Farlow の第7課の表と図を再現してみる

桂田 祐史

2021年12月10日, 2021年12月10日

1 はじめに

何か文献 (テキストや論文) をきちんと理解するために読むとき、計算などは自分で再現するよう努力すべきである (「こうなるそうです」ではなくて「確かにそうなりました。」, 「ここは本当はこうだと考えます。」)。

数値計算はもちろん、式の計算もコンピューターに任せて構わない (もちろん自分で計算すると理解が深まることが多いが、自分で計算することにこだわらないように)。

今回は Farlow [1] の第7課の固有値の計算、固有関数のグラフ描き、解のグラフ描きを再現してみる。

2 Mathematica による再現

実は、桂田研の学生で微分作用素の固有値問題に関係する研究テーマを選んだ人は多く、この節の内容はそのときのノウハウに基づく。
(Mathematica を使ったのは歴史的経緯で、他のものでは出来ない、というわけでもない。)

2.1 使うことの説明

1. 現象数理学科 Mac にインストールされている **Mathematica** は、**数式処理系**と呼ばれるソフトウェアである。プログラミング言語処理系の一種でもあるが、多くのプログラミング言語 (例えば C, Python, Julia, MATLAB, …) は、数値計算はできても数式の計算はできない。

2. 基本的な操作

   - 現象数理学科でライセンスを購入しているので、所属する学生は利用できる (はず)。動かない場合は、桂田か池田先生に相談する (ライセンス・キーの発行ができる)。
   - アプリケーション・フォルダに Mathematica.app がある (私は Dock に追加しています)。
   - (新しくプログラムを作る場合) Mathematica を起動後、「新規ドキュメント」でノートブックを開き、コマンドを入力して実行する。
   - コマンドの最後に shift+return (enter) とタイプする。
   - 直前の結果は % で参照できる。直前のコマンドは command+L で呼び出せる。
   - コマンドは編集して再実行できる (挿入、上書き修正、削除、などが可能)。

- ??関数名 としてマニュアルが開ける (非常に便利。これに慣れること。)。
- 関数名の大文字・小文字に注意する。用意されている関数名の先頭は大文字である。
- ノートブックとして保存しておける (ファイル名の末尾は .nb)。
  保存前に不要なものを削除するのがお勧め (やり方を文章で説明するのは面倒なので、やって見せたときに覚えること)。
- 既存のノートブックはダブルクリックで開ける。
  コマンドを 1 つ 1 つ shift+return で実行する以外に、[評価] → [ノートブックを評価] で順番に全部実行することもできる。
- 計算の実行中に止めるには、[評価] → [評価を放棄] (またはキーボードで command+.) とする。

3. 覚えるべき関数

- 1 変数関数のグラフを描くには、Plot[f[x],{x,a,b}] とか Plot[{f[x],g[x]},{x,a,b}]
  グラフィックスを保存するには、変数 = Plot[f[x],{x,a,b}] で記憶しておいて、
  Export["ファイルの名前", 変数]

```
Plot[Sin[x],{x,-10,10}]
g1=Plot[{Sin[x],Cos[x]},{x,-10,10}]
Export["sincosgraph.png", g1]
```

- 簡単な方程式を解くには、Solve[x の式 1 == x の式 2,x]　(今回は使わない)
  未知数とする x に何か値が代入してあってはダメ (事前に Clear[] しておく)。

```
Clear[x]
Solve[x^2+2x+3==0,x]
Clear[x,y]
Solve[{2x+3y==4,5x-6y==7},{x,y}]
```

- Solve[ ] で解けない方程式を反復法で解くには、FindRoot[x の式 1 == x の式 2, {x, x0}]
  初期値 x0 が必要。

```
Clear[x]
FindRoot[Tan[x]==-x,{x,2}]
FindRoot[Tan[x]==-x,{x,2},WorkingPrecision->100]
```

  WorkingPrecision (作業精度) のデフォールトは、WorkingPrecision->MachinePrecision (普通の C 言語処理系の double 相当で 16 桁弱) である。
- 繰り返しをしたいときは For[] もあるけれど、Table[] を使うのが大抵の場合のお勧め。(結果はリストになり、[[番号]] で要素にアクセスできるが、今日は使わないので省略する。)

```
Table[n^2,{n,10}]
Table[n^2,{n,1,10}]
Table[n^2,{n,1,10,1}]
```

(どれも結果は同じである。)

```
square=Table[x^2,{x,0,1,1/10}]
```

次の違いを理解しよう (実行してから考えるで構わない)。

```
Table[Plot[Sin[n x], {x, 0, 2 Pi}], {n, 5}]

Plot[Table[Sin[n x], {n, 5}], {x, 0, 2 Pi}]
```

- 定積分 $\int_a^b f(x)dx$ の計算は、`Integrate[f[x],{x,a,b}]` とする。

```
Integrate[Sin[lam x]^2,{x,0,1}]
```

Cf. [1] p. 57 の $\int_0^1 \sin^2(\lambda_m \xi)d\xi = \dfrac{\lambda_m - \sin\lambda_m \cos\lambda_m}{2\lambda_m}$ と見比べてみよう。

- 自分で関数を定義する。関数の名前 `[x_]:=x` の式 という形式。関数の名前を使ってあるとおかしなことが起こることがある (定義する前に `Clear[]` する習慣をつけよう)。

```
Clear[f]
f[x_]:=Sin[x]
```

Mathematica の関数は、数学でいう関数と違い、戻り値を使わない場合もある。

```
sincosgraph[n_]:=Plot[{Sin[n x],Cos[n x]},{x,0,2Pi}]
```

4. 漸化式で定まる数列の、Mathematica での扱いの工夫。これは良く出て来る。フィボナッチ数列 ($a_1 = a_2 = 1$, $a_n = a_{n-1} + a_{n-2}$ $(n \geq 2)$) を計算する関数を定義してみる。

```
Clear[a]
a[n_]:=Which[n==0,1,n==1,1,True,a[n-1]+a[n-2]]
Table[a[n],{n,0,10}]
```

10 ならば良いが、40 くらいだと大変になる (「実行中」のまま計算がなかなか終わらない)。無駄な計算をたくさんするから。command+. で計算を放棄する。次が改良版。

```
Clear[a]
a[n_]:=a[n]=Which[n==0,1,n==1,1,True,a[n-1]+a[n-2]]
Table[a[n],{n,0,100}]
```

2.2 $\tan x = -x$ を解く

FindRoot[] を使って、$\tan x = -x$ の正の範囲の解を求める。$y = \tan x$, $y = -x$ のグラフを描いて、2 の近くの解を求めてみる (16 桁精度、50 桁精度)。

```
f[x_]:=Tan[x]
g[x_]:=-x
Plot[{f[x],g[x]},{x,0,100}]

solnear[x0_]:=FindRoot[f[x]==g[x],{x,x0}]
solnear[2]

solnear2[x0_,prec_]:=FindRoot[f[x]==g[x],{x,x0},WorkingPrecision->prec]
solnear2[2, 50]
```

Solve[] にしても FindRoot[] にしても、解の値ではなく、代入規則を結果として返す。

代入規則でなく、値が欲しいならば。x /. (変数名 スラッシュ ドット) の後に代入規則を続けて、x への一時的な代入を実行すれば良い。

```
solnear3[x0_]:=x /. FindRoot[f[x]==g[x],{x,x0}]
solnear3[2]
```

初期値の選び方が大事。ちょっと自分で考えて何回か試してみよう (こういうのが大事です)。

$\tan x = -x$ の正の解の、小さい方から n 番目の値を λ_n として、それを計算する関数 lambda[] を作った (そのコードは後述する)。それを使うと lambda[n] で λ_n が得られる。

実行結果 (In[] := の右側が入力したもの)

```
In[] := lambda[1]
Out[] = 2.02876
In[] := lambda[5]
Out[] = 14.2074
In[] := Table[lambda[n],{n,10}]
Out[] = {2.02876, 4.91318, 7.97867, 11.0855, 14.2074, 17.3364, 20.4692, 23.6043, 26.7409, 29.8786}
```

Farlow [1] の 表 7.1 と見比べよう。

2.3 固有関数のグラフを描く

関数 lambda[] が出来れば、固有関数のグラフをとりあえず描くのは簡単である。

```
Table[Plot[Sin[lambda[n] x],{x,0,1}],{n,5}]

Plot[Table[Sin[lambda[n] x],{n,5}],{x,0,1}]
```

やや分かりにくいので、少し工夫する (グラフの体裁を整えるのは意外と大変)。

```
gr=Plot[Evaluate[Table[Sin[lambda[n] x],{n,5}]],{x,0,1},PlotLegends -> "Expressions"]
Export["eigenfunction12345.png", gr]
```

Cf. 図 1 を、[1] p. 56 の図 7.4 と見比べてみよう。

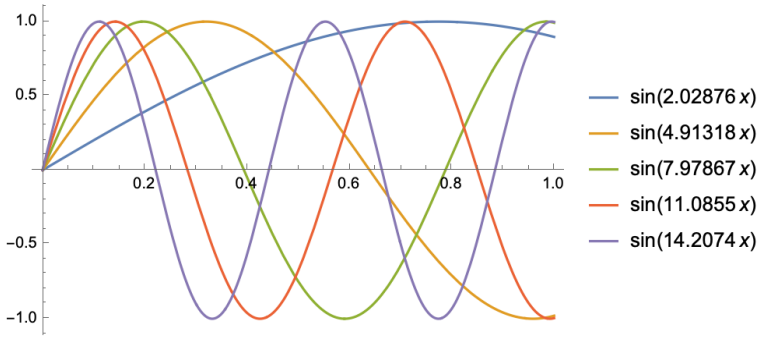

図 1: $X_n(x)$ のグラフ $(n = 1, 2, 3, 4, 5)$

2.4 解の係数を求める

私は、信号処理とフーリエ変換という授業で次のように教えています。

直交系による展開の係数

直交系 $\{X_n\}_{n\in\mathbb{N}}$ によって、関数 f が

$$f(x) = \sum_{n=1}^{\infty} a_n X_n(x)$$

と展開されているとき、

$$a_n = \frac{(f, X_n)}{(X_n, X_n)}$$

が成り立つ。ただし $(\cdot,\cdot)$ は次式で定義される関数の内積である。

$$(f, g) = \int_0^1 f(x)\overline{g(x)}dx.$$

$f(x) = x$ の場合は、$a_n = \dfrac{(x, X_n)}{(X_n, X_n)}$, $X_n(x) = \sin\lambda_n x$ であるから、次のコードで計算できる。

```
Clear[a]
a[n_]:=a[n]=Integrate[x Sin[lambda[n] x], {x,0,1}]/Integrate[Sin[lambda[n] x]^2, {x,0,1}]

Table[a[n],{n,10}]
```

次のような結果を返す。

```
Out[] = {0.729175, -0.156164, 0.0613973, -0.0321584, 0.0196707,
         -0.0132429, 0.00951282, -0.00715998, 0.0055821, -0.00447313}
```

[1] p. 58 の表 7.2 と見比べてみよう。

2.5 Fourier 級数による解の部分和のグラフを描く

```
u[x_, t_] := Sum[a[n] Sin[lambda[n] x] Exp[-lambda[n]^2 t], {n, 1, 10}]
Plot[Table[u[x, t], {t, 0, 0.20, 0.05}], {x, 0, 1}]
```

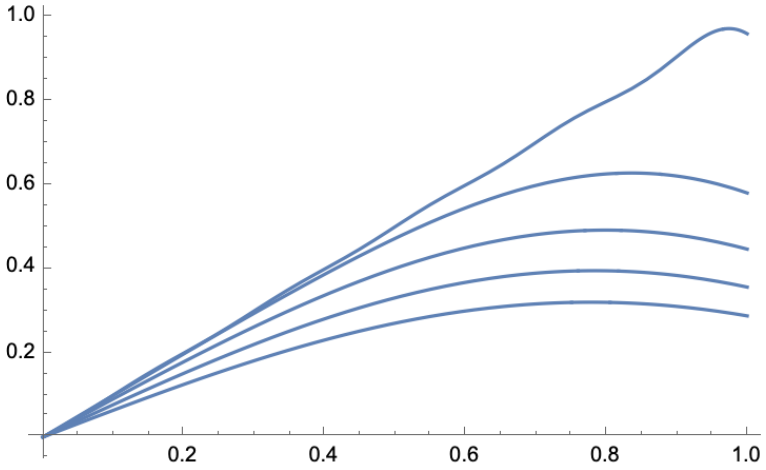

図 2: 10 項までの部分和によるグラフ $(t = 0, 0.05, 0.10, 0.15, 0.20)$

$x = 1$ の近くでは今ひとつ。30 項の和として計算すると、途中で “Exp[-746.589] は正規化された機械数として表すには小さすぎます．精度が失われる可能性があります．” のような警告が出る。意外と難しい。

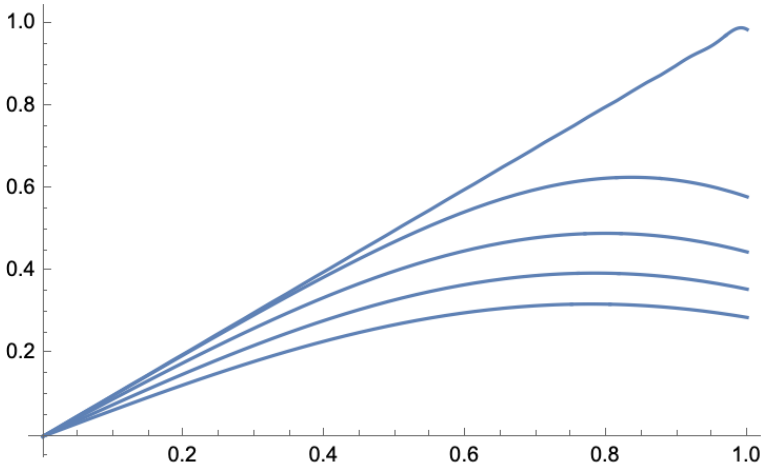

図 3: 30 項までの部分和によるグラフ $(t = 0, 0.05, 0.10, 0.15, 0.20)$

図 4 は工夫して 100 項の和で描いた。

2.6 lambda[] 1 つの解答

実は $\lim_{n\to\infty}(\lambda_{n+1} - \lambda_n) = \pi$ である (グラフを考えると納得できるかも)。そこで $n \geq 2$ に対して λ_n を求めるときの初期値を $\lambda_{n-1} + \pi$ としてみた。幸いこれで動いてくれた。

```
Clear[lambda]
lambda[n_]:=lambda[n]= If[n== 1, solnear3[2], solnear3[lambda[n - 1] + Pi]]
```

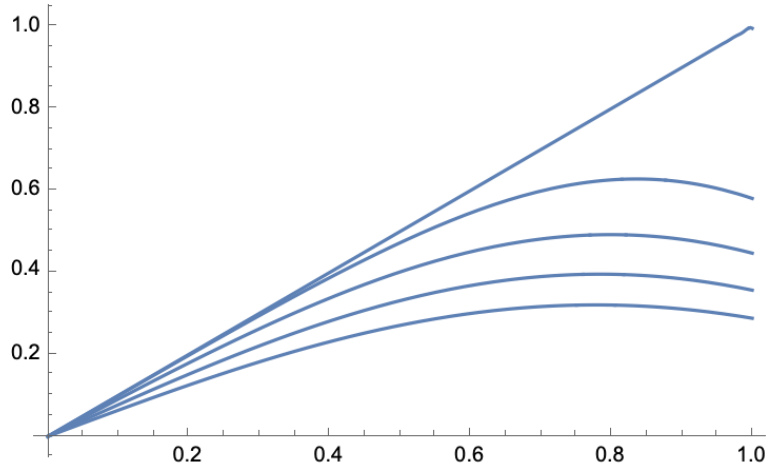

図 4: 100 項までの部分和によるグラフ (どうやったかは秘密)

3 Mathematica 以外のプログラミング言語でやってみる

3.1 方針

C, Python, Julia などのプログラミング言語でも解いてみよう。このゼミでは、C と Julia のサポートをするが、Python を使いたければそうしても構わない。

Mathematica の `FindRoot[]` のところをどうするかが問題となる。Newton 法を使うと良い。

Newton 法

$f(x) = 0$ を解くために、適当な初期値 x_0 を選んで、

$$(\heartsuit) \qquad x_{n+1} = x_n - f'(x_n)^{-1} f(x_n) \quad (n = 0, 1, 2, \cdots)$$

により $\{x_n\}$ を定めると、多くの場合に解に収束することが期待できる。

(♡) の意味 x_n の十分近くでは、$f(x)$ は

$$f'(x_n)(x - x_n) + f(x_n)$$

により近似できる。$f'(x_n)(x - x_n) + f(x_n) = 0$ を解いて、$x = x_n - f'(x_n)^{-1} f(x_n)$. これを x_{n+1} に採用した、ということである。

解 a が $\det f'(a) \neq 0$ を満たすならば (重解でないならば)、$\{x_n\}$ は “2 次の収束” をする (とても速く真の解に近づく) ことが期待できる。

3.2 Julia で Newton 法

まず Julia をインストールしておこう。「インストール」[1]

[1] http://nalab.mind.meiji.ac.jp/~mk/labo/text/julia-memo/node4.html

newton.jl

```
# newton.jl

using Printf
using Plots

function newton(f, df, x0, eps)
  x = x0
  for i=1:10
    dx = f(x) / df(x)
    x = x - dx
    #@printf("Δ x=%e, x=%g, f(x)=%e\n", dx, x, f(x))
    if abs(dx) < eps
      return x
    end
  end
  println("newton: 収束しませんでした。修正量=$(dx)")
  return x
end

#
println("問題 1")
f(x) = x^3 - 2x - 5
df(x)=3x^2-2

println("通常の浮動小数点演算")
println(newton(f, df, 2.0, 1e-14))

println("BigFloat を用いた 10 進 100 桁計算")
# 2 進 350 桁 (10 進 100 桁のためには 330 桁あれば良い)
setprecision(350)
println(newton(f, df, BigFloat(2), 1e-100))

#
println("問題 2")
f(x) = tan(x)+x
df(x)=1/cos(x)^2+1
println("通常の浮動小数点演算")
println(newton(f, df, 2.0, 1e-14))

println("10 進 100 桁計算")
println(newton(f, df, BigFloat(2), 1e-100))
```

@printf() の注釈を外すと中間結果が見える。

Julia で Newton 法を試す (% や julia> の右側を入力する)

```
% curl -O http://nalab.mind.meiji.ac.jp/~mk/misc/20211210/newton.jl

% julia
julia> include("newton.jl")
通常の浮動小数点演算
2.0945514815423265
BigFloat を用いた 10 進 100 桁計算
2.094551481542326591482386540579302963857306105628239180304128529045312189983483667146267281777157757
通常の浮動小数点演算
2.028757838110434
10 進 100 桁計算
2.028757838110434223576971124734714376108380028759394088817166074449866503104276234592279515042563063
```

あるいは

```
% julia newton.jl
```

このプログラムを発展させて、解 $u(x,t)$ を計算してグラフを描くプログラムを描いてみよう。

4 差分法

差分法で解いてみよう (そのうちにやるつもり)。境界条件 $u_x(1,t)+u(1,t)=0$ をどのように扱うかが問題となる。桂田 [2] の第 1 章に説明がある (§1.5.1)。

参考文献

[1] Farlow, S. J.: *Partial Differential Equations for Scientists and Engineers*, John Wiley & Sons, Inc (1982), 邦訳: スタンリー・ファーロウ 著, 入理 正夫・入理 由美 訳, 偏微分方程式, 朝倉書店 (1996).

[2] 桂田祐史：熱方程式に対する差分法 I — 区間における熱方程式 —, http://nalab.mind.meiji.ac.jp/~mk/labo/text/heat-fdm-1.pdf (1998 年〜).