シミュレーション演習

G. 総合演習（*Mathematica*演習）

システム創成情報工学科

テキスト作成：藤尾 光彦
講義担当：尾下 真樹

本演習の目的

- さまざまな次元のデータ量を計算機で扱うための基本的な考え方を学習する
  - 1次元、2次元、3次元
  - 質点系、スカラ場、ベクトル場
  - 連続値、離散値
- Mathematica の基本的な使い方を学習する
  - Mathematica とは何か？
  - Mathematica を使ってデータ量を表現する
  - Mathematica を使ってデータ量を可視化する

演習の流れ

- Mathematica の概要
- Mathematica の基本的な使い方
  - 講義＋演習（テキスト G33~G42）
  - テキストのプログラムを入力して実行してみる
- 各自、プリントの演習課題を行う
  - 時間内に課題を終えて提出

演習の流れ（次回以降）

- 計算機でのデータ量の表現
  - 講義（テキスト G1~G9）
- Mathematica を使ったデータ表現と可視化
  - 資料を見ながら各自演習（テキスト G9~G32）
- 各自、プリントの演習課題を行う
  - 時間内に課題を終えて提出

Mathematicaの概要

Mathematicaの特徴

- インタプリタ言語（環境）である
- 数式処理のための機能が充実している
  - 無理数をそのままで扱える
    - 例：$(1/3) \times 3 = 1$
  - 数式を解いたり、記号計算を行ったりできる
    - 例：$y = x^2 - 4x + 4 \rightarrow x = \sqrt{y} + 2$
  - 解析解と数値解の両方を計算することができる
  - 数式やデータを混在させて扱うことができる
- 数値データを図形として描画できる

Mathematicaの応用

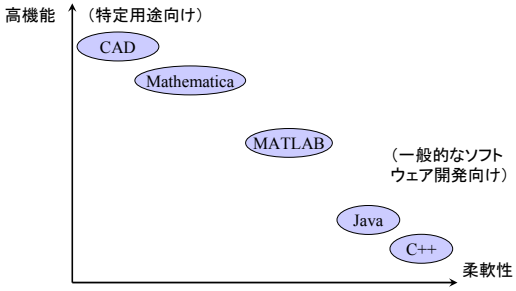
- 何に使えるのか？
- 数式や記号をそのまま扱えるというメリット
  - 因数分解や微積分など、式や記号を含む計算を解かせることができる
  - 数値データを対象とする数学モデル・統計処理・データ解析・分析などの用途で役に立つ
  - 将来、研究や仕事で、数値データ（モデル）を対象とするようなときに使えるかもしれない

プログラミング言語・環境の選択

- 一般にはさまざまな言語・環境が存在
  - それぞれ向き不向きがあるので、状況に応じて適切なものを使う必要がある
    - 一般に高機能な言語・環境は、想定されている用途以外のことをやろうとすると大変
    - 柔軟性の高い言語・環境は、あらゆることができるが、全て自分でやる必要がある
    - 適切な言語・環境は、使用者の習熟度によっても異なる
  - なるべく多くの言語・環境を体験しておくことが望ましい

プログラミング言語・環境の選択

- 言語・環境の比較



インタプリタ言語

- コンパイラ言語
  - C/C++, Java（微妙）, LaTeX など
  - ソースファイルを最初に実行可能な形式に変換
  - 以降は変換後のファイルを処理するので高速
- インタプリタ言語（環境）
  - BASIC など
  - 利用者が対話的に処理をする
    - 入力を逐次解釈しながら実行する
    - 入力を対話的に与えることができる反面、処理速度が遅くなるという問題もある

Mathematicaの基本操作

- Mathematica の実行環境を試してみる
  - シフト＋リターン でプログラムを一行入力
  - すぐに結果が表示される
  - ; をつけると結果を表示しないこともできる
  - % で前の出力結果を参照することができる
  - 前に実行した結果はずっと保持されるので注意
    - 詳しくは後述

In[1]:= 1+1 Shif+リターン

Out[1]= 2

q=1+1

として Shif+リターン すれば

In[2]:= q=1+1

Out[2]= 2

と返ってくる．以後 q を呼び出すたびに

In[3]:= q

Out[3]= 2

と, その値が返ってくる. 変数への代入のときのように, いちいち返り値を見たくないときには行末に ; (セミコロン) をつければよい. このときは Shif+リターン しても, *Mathematica*は何も返さない:

```
In[4]:= r=1+2+3+4
```

しかし, 次に r を呼び出すと

```
In[5]:= r
Out[5]= 10
```

と値が格納されていることが確認できる. 直前の計算結果を参照したいときには % (パーセント) を用いる:

```
In[6]:= s=%+(%+1)+(%+2)+(%+3)
Out[6]= 46
```

もっと前の計算結果を利用したいときには Out[i] または %i を用いればよい.

```
In[7]:= t=%4*%4
Out[7]= 100
```

数式処理のための機能

- Java などの一般のプログラミング言語とは異なり、Mathematica では数式や記号をうまく扱うための機能が充実している
  - 無理数をそのままで扱える
    - 通常は無理数も有限精度にまとめられてしまう
  - 記号計算ができる
  - 解析解と数値解の両方を計算することができる
    - 通常は数値解のみしか計算できない
  - 数式やデータを混在させて扱うことができる

解析解と数値解

- 全ての数値は、近似化されずに扱われる
  - 複素数
  - 無理数
  - π (Pi), e
- 有限精度での近似値を表示する場合は、N[ ] コマンドを使用する
- 式の解も、解析解と数値解の両方を求めることができる
  - Solve[] と NSolve[]

分数は有限精度で近似されずに保持される.

```
In[2]:= q=1/3;
In[3]:= 3q
Out[3]= 1

In[4]:= p=Pi;
In[5]:= p
Out[5]= π
In[5]:= N[p]
Out[5]= 3.14159
```

関数 N で桁数を指定すると必要な桁数だけ近似値を返す.

```
In[6]:= N[p,100]
Out[6]= 3.14159265358979323846264338327950288419716939937510582097
        4944592307816406286208998628034825342117068
```

Mathematicaのコマンド

- Solve[式, 変数], NSolve[式, 変数]
  - 与えられた式を指定された変数について解く
- N[式 or 変数]
  - 式 or 変数の有限精度の数値解を表示
- Sum[式, { 変数, 開始値, 終了値 }]
  - Σ 計算
  - 変数の値を開始値～終了値まで変化させながら、与えられた式の値の和を計算

```
In[7]:= Solve[x^2+3x+1==0, x]
Out[7]= {{x → 1/2 (-3 - √5)}, {x → 1/2 (-3 + √5)}}

In[8]:= NSolve[x^2+3x+1==0, x]
Out[8]= {{x → -2.61803}, {x → -0.381966}}

In[1]:= Sum[x^i, {i, 0, 17}]
Out[1]= 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 +
        x^10 + x^11 + x^12 + x^13 + x^14 + x^15 + x^16 + x^17
```

記号計算

- 記号計算
  - x などの記号（変数）をそのまま扱うことができる
  - 記号を使った計算ができる
    - 例：式をある変数について解く
    - 例：式を素因数分解
- 注意
  - x などの変数に実際に値を代入すると、以降、式は自動的に値を計算する
  - 記号の状態にもどすときは、変数をクリアする

```
In[1]:= Sum[x^i, {i, 0, 17}]
Out[1]= 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 +
        x^10 + x^11 + x^12 + x^13 + x^14 + x^15 + x^16 + x^17
In[2]:= Factor[%]
Out[2]= (1+x) (1-x+x^2) (1+x+x^2) (1-x^3+x^6) (1+x^3+x^6)
```

```
In[5]:= x
Out[5]= -1
In[6]:= Sum[x^i, {i, 0, 17}]
Out[6]= 0
```

文字としての x を復活させるには値をクリアする必要がある．それには x=. とする．

```
In[7]:= x=.
```

関数定義

- 関数を定義できる
  - f[x_] := x ^ 2
    - 仮引数には _ をつける
  - f[10], f[y] などと使える

```
In[1]:= f[n_]:=Sum[x^i, {i, 0, n}]
In[2]:= f[3]
Out[2]= 1 + x + x^2 + x^3
In[3]:= f[17]
Out[3]= 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 +
        x^10 + x^11 + x^12 + x^13 + x^14 + x^15 + x^16 + x^17
```

リスト表現

- Mathematica では、多次元のデータを全てリスト表現により表す
  - リストの宣言方法
    - q = {0, 1, 4, 9, 16}
  - データへのアクセス方
    - q [[3]]
  - リストを入れ子にすることで、2次元以上のリストを表現
  - ベクトルや行列も全てリストにより表現することができる

Mathematicaのコマンド

- Table[式, { 変数, 開始値, 終了値 }]
  - Sum[…] と同様
  - 変数の値を変化させながら、各変数値から計算された式の値を要素とするリストを作成
- Table[式, { 変数, 開始値, 終了値 }]
  - 入れ子にすることもできる
  - Table[i^j, {i, 0, 4}, {j, 1, 4 }]
  - 以下のような Java プログラムに相当

```
for ( i=0; i=4; i++)
    for( j=1; j<=4; j++ )
        table[ i ][ j ] = i ^ j;
```

```
In[3]:= c=Table[i^3, {i, 0, 4}
Out[3]= {0, 1, 8, 27, 64}
In[4]:= c[[4]]
Out[4]= 27
```

Table 関数を入れ子にすることで2次のリスト (リストのリスト) がつくれる.

```
In[5]:= d=Table[i^j, {i, 0, 4}, {j, 1, 4}]
Out[5]= {{0,0,0,0}, {1,1,1,1},
         {2,4,8,16}, {3,9,27,81}, {4,16,64,256}}
```

2次のリストの i 番目の成分は通常のリストである.

```
In[6]:= d[[3]]
Out[6]= {2, 4, 8, 16}
```

リスト同士の演算

- リスト同士の演算は、リストの各要素同士に対して適用される
- そのままではベクトルや行列同士のかけ算ができない
  - ベクトル同士の内積には .（ピリオド）
  - ベクトル同士の外積には Cross[] 関数
  - 行列同士の積・行列とベクトルの積にも .（ピリオド）

リスト操作

- RotateRight[リスト]
  - リストの要素を1つずつ右に移動
  - リストの一番右の要素は、一番左の要素にする
  - 例: {2, 4, 8, 16} → {16, 2, 4, 8}
- Position[リスト, 値]
  - リストの中で、値のある位置を出力する
  - Position[{2, 4, 8, 16}, 4] → 2

リストに対する演算はリスト操作と数学的演算がある.

```
In[7]:= RotateRight[d[[3]]]
Out[7]= {16, 2, 4, 8}
In[8]:= Position[d[[3]],4]
Out[8]= {{2}}
In[9]:= 3d[[3]]
Out[9]= {6, 12, 24, 48}
In[10]:= d[[3]]+d[[4]]
Out[10]= {5, 13, 35, 97}
In[11]:= d[[3]]/d[[4]]
Out[11]= {2/3, 4/9, 8/27, 16/81}
```

```
In[1]:= q={x, y, z};
In[2]:= p={u, v, w};
In[3]:= m={{a, b, c}, {e, f, g}, {h, i, j}};
```

ベクトルや行列の基本演算である加法とスカラー倍は前節で説明したリスト演算でそのまま行える.

```
In[3]:= q+p
Out[3]= {x+u, y+v, z+w}
In[4]:= 3m
Out[4]= {{3a, 3b, 3c}, {3e, 3f, 3g}, {3h, 3i, 3j}};
```

そのままかけてしまうと成分ごとの積になってしまうので，内積や外積などのベクトルの積は注意が必要である. 内積は □ (ピリオド) を使い，外積は Cross 関数を使う.

```
In[5]:= q.p
Out[5]= u x + v y + w z
In[6]:= Cross[q, p]
Out[6]= {w y - v z, -w x + u z, v x - u y}
```

行列もそのまま並べてしまうと成分ごとの積となるので，行列としての積を計算したいときは内積と同じく⊡ (ピリオド) を使う．⊡ (ピリオド) は行列とベクトルの積にも用いられる．

```
In[7]:= m m

Out[7]= {{a^2,b^2,c^2}, {d^2,e^2,f^2}, {g^2,h^2,i^2}}

In[8]:= m.m

Out[8]= {{a^2 + b d + c g, a b + b e + c h, a c + b f + c i},
         {a d + d e + f g, b d + e^2 + f h, c d + e f + f i},
         {a g + d h + g i, b g + e h + h i, c g + f h + i^2}}

In[9]:= m.q

Out[9]= {a x + b y + c z, d x + e y + f z, g x + h y + i z}
```

演習