておくのがよいですが、ときどき自分で強制的に調整したほうがよい場合もあります。たとえば、積分

$$\int_0^x \cos t\,dt = \sin x$$

を書くのに

```
\[\int_0^x \cos t dt = \sin x\]
```

とすると、出力は

$$\int_0^x \cos t dt = \sin x$$

となります。よく見比べてください。$\cos t$ と dt の間のスペースが微妙に違います。このような場合は、強制的に空白を挿入して

```
\[\int_0^x \cos t\, dt = \sin x\]
```

と書きます。スペースは次のようなものがあります。

`\quad`	本文のポイントと同じ幅の空白
`\,`	`\quad` の 3/18 ほど
`\␣`	`\quad` の 1/3 から 1/4 ほど（半角スペース相当）
`\;`	`\quad` の 5/18 ほど（等号 = の両側の空きに相当）
`\!`	`\quad` の −3/18 ほど

ここで「ほど」と書いているのは、状況に応じて若干伸び縮みするからです。たとえば `$i,\, j$` のように使ったときと `$a_{i\, j}$` のように添え字の中で使ったときとで、長さが変わります。

太字：　日本語の文章を太字にするには `\textgt` を使いますが、数式には `\boldsymbol` を使います。たとえば、実数全体の集合として $\mathbf{R}$ を出力したい場合は `$\boldsymbol{\mathrm{R}}$` と書きます。おなじ実数全体の集合でも、もし黒板書体 $\mathbb{R}$ のほうが好みならば `$\mathbb{R}$` とします。

命令を自作する：　実数全体の集合は、数学においてもっともよく登場する集合でしょう。その度に `$\boldsymbol{\mathrm{R}}$` と書くのは面倒です。もっと短く書きたい。そんなときはプリアンブルに

```
\newcommand{\R}{\boldsymbol{\mathrm{R}}}
```

と記述しておきます。すると `$\R$` とするだけで $\mathbf{R}$ が出力されます。