コンピュータによる数値演算の精度と注意

数値演算プログラムの一般的な注意

浮動小数点型の演算では、常に誤差を意識すること

- 変数長の制限による誤差: underflow, overflow
  IEEE 754の標準で、64bit浮動小数点の範囲は
  指数部: 11 bit -1024 ~ +1023
  仮数部: 23 bit 4,503,599,627,370,495: 16桁
- **浮動小数点では、整数を “正確に” 表現できない**
  100.0_{10} $= 1.5625 \times 64 = (1+2^{-1}+2^{-4}) \times 2^6 = (1.1001)_2 \times 2^4$
- **有限の桁数の浮動小数点の表現は、ほぼすべての場合に誤差を含む**
  1.0/3.0 ＝ 0.3333…333 (小数点以下16桁)　　10^{-16} 程度の誤差が発生する

条件分岐の判断:

悪い例:

if x * 10.0 == 30.0:　x = 3.0 であっても、**true と判断される保証はない**

よい例:

eps = 1.0e-30　# epsilon: 想定される誤差よりも十分大きいが、なるべく小さい値を設定する。
if **abs(x * 10.0 - 30.0) < eps**:　**誤差 eps 以内で必ず実行される**

浮動小数点 => 整数変換: xmin ~ xmax の範囲を xstep 毎の幅で分割したときの分点の数

悪い例:

n = int((xmax – xmin) / xstep) + 1: int() の中が誤差により 3.99999… となった場合、
本来はint() = 4 となって欲しいのに、3 となってしまう

よい例:

eps = 1.0e-6
n = int((xmax – xmin) / xstep + eps) + 1: int() の中が誤差により期待する整数値より小さくなっても、
誤差がepsより小さければ、本来期待している整数値が得られる

Numeric representation: Integer (整数型)

Integer type: Based on the CPU bit (CPUのbit数が基本)

16bit for 16bit CPU

unsigned int (符号無し整数型) **$0 \sim 2^{16} - 1 = 65{,}535$**

signed int (符号付き整数型) **-32,768 ~ +32,767**

32bit for 32bit CPU

unsigned int (符号無し整数型) **0 ~ 4,294,967,295**

signed int (符号付き整数型) **-2,147,483,648 ~ + 2,147,483,647**

For all CPUs:

int : depends on CPU bits

short int : 16 bit

long int : 32 bit

long long int: 64 bit

Numeric representation: Floating point, Real

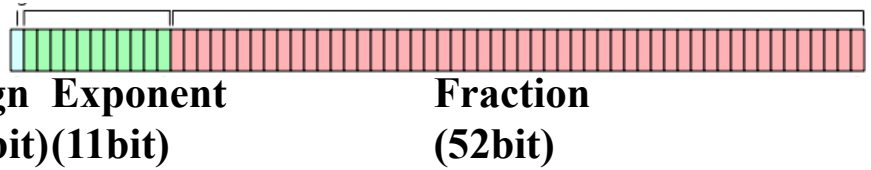
(浮動小数点型, 実数)

Floating point type: Minimum 32bit (except half precision)

The range of available value depends on computer architectures, programming language etc.

$$-(1.011101)_2 \times 2^{-0101}$$

Sign (符号) — **Fraction** (仮数部) — **Exponent** (指数部)



C language (C言語)

float : 32 bit 3.4E-38 ~ 3.4E+38

double : 64 bit 1.7E-308 ~ 1.7E+308

long double: 64 bit

Fortran

Single precision (単精度) FP (REAL) : 32 bit

Double precision (倍精度) FP (DOUBLE) : 64 bit, 16 digits (桁) in decimal

Quadruple precision (4倍精度) FP (REAL*16) : 128 bit

Definition of IEEE 754 (binary32, binary64):

Sign : 1 bit

Exponent: 8 bit (REAL, -128 ~ +127) 11 bit (DOUBLE, -1024 ~ +1023)

Fraction : 23 bit (REAL) 52bit (DOUBLE)

8,388,608: 7 digits 4,503,599,627,370,495: 16 digits

Error of floating point (浮動小数点の誤差)

Representation of floating point in computer:

$-(1.01100101)_2 \times 2^{-015}$

Errors come by converting Base 10 to Base 2.

- **Some values do not have errors between Base 10 and Base 2 if fraction equals to 2^n**

  1.0 $= (1.0)_2 \times 2^0$

  0.5 $= (1.0)_2 \times 2^{-1}$

  0.125 $= (1.0)_2 \times 2^{-3}$

  0.039063 $= 1.25 \times 2^{-5} = (1.01)_2 \times 2^{-5}$

  100.0 $= 1.5625 \times 64 = (1+2^{-1}+2^{-4}) \times 2^6 = (1.1001)_2 \times 2^4$

- **Other values have errors even if it is represented by a simple figure in Base 10:**

  $0.1 = (1.1001100110011001\cdots)_2 \times 2^{-3}$

python

Pythonの基礎

Ver 2.X	Ver 3.X
コンソール出力 print は組み込み構文のため、() がなくてもよい print “x=“, x	print が関数になったため、()が必要。 print(“x=“, x) 改行を抑制するには end = ‘’ 引数を与える print(“x=“, x, end = ‘’)
文字列で uniconde を指定する場合は u’’ を使う s = u’パイソン‘	文字列はunicondeに統一されているので、u’’ は不要
Ver3 .Xには Ver 2 =>Ver 3 変換ツールが含まれている 2to3 –w v2_script.py => バックアップファイル v2_script.py.bak をつくり、Ver 3のファイルが v2_script.pyに出力される 参考: https://www.python-izm.com/tips/2to3/	

PythonのTips

モジュールのインストール

参考: https://insilico-notebook.com/conda-pip-install/

Anacondaの場合、conda も使う方がよい (競合によるトラブルが少ない)

conda install module_name

一般の場合は pip を使う

pip install module_name

必要な環境変数設定

PATH=C:¥Anaconda3;C:¥Anaconda3¥Library¥mingw-w64¥bin;C:¥Anaconda3¥Library¥usr¥bin;C:¥Anaconda3¥Library¥bin;C:¥Anaconda3¥Scripts;C:¥Anaconda3¥bin;C:¥Anaconda3¥condabin

PYTHONPATH= 標準以外のモジュールのルートパス

Anaconda固有の環境変数

CONDA_DEFAULT_ENV=base

CONDA_EXE=C:¥Anaconda3¥Scripts¥conda.exe

CONDA_PREFIX=C:¥Anaconda3

CONDA_PROMPT_MODIFIER=(base)

CONDA_PYTHON_EXE=C:¥Anaconda3¥python.exe

CONDA_SHLVL=1

Pythonと他言語の比較

python	C	Perlなど
変数はすべてオブジェクト (実態は 辞書型 (連想配列) 変数)	オブジェクト指向はC++などで拡張	オブジェクト指向は連想配列を使って疑似的に実現
変数宣言 ・宣言構文はない。 ・値を代入したときに変数が生成され代入する値によって変数の型が決まる。 ・違う型の値の代入により、変数の型も変わる。 a = 10　整数型 a = 10.0　浮動小数点型 a = '10'　文字列型	・明示的な型宣言が必須。 ・変数の型は変更できない。 int a;　整数型 double a = 10.0;　浮動小数点型 char *a = '10' ;　文字列型	・明示的な宣言は必要ない。 ・my, local で明示的に宣言できる。 ・use strict; を使うことで、明示的な変数宣言を必須にできる。 ・代入する値によって変数の型が決まる。 ・値の代入により変数の型も変わる。 $a = 10 ;　整数型 my $a = 10.0;　浮動小数点型 my $a = '10' ;　文字列型
文区切り ・原則として改行で区切り ・文末に ; をつけてもエラーにはならない ・継続行を使うには、行末に ¥ をつけ、改行コードをエスケープする ・括弧 (の後に改行をいれても、継続行と判断される。	・; を見つけると文の区切りになる	・; を見つけると文の区切りになる
ブロックの定義 ・インデントの大きさが同じ範囲が同一のブロックになる	・{} で囲まれた範囲が一つのブロック	・{} で囲まれた範囲が一つのブロック (pascalでは begin ~ end)

Pythonと他言語の比較

python	C	Perlなど

サブルーチン、関数定義
・def 文で関数を定義。
・同一インデントのブロック範囲が関数
・引数は関数定義で受け取る
・return文により戻り値を返すことができる
・関数自体は関数型の変数
def func_name(arg1, arg2):
　sum = arg1 + arg2
　return sum
	・(引数) を与えるとdef 文で関数を定義。
	・値を返す関数はreturn文が必須
	・値を返さない関数は void型宣言をする
	・引数は関数定義で受け取る
	・関数自体は関数型の変数
	int func_name(int arg1, int arg2) {
	　int sum = arg1 + arg2;
	　return sum;
	}
	void func_name(int arg1,) {
	　printf(“arg = %d¥n”, arg1);
	}
		・sub 文で関数を定義
		・return文により戻り値を返すことができる
		・returnで戻り値を返さない場合、戻り値は undef になる
		・引数は関数内で @_ リスト変数として受け取る: 関数内でわかりやすい名前の変数に展開する方がいい
		・関数自体は関数型の変数
		sub func_name　{
		　my ($arg1, $arg2)　= @_;
		　$sum = $arg1 + $arg2;
		　return $sum;
		}

PythonのTips: 局所変数と大域変数

参照: http://conf.msl.titech.ac.jp/Lecture/python/python-tips.html

変数の型とスコープは代入時に決まる。
- 関数外で代入すると大域変数
- 関数内で代入すると局所変数

【注意】関数内で、大域変数と同じ名前の変数に代入すると、その名前の局所変数が生成される。
関数内で大域変数へ代入する場合は、global 宣言を行う

global1.py

```
a = 1

def f():
    a = 2          局所変数を生成、大域変数は変えない
    print("in func:", a)

print("outer func1:", a)
f()
print("outer func2:", a)
```

実行結果

```
outer func1: 1      大域変数
in func: 2          局所変数
outer func2: 1      大域変数
```

global2.py

```
a = 1

def f():
    global a
    a = 2                    大域変数に代入
    print("in func:", a)

print("outer func1:", a)
f()
print("outer func2:", a)
```

実行結果

```
outer func1: 1      大域変数
in func: 2          大域変数
outer func2: 2      大域変数
```

基本的に、**大域変数は極力使わない方がいい**
多くの大域変数を使う場合、class宣言をしてオブジェクトの attribute として変数を宣言する
ただし、pythonのオブジェクトは辞書型を使うため、実行速度は遅い

Pythonのfor 文

一般的なプログラミング言語の for 文

BASIC: for i = 1, 10, 2

i を 1 から 10 まで、i に 2 を加えながら変えて、ブロックを繰り返し実行

C : for(int i = 1; i <= 10; i += 2)

i を 1 に初期化し、i に 2 を加えながら、I <= 10 の条件を満足している間、
ブロックを繰り返し実行 (perl, phpなど、pascal以降の言語の多くがこの形式を採用)。
BASIC, Fortranなどと同じ機能を持つが、より柔軟な繰り返し処理が可能。

```
int c = 0
for(int i = 1, k = 0; i <= 10 and k > 10; i += 2, k -= 1) {      # 複数のループ変数を使える
      c += i;
       if(k > -3) {
           i = 3;                                  # ループ変数をブロック内で変更できる
       }
 }
```

Python の for 文は、イテレータをリストとして渡す。

```
for v in list:
      a += v
```

BASICのような使い方をする場合、繰り返す値をすべて list 変数に入れて渡す。リスト変数を作製する際には、range() 関数を使うことが多い。

```
for i in range(0, 10, 2) :
      a += v[i]
```

for 文とリスト内包表記

Python の for 文:

```
a = []
for i in range(0, 10, 2) :
        a.append(v[i] + i)
```

これを 1行で書くことができる: リスト内包表記

a = [v[i] + i for i in range(0, 10, 2)]

- v[i] + i の計算を range(0, 10, 2) の各値 i について実行
- 実行した値を **リスト []** で返す

Python はインタープリタであるため、通常の文の実行は遅い
⇒リスト内包表記の方が圧倒的に早い場合がある。

文字列の整形: .format と置換フィールド

参考: https://gammasoft.jp/blog/python-string-format/

Pythonの変数はすべてオブジェクト: 文字列型にもいろいろな attribute (メソッド) がある

a = 1.0
b = 1.0

・書式指定なし
print(“a=“, a, “ b=“, b)

・.format() と 置換フィールド {} で文字列を整形する
“a={} b={}“.format(a, b)　　=> “a=1.0 b=2.0” という文字列を返す
print(“a={} b={}“.format(a, b))　　=> print関数などに渡せる

・f”{変数名}” で文字列を整形する
print(f“a={a} b={b}“)

・f”{変数名}” で文字列を整形する
print(f“a={a} b={b}“)

・置換フィールド {インデックス番号:書式指定} で文字列を整形する
{2:.1f}　　2つ目の引数の値を、浮動小数点 (f) として小数点以下1桁に整形
{:12.4e}　次の引数の値を、指数形式 (e) で小数点以下4桁、最小幅12桁で整形

文字列の整形: % を使う C の printf形式

参考: https://note.nkmk.me/python-print-basic/

a = 1.0
b = 1.0

・C言語の printf() 関数と同じ書式も使える
注意: 変数の型によっては変換に失敗する。.format()を推奨

“a=%3.1f b=%3.2f“ % (a, b)　　=> “a=1.0 b=1.00” という文字列を返す
print(“a=%3.1f b=%3.2f“ % (a, b))　　=> print関数などに渡せる

%5d	整数型で最低5桁
%12.4g	浮動小数点型で小数点以下4桁、最低12桁
%s	文字列型を右詰めで表示
%-s	文字列型を左詰めで表示

神谷作製 pythonプログラムの構造

実行方法: だいたい次のどれか

1. **開発初期や簡単なプログラム:**
   パラメータをプログラム中にハードコーディング。引数なしの実行のみ。
   python script.py
2. **頻繁にパラメータを変えるプログラム:**
   起動時引数で変えられる。引数なしの場合は初期値で実行する。
   実行時の最初 あるいは 最後に usage を表示するので引数なしで実行したのち、引数を変えて実行する。
   python script.py
   python script.py 1.0 2.0
3. **引数が複雑になった場合 I: 計算時間がかからないプログラム**
   実行時の最初 あるいは 最後に usage を表示するので引数なしで実行したのち、引数を変えて実行する。
   usage の表示に、実行例 ex: を表示するので、ex の文をコピペして実行してみるとわかりやすい。
4. **引数が複雑になった場合 I: 計算時間がかかるプログラム**
   引数なしで実行すると、usage と ex だけ表示して終了。
   ex の文をコピペして実行して動作確認したのち、引数の値を変えて実行

神谷作製 pythonプログラムの構造

極最近のプログラムの実行順は以下の通り (regular_solution.py 参照)

1. **モジュールの取り込み**
2. **プログラムの説明:** """ ~ """ コメント文で、ぷグラムの説明を記述
3. **一般的な大域変数の定義:** 物理定数など
4. **プログラム固有の大域変数の初期値の設定**
5. **matplotlib で描画するグラフのパラメータ**
6. **プログラム間で共通に使う小さい関数の定義:**
   pfloat, pint, getarg, getfloatarg, getintarg
   usage, terminate, updatevars
   terminate() は共通の終了処理をする: usage() を表示してから exit() など。
7. **起動時引数で初期値の更新**
   get(float/int)arg() を使うことで、引数が与えられていない場合は初期値を使う ということを1行で実行できる
   次の文をコメントアウトすると、引数がない場合には usage() を表示して実行を終了する。
   ```
   if len(argv) == 1:
       terminate()
   ```
8. **その他、プログラム固有の関数の定義**
9. **main() 関数の定義**
10. **直接スクリプトが実行されたときのみ、main() 関数を実行する**
    ```
    if __name__ == '__main__':
        main()
    ```

Pythonのお約束: __name__ 変数

直接スクリプトが実行されたときのみ、main() 関数を実行する

Pythonでは、スクリプトファイルが実行されるとき、__name__ 変数が設定される。
- **スクリプトファイルがモジュールとして読み込まれる場合: モジュール名**
- **スクリプトファイルが直接実行される場合: '__main__'**

__name__変数の内容により、スクリプトが直接実行された場合とモジュールとして読み込まれた場合で、動作を変えることができる。

```
def main():
    print("executed directy")

if __name__ == '__main__':
    main()
```

とすると、モジュールとして読み込まれたときには main() は実行されないが
直接実行された場合は main() が実行される。

モジュールを開発している際に、モジュール単体を実行して動作確認をする場合などに
使う。

Pythonのエラー処理: try ~ except

Pythonのエラー処理はかなり厳しい

以下のように、頻繁に発生するエラーのすべてでプログラムの実行が終了する

- int(str) で文字列を整数型に変換しようとしたときに、文字列の書式が整数でなかった場合
- リストの要素を a[i] で取得しようとしたときに、i が a[] の要素範囲外だった時
- 未定義値の変数 (None) や変換できない型を含む演算
  (str => floatなどの変換は自動的には実行されない)

参考: perl は、程度の低いエラーには寛容

- int(str) での文字列書式エラー: できる限り整数型に変換する。だめな時は 0 を返す
- リストの範囲外の要素にアクセスした場合: undef (未定義値) を返す
- undef や未定義の変数を含む演算: 未定義値は 0 として演算を実行
- 可能な限り、型の変換は自動的に行われる ("3.0" + 1 は 4.0になる)

Pythonでエラーが起こったときにプログラムを停止しないためには、try ~ except 構文を使う

まず try ブロックを実行し、エラー (例外、exception) が発生したら、exceptブロックを実行する

```
def pfloat(str):
try:
# float(str) を実行してみる
# str が浮動小数点として変換できる文字列であれば、変換した値を戻す
# str が浮動小数点として変換できない場合、例外を発生して except ブロックを実行する
        return float(str)
    except:
# str が浮動小数点として変換できない場合、None (未定義値) を返す
       return None
```

getarg() 関数

Pythonの問題: 起動時引数を取得する sys.arg リスト変数は、範囲外の index を渡すとエラーになってプログラムが終了する
Perlの場合 : リストに範囲外のindexを渡しても、undef が返ってくる。実行は継続。

引数で初期値を更新する関数 getarg() では、どのようなことが想定されるか

1. 何番目の引数か、**index (position) を渡し**、その値を return で返す
2. 引数が与えられない場合 ＝＞ 初期値をそのまま使う
   引数のindexの他、**初期値も渡す必要**がある
3. 引数が与えられた場合 ＝＞ 初期値を引数で置き換える
4. 初期値を与えない場合は None を返す
   初期値を与えないで getarg(position) を呼び出せるようにし、この際には初期値を Noneにする。
   “デフォルト引数” 機能を使う

```
# getarg()の引数としては、position と defval を渡す
# defval はデフォルト引数となっているので、
# getarg(position) と呼び出されたら、defvalに は Noneが代入される
def getarg(position, defval = None):
    try:
# sys.argv[position] が存在したら sys.argv[position] を返す
        return sys.argv[position]
    except:
# sys.argv[position] が存在しなかったら、defval を返す
        return defval
```

Pythonと他言語の比較: モジュール

Python	C	Perlなど
基本的な機能とモジュールの読み込み ・print() などは 組み込み関数 ・ほとんどの機能はモジュールを読み込むことによって使う **import numpy as np** numpyモジュールを読み込み、npという変数名でアクセスできるようにする **from math import log, exp** mathモジュールの中の関数のうち、log と exp 関数のみをインポートする **from matplotlib import pyplot as plt** matplotlibモジュールのpyplotモジュールをインポートし、pltという変数名でアクセスできるようにする	・低レベル関数のみ組み込み関数 ・printf()などの入出力関数なども組み込まれていない ・他はライブラィをリンクする。 ・ソースコードでは対応するヘッダーファイルを読み込み、変数、関数の型宣言を取り込む **#include <stdio>**	・print() などは 組み込み関数 ・ほとんどの機能はモジュールを読み込むことによって使う **use strict;** **require strict.pm;** ・モジュールを無効化することもできる **no strict;**
モジュール、ライブラリィファイル ・実行スクリプトとモジュールファイルには区別はない。一般に、どちらも拡張子 .py をつける ・モジュールに main() 関数がある場合、 if __name__ == '__main__': main() テクニックを使い、直接実行時のみ main() を実行するようにする	・ライブラリィのソースコードには main() 関数はあってはいけない ・ライブラリィファイルの拡張子も一般的に .c, .cpp などを使う ・ライブラリィはコンパイル後にリンクして実行可能ファイルを作製	・モジュールファイルの拡張子は一般的に .pm を使う

よく使うpythonモジュール

システム関係など

- sys
  sys.argvで起動時引数を取得
- csv
  CSVファイルの読み書き

科学計算関係

- math
  基本的な数学関数。sin, cos, tan, asin, log, exp　など
- numpy
  配列、行列を含む数値計算などにはほぼ標準。
  Python標準のリストより、numpy.ndarray を使う方がいい。
  numpy.ndarray からリストへは、ほとんど場合に自動的に型変換してくれる
  線形代数関数 (逆行列、一次連立方程式の解など)
- scipy
  numpyの機能に加え、信号処理 (FFT) など多様な数学関数がある

グラフ関係

- matplotlib
  グラフの描画

matplotlibでのグラフ表示

参考: http://conf.msl.titech.ac.jp/Lecture/python/matplotlib.html

```
fig = plt.figure(figsize = (8, 8))    # グラフサイズを指定して fig変数を取得
ax1 = fig.add_subplot(1, 1, 1)        # あとで複数のグラフ枠の描画もできるように、
                                        add_subplotを使い、最初のグラフ枠の変数を取得
ax1.plot(xarray, yarray)              # xarray, yarrayをデータの組とするグラフを描画する
```

一般的な場合

- matplotlibでは通常、グラフに表示するデータのリスト変数を plt.plot() で設定したのち、**plt.show()** でグラフを表示します。
- この場合、グラフウインドウを閉じるまで、プログラムが停止される。プログラムを終了するには、グラフウインドウを閉じる必要がある。

神谷の場合

- plt.pause(0.1) を使うと、グラフを最新のデータで表示したのち、プログラムの実行を継続する。0.1 は、pause() 関数を実行している際のsleep時間。なるべく短い時間に設定
- 放っておくと、プログラムが勝手に終了し、プログラムウインドウも閉じてしまうので、input() でプログラムが終了するのを止め、グラフを表示し続ける。

```
plt.pause(0.1)
print("Press ENTER to exit>>", end = '')
input()
```

- コンソールでENTERを押せばプログラムを終了できる
- plt.pause() を使うと、グラフをリアルタイムでupdateするプログラムも作れる

方程式の解法

課題

$f(x) = \exp(x) + x = 0$ の解を二分法を使って解け。

Excelなどを使ってもいいし、pythonなどのプログラムを作ってもよい。

余力があれば、Newton-Raphson法でも解いてみるといい。

参考:
http://conf.msl.titech.ac.jp/Lecture/

- 計算材料学特論 資料

PowerPointのプレゼンテーションファイルにして提出
期限: 今日の17:00までに
できたところまでで可

収束計算の収束判定

- **F(x) = 0 となる x を求めよ**

  case F: 収束判定条件を eps より小さくなるとして、

  $\mathrm{abs}(F(x_{i+1})) < \mathrm{eps}$

  とすると、dF/dx が 1 より非常に小さい場合は x の精度が悪くなる。

  eps = 10^{-6}として、$F(x) = 10^{-10}(\exp(x) + x)$ を考えてみよう。

  case x: $\mathrm{abs}(x_{i+1} - x_i) < \mathrm{eps}$

  とすると、dF/dx が 1 より非常に大きい場合は F(x) の精度が悪くなる。

  eps = 10^{-6}として、$F(x) = 10^{10}(\exp(x) + x)$ を考えてみよう。

  ※ 目的による。xの精度だけが必要なら、F(x)での収束判定はしなくてもいい

  良い方法: 2つのeps、xeps と yepsを用いる。

  $F(x) = 10^{-10}(\exp(x) + x)$

  xeps = 1.0e-6

  yeps = 1.0e-16

  $\mathrm{abs}(x_{i+1} - x_i) < \mathrm{xeps}$ and $\mathrm{abs}(F(x_{i+1})) < \mathrm{yeps}$

- **F(x) を最小 (最大) にする x を求めよ**

  変化量で収束判定をする

  $\mathrm{abs}(x_{i+1} - x_i) < \mathrm{xeps}$ and $\mathrm{abs}(F(x_{i+1}) - F(x_i)) < \mathrm{yeps}$

収束計算における最終精度の判定

収束計算において、最終的な解の精度を正確に知ることは難しい

繰り返し計算において、$x_0 \Rightarrow x_1 \Rightarrow x_2 \Rightarrow \cdots \Rightarrow x_{n-1} \Rightarrow x_n$ と、パラメータが順次変化していくため、たとえば $x_n - x_{n-1} = 1.0e\text{-}6$ になったとしても、そのあと10サイクル同様の変化が続けば、x_{n+10}は x_{n-1} より 2.0e-6 変化することになる。

=> 一般の収束計算 (Newton-Raphson法など) での最終解の精度は、あくまでも「収束精度」

x	F(x)
-0.567143226	1.0145674E-07
-0.567143357	-1.0404311E-07
-0.567143291	-1.2931852E-09

の結果からは、**収束解は -0.567143 と言える**。

一部のアルゴリズムでは、解の精度を正確に判定できる

二分法: 解 x は初期値区間 [xmin, xmax] 内に存在する。

一回の計算で範囲を ½ にする

＝＞ n回の繰り返しで範囲は（xmax – xmin) / 2^n に狭まる。これが最終解の精度

※ 二分法の場合、繰り返し回数で精度が確定するので、収束判定条件は必要ではない

単調関数方程式の解法: 二分(Bisection)法

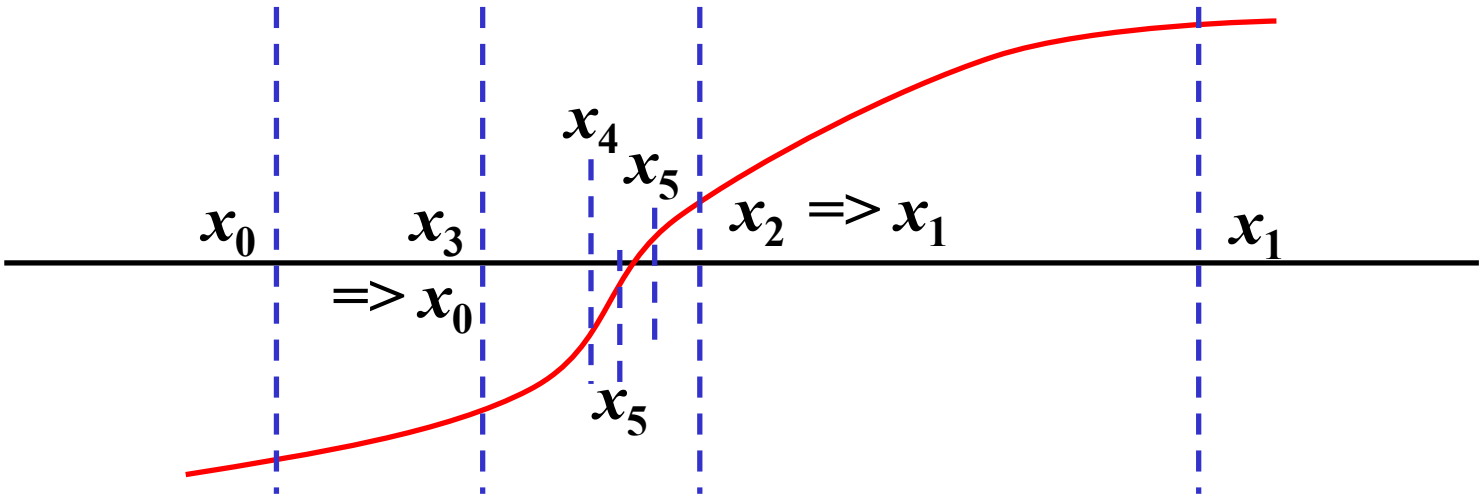
単調関数 $f(x) = 0$ の解

$f(x_0) < 0$ かつ $f(x_1) > 0$ (あるいは $f(x_0) > 0$ かつ $f(x_1) < 0$) の区間 $[x_0, x_1]$から反復的に解く

$f(x)$ が単調関数であれば、解 x はかならず $x_0 < x < x_1$

$f(x_0) < 0$ かつ $f(x_1) > 0$の場合を考える ($f(x_0) \cdot f(x_1) < 0$ で判断)。

1. $x_2 = (x_0 + x_1) / 2.0$
2. $f(x_2) > 0$ ($f(x_0) \cdot f(x_2) < 0$) であれば、x_1 をx_2 で置き換える
   $f(x_2) < 0$ ($f(x_1) \cdot f(x_2) < 0$) であれば、x_0 をx_2 で置き換える
3. $|x_1 - x_0|$, $|f(x_1) - f(x_0)|$ が必要な精度以下になったら、
   解を x_2 にして反復終了
4. 1. に戻って反復



Newton-Raphson法 (Newton法)

$f(x) = 0$の解を求める

$$f(x_0+dx) = f(x_0) + dx\, f'(x_0) \sim 0$$
$$\Rightarrow \quad x_1 = x_0 + dx = x_0 - f(x_0) / f'(x_0)$$

計算では $f'(x_0)$ を差分計算で置き換えられる

割線法 (セカント法、はさみうち法):

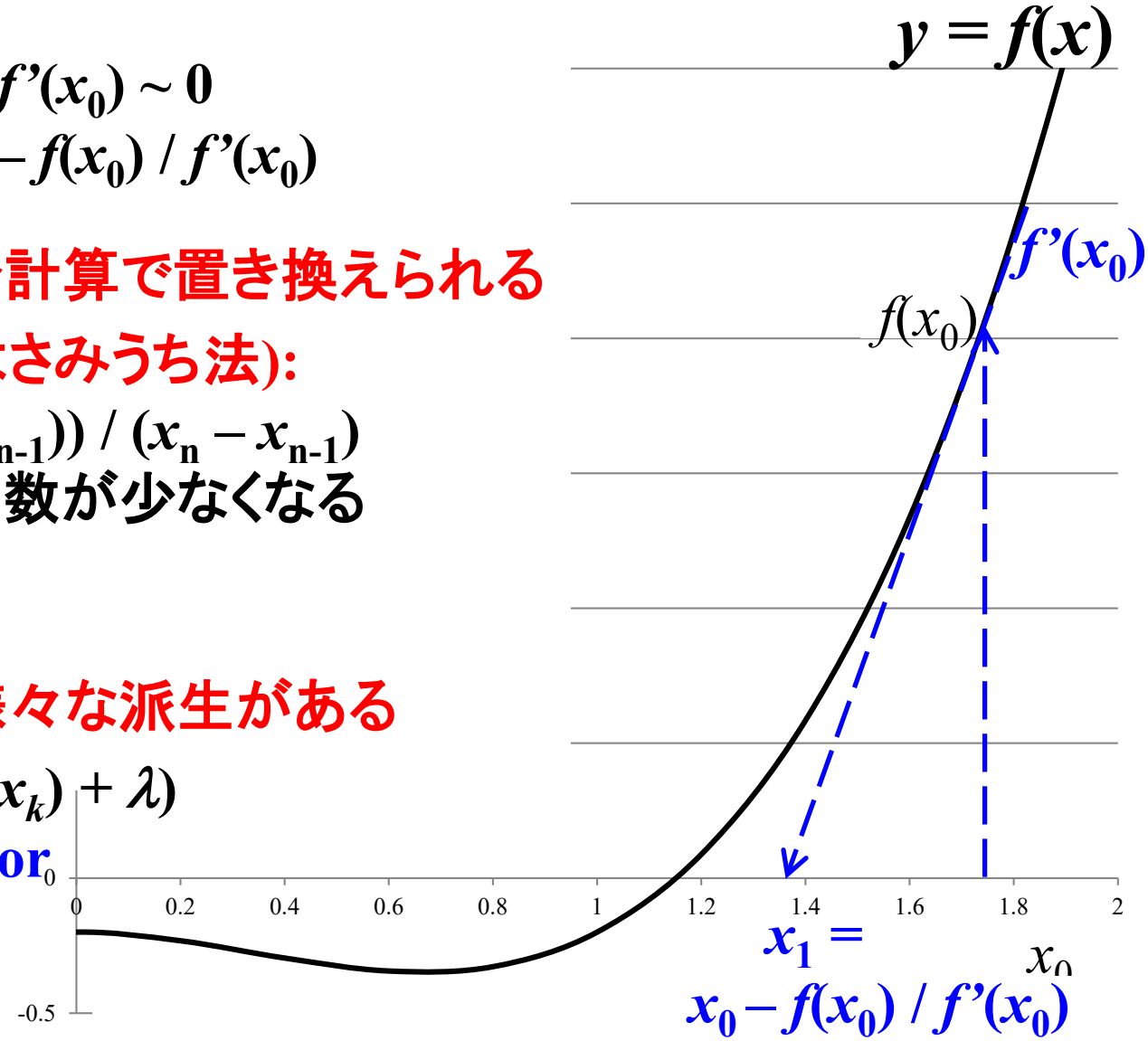
$$f'(x) = (f(x_n) - f(x_{n-1})) / (x_n - x_{n-1})$$

を使う。$f(x)$の計算回数が少なくなる

発散を抑える工夫で様々な派生がある

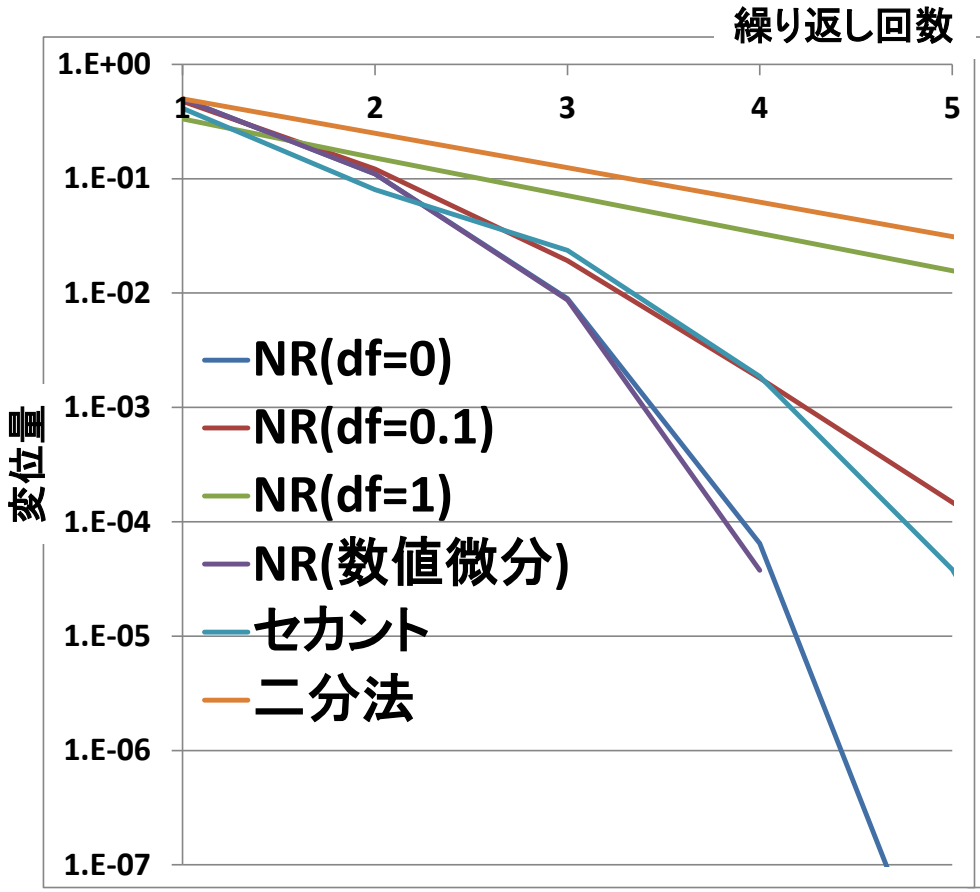
$$x_{k+1} = x_k - f(x_k) / (f'(x_k) + \lambda)$$

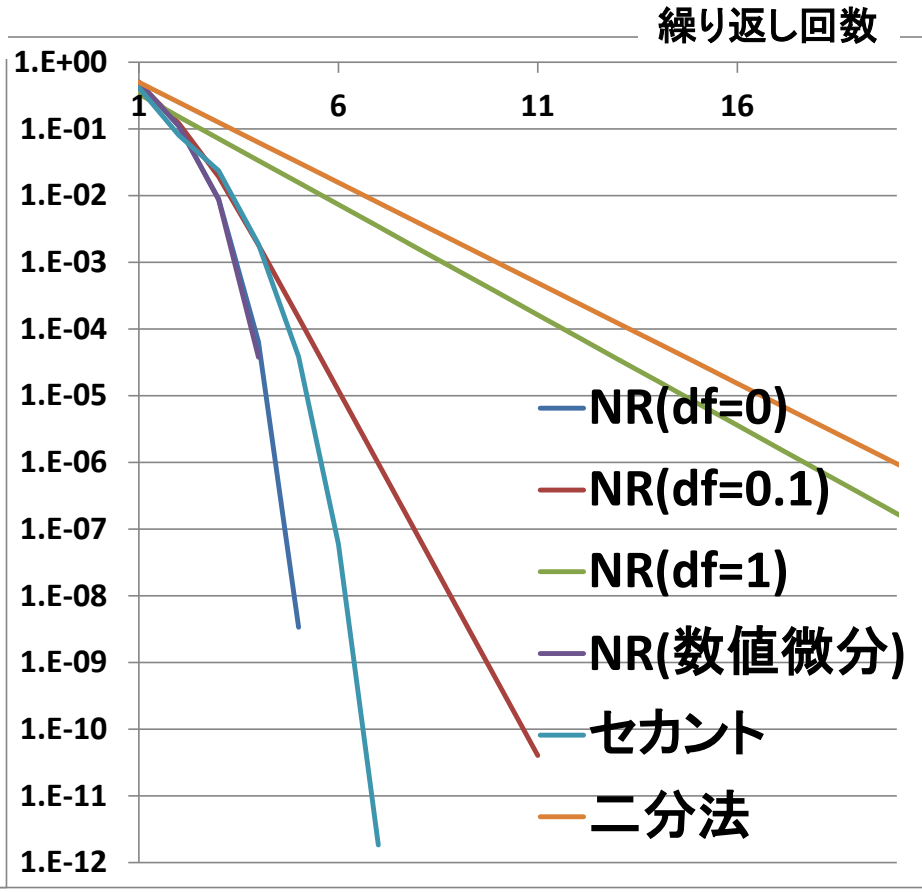
λ: Dumping Factor



収束過程の比較

$f(x) = \exp(x) - 3x = 0$ (初期値 $x = 0$)　　精確値 0.619061





NR: Newton-Raphson法
df: Dumping Factor

最小二乗法: 磁気抵抗効果等

課題

MRxx.csv (MRxx.xlsx) のデータ (磁場 B ー 抵抗率 ρ) について、

$$\rho(B) = \rho_0 + aB^2$$

を仮定し、定数 ρ_0 と a を求めよ。マニュアルフィッティングしてもよいし、pythonプログラム lsq1.py や csvplot.csv などを参考に線形最少二乗法などで求めてもよい。

参考:

http://conf.msl.titech.ac.jp/Lecture/

- 計算材料学特論 資料

PowerPoint 等 のプレゼンテーションファイルにして提出

期限: 今日の17:00までに

できたところまでで可

最少二乗のパターン

- Excelでマニュアルフィッティング
- Excelのソルバーでフィッティング
  うまくいかなかった => 初期値の設定
- pythonで一次多項式 $f(x^2) = a + b\,(x^2)$ で線形最少二乗
- pythonのライブラリィを使ってフィッティング
- gnuplotの関数フィッティング

解答

線形最小二乗法: 収束誤差はない

- $\rho(B) = c_0 + c_1 x + c_2 B^2$ に対して線形最小二乗を使う

  $\rho(B) = 0.003077578181 + 1.16241 \times 10^{-9} B + 3.519853 \times 10^{-7} B^2$

- $\rho(B) = c_0 + c_1 B^2$ に対して線形最小二乗法を使う

  $\rho(B) = 0.003077578505 + 3.51966527 \times 10^{-7} B^2$

- python の numpy.polyfit() を使う

非線形最小二乗法: 初期値依存、収束誤差

- Excelで関数$\boldsymbol{y = ax^2 + b}$によるフィッティング
- python の scipy.optimize.curve_fit() を使う

  初期値を与えない場合は、1.0 を初期値とする

  $\rho(B) = 0.003078 + 3.520 \times 10^{-7} B^2$

- gnuplot の 関数フィッティング

  $\rho(B) = 0.00307758 + 3.51965 \times 10^{-7} B^2$

マニュアルフィッティング: 人間依存、誤差不明

$\rho(B) = 0.0030775 + 3.5 \times 10^{-7} B^2$

移動度の計算

$$\rho(B) = \rho_0 \left(1 + \frac{a}{\rho_0} B^2\right) = \rho_0 \left(1 + \left(\frac{e\tau}{m_e^*}\right)^2 B^2\right)$$

$\rho(B) = 0.00308 + 3.52\text{x}10^{-7}\ B^2 = 0.00308(1.0 + 0.107^2\ B^2)$ (単位はMKS)

※ $\mu = \frac{e\tau}{m_e^*}\sqrt{\frac{a}{\rho_0}} = 0.0107\ \text{m}^2/(\text{Vs}) = 107\ \text{cm}^2/(\text{Vs})$

自作の多項式線形最小二乗法: mlsq()

```
def mlsq(x, y, m, *, iPrint = 0):
    n = len(x)
# 配列をnp.ndarrayで宣言
# Siは一次元ベクトルだが、numpyの
# 行列演算のために2次元配列で宣言
    Si  = np.empty([m+1, 1])
    Sij = np.empty([m+1, m+1])

    for l in range(0, m+1):
# Si[l] = Σy_i x_i^l
        v = 0.0
        for i in range(0, n):
            v += y[i] * pow(x[i], l)
        Si[l, 0] = v

    for j in range(0, m+1):
        for l in range(j, m+1):
# Sij[j, l] = Σx_i^(j+l)
            v = 0.0
            for i in range(0, n):
                v += pow(x[i], j+l)
            Sij[j, l] = Sij[l, j] = v

# デバッグのため、Si, Sijを出力する
# オプションを用意
if iPrint == 1:
        print("Vector and Matrix:")
        print("Si=")
        pprint(Si)
        print("Sij=")
        pprint(Sij)
        print("")

# LAPACKのpythonライブラリィ linalgを使う
# ここでは .inv() 逆行列を求めているが、
# ci = np.linalg.solve(Sij, Si)
# で一次連立方程式を直接解く方がいい

    ci = np.linalg.inv(Sij) @ Si
# ci は 二次元のndarray()で返ってくる
# .transpose() で転置行列を取ったのち、
# .tolist() でリスト型に変換
    ci = ci.transpose().tolist()
# ci は 1 × (m+1) の二次元のリストに
# なっているので、ci[0]を取り出して
# 一次元リストを返す
    return ci[0]
```

CSVファイルの構造

CSV: Comma Separated Values

- 一行に複数のテキストデータを “,” で区切って並べたテキストファイル
  一行ごとに文字列変数に読み込み、文字列型の .split(“,”) メソッドで分割
- データ内に “,” がある場合は、テキストデータを “” でくくる
- データ内に “ がある場合のフォーマットは複数

  複雑な正規表現か専用の関数を使う必要

＊すべてのCSVファイルに対応するのは意外と大変

=> csv モジュールを使う

- 読み込んだテキストデータを浮動小数点に変換
- 数値データが浮動小数点に変換できない場合に対応する

  Fortran形式の倍精度実数 5.0D+5
  余計なセルがあるExcelファイルを CSV に変換 (非数値文字列)
  余計な空行があるExcelファイルを CSV に変換 (空文字列)
  Unicode以外のCSVファイル

CSVファイルの読み込み

```
import csv                          # csv モジュール読み込み

def read_csv(fname):
    x = []
    y = []
    with open(fname) as f:                  # ファイル fname を読み込みモードで開く
        fin = csv.reader(f)                 # csvクラスの readerオブジェクトを取得
        xlabel, ylabel, = next(fin)         # 最初の1行はラベルなので、next()関数で読み込む
        for row in fin:                     #  残りの数値データを読みこむ
            try:
                x.append(float(row[0]))     # float() で浮動少数点型に変換してリストに追加
                y.append(float(row[1]))
            except:                         # 数値データが浮動小数点に変換できなかったら
                                            # Warningを出すが、プログラムは停止しない
                                            # 該当する行のデータは捨てる
                print("Warning: Invalid float data [{}] or [{}]".format(row[0], row[1]))
    return x, y

x, y = read_csv(infile)
```

*l*LSQ: General functions

線形最小二乗法: 一般関数の場合

$$f(x)=\sum_{k=1}^{n} a_k f_k(x) \qquad S=\sum_{i=1}^{N}\left(y_i-\sum_{k=1}^{n} a_k f_k(x_i)\right)^2$$

$$\frac{dS}{da_l}=-2\sum_{i=1}^{N} f_l(x_i)\left(y_i-\sum_{k=1}^{n} a_k f_k(x_i)\right)=0$$

$$\begin{pmatrix} \sum f_1(x_i)f_1(x_i) & \sum f_1(x_i)f_2(x_i) & \sum f_1(x_i)f_3(x_i) & \cdots & \sum f_1(x_i)f_N(x_i) \\ \sum f_2(x_i)f_1(x_i) & \sum f_2(x_i)f_2(x_i) & \sum f_2(x_i)f_3(x_i) & & \sum f_2(x_i)f_N(x_i) \\ \sum f_3(x_i)f_1(x_i) & \sum f_3(x_i)f_2(x_i) & \sum f_3(x_i)f_3(x_i) & & \sum f_3(x_i)f_N(x_i) \\ \vdots & & & \ddots & \\ \sum f_N(x_i)f_1(x_i) & \sum f_N(x_i)f_2(x_i) & \sum f_N(x_i)f_3(x_i) & & \sum f_N(x_i)f_N(x_i) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} \sum y_i f_1(x_i) \\ \sum y_i f_2(x_i) \\ \sum y_i f_3(x_i) \\ \vdots \\ \sum y_i f_N(x_i) \end{pmatrix}$$

If *f(x)* is *linear with respect to fitting parameters,*
final solution is obtained by one matrix operation

係数に関して線形であれば、1度の行列計算で最終解が得られる

ex. $f(x)=a+b\log x+c/x$

$$f(x,y)=a+bxy+cy/x$$

関数の解: Newton-Raphson法 (Newton法)

$f(x) = 0$の解を求める

$$f(x_0+dx) = f(x_0) + dx\, f'(x_0) \sim 0$$
$$\Rightarrow \quad x_1 = x_0 + dx = x_0 - f(x_0) / f'(x_0)$$

計算では $f'(x_0)$ を差分計算で置き換えられる

割線法 (セカント法、はさみうち法):

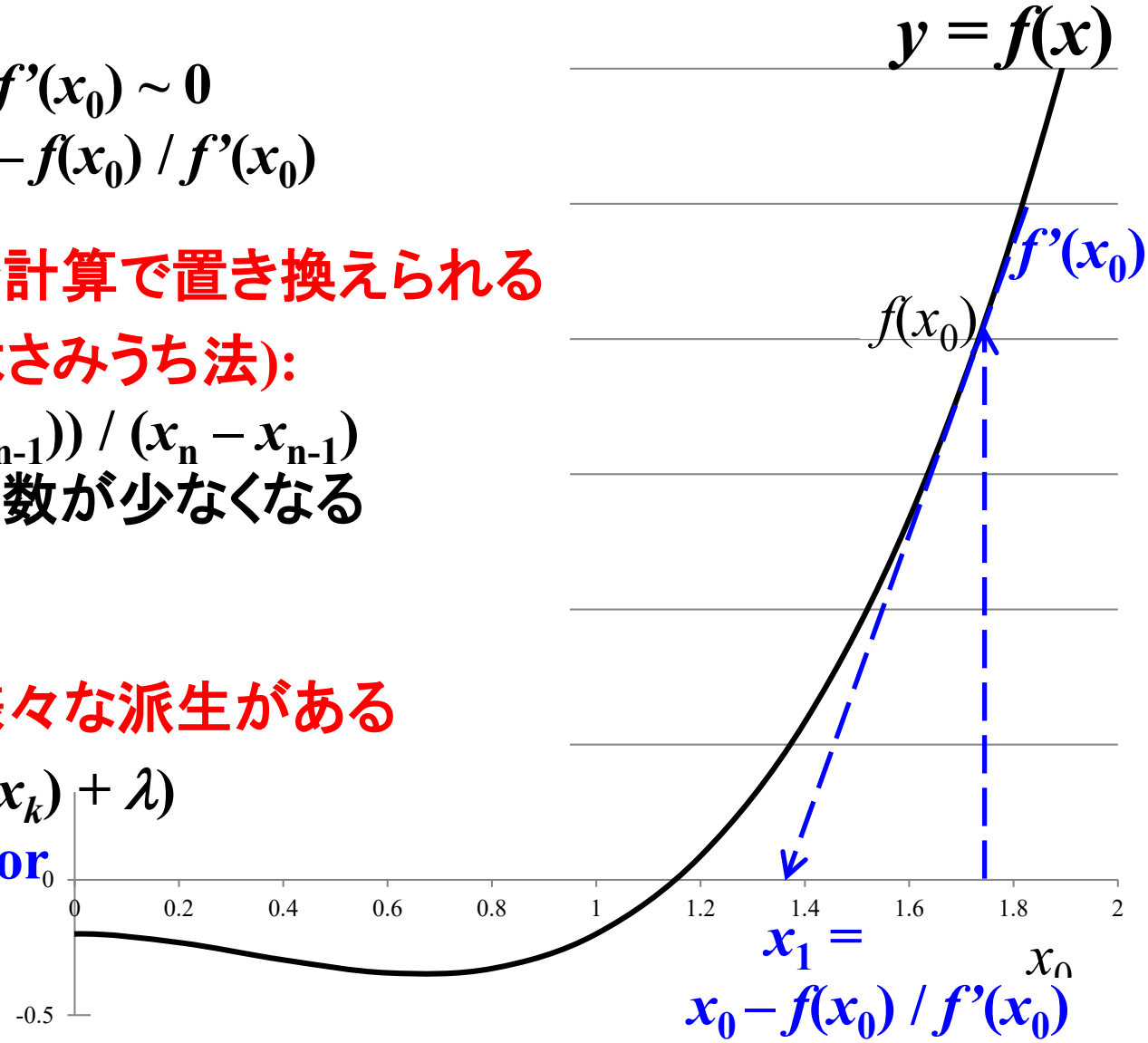
$$f'(x) = (f(x_n) - f(x_{n-1})) / (x_n - x_{n-1})$$

を使う。$f(x)$の計算回数が少なくなる

発散を抑える工夫で様々な派生がある

$$x_{k+1} = x_k - f(x_k) / (f'(x_k) + \lambda)$$

λ: Dumping Factor

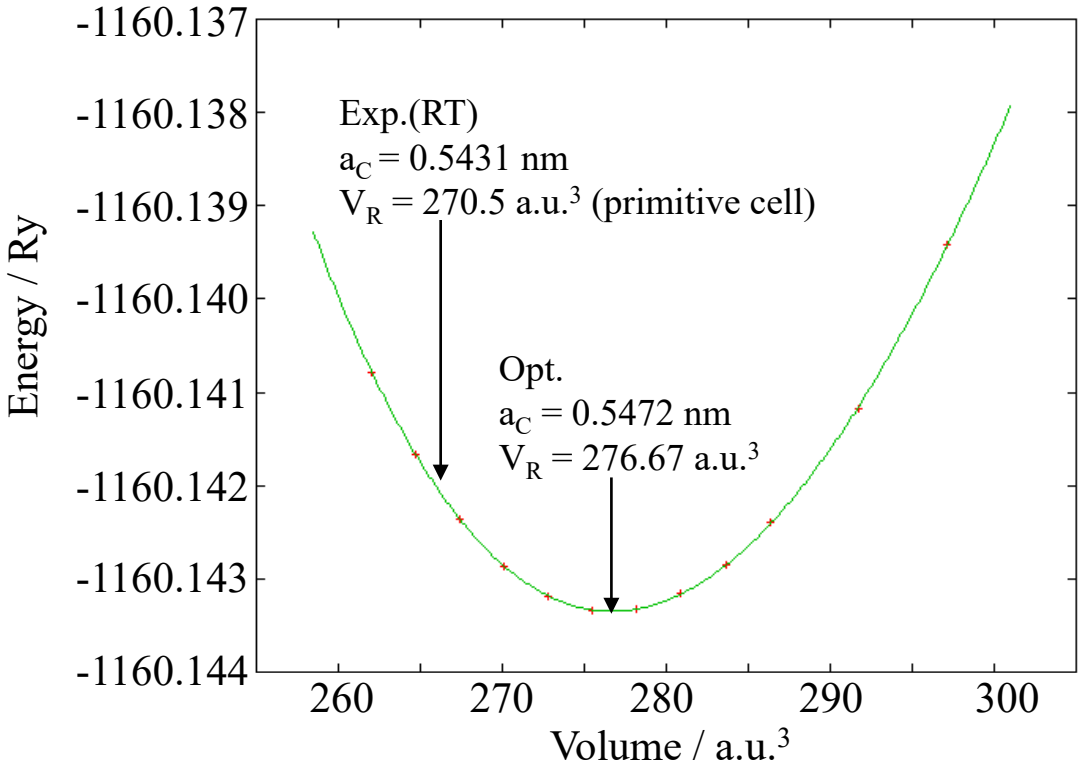


NL optimization of crystal structure: Illustrative approach

安定構造: 図解による解法

Calculate total energy by quantum calculations by varying a lattice parameter

***ex.* Si**

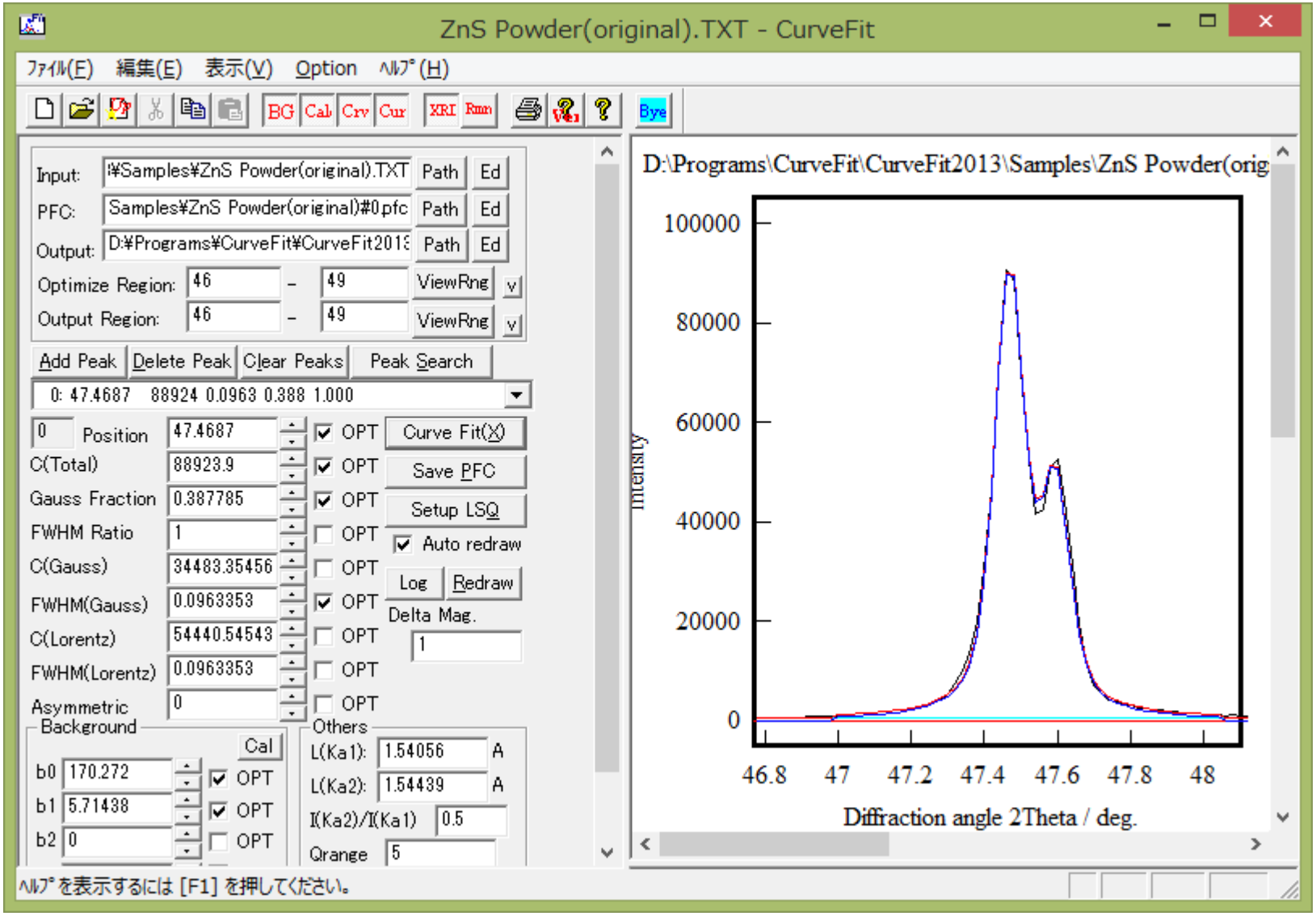


$$E = E_{\min} + 1/2 B_0 \left(V / V_0\right)^2$$

B_0 (GPa) = 87.57 GPa (exp: 97.88 GPa)

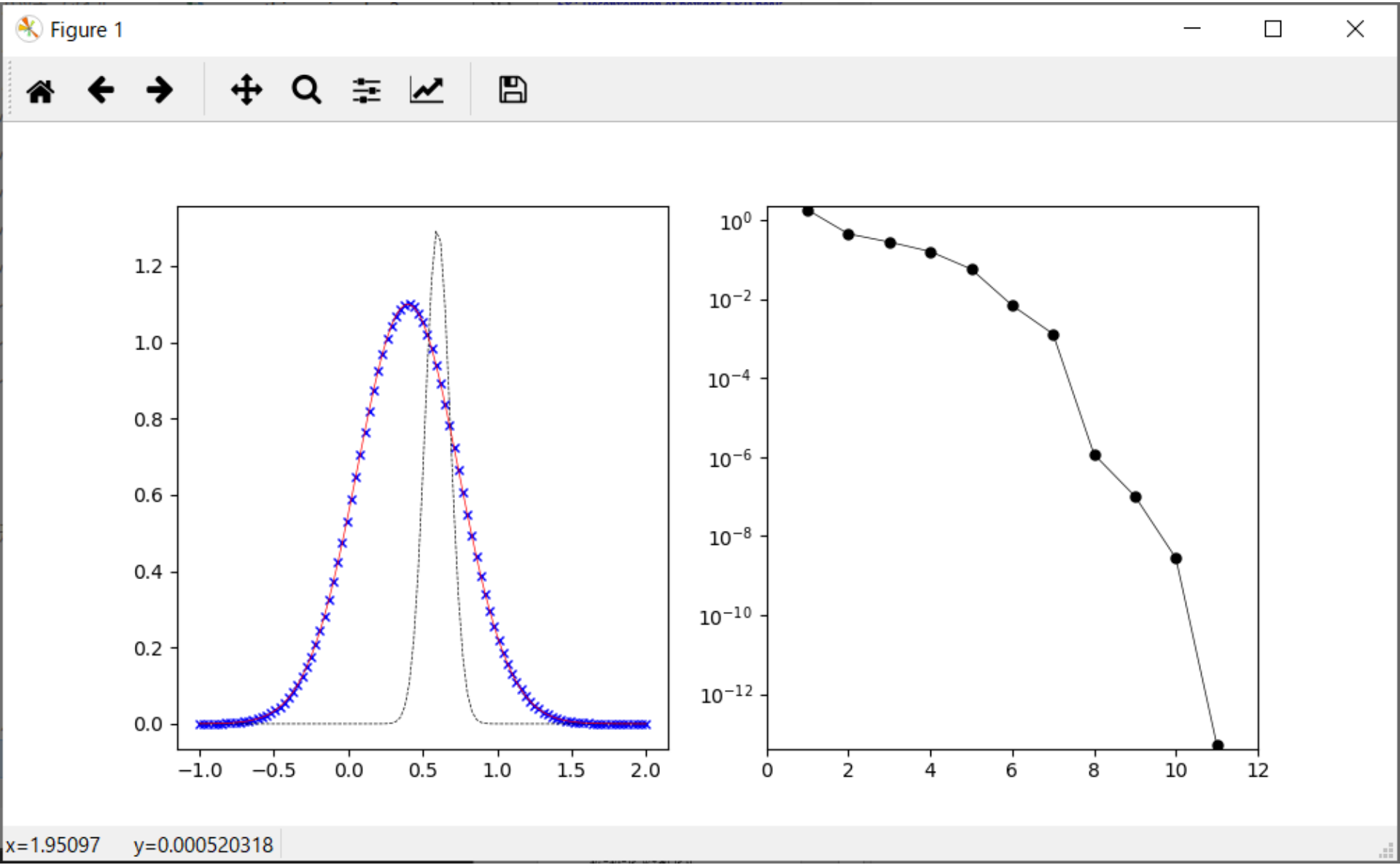
多変数最適化問題例: 粉末XRDのピーク分離

$K\alpha_1$と$K\alpha_2$線が 強度 2:1 で出現することを考慮



Ex.: Deconvolution of powder XRD peak

peakfit-scipy-minimize.py



分光解析に使われるプロファイルモデル

Lorentz関数

$$I_L(x) = \frac{1}{1+\left[(x-x_0)/w\right]^2}$$

w: 半値半幅

Gauss関数

$$I_G(x) = \frac{1}{a_w w \pi^{1/2}} \exp\left\{-\left[(x-x_0)/(a_w w)\right]^2\right\}$$

$$a_w = (\ln 2)^{-1/2} = 0.832554611$$

Voigt関数:

Lotentz型の固有スペクトルに
他の要因のGauss型の広がりが重なる
畳み込み積分 (Convolution) であらわされる

$$I_V(x) = \int_{-\infty}^{\infty} I_G(x') I_L(x-x') dx'$$

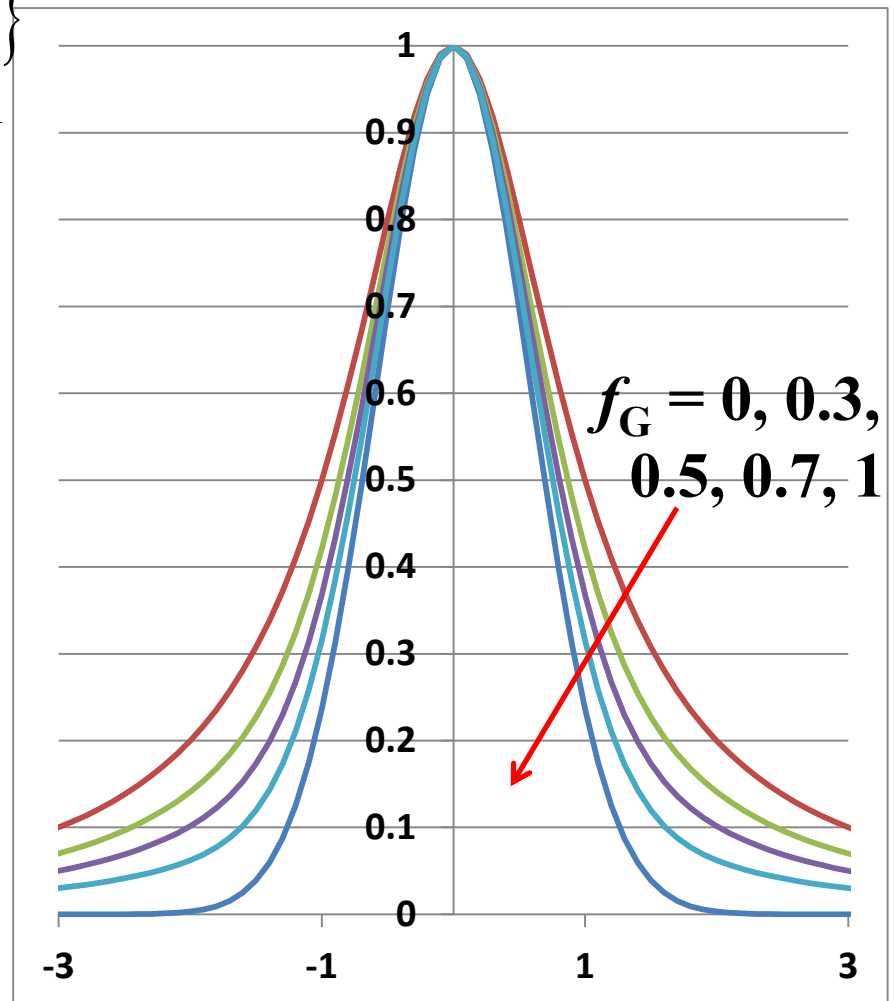
$$= \frac{a_V}{\pi} \int_{-\infty}^{\infty} \frac{\exp(-x'^2)}{a_V{}^2 + (x-x')^2} dx'$$

Pseudo-Voigt関数:

Voigt関数の簡略版

$$I_{PV}(x) = f_G I_G(x) + (1-f_G) I_L(x)$$

f_G: Gauss関数分率



多変数関数の最適化: Newton-Raphson法

多変数への拡張: 最小化関数 $F(x_l)$ の最小値を求める

$$f_k(x_l) = \partial F(x_l)/\partial x_k = 0$$

繰り返し計算: $f_k(x_l + \delta x_l) \sim f_k(x_l) + \sum_{k'} \delta x_{k'} \partial f_k(x_l)/\partial x_{k'} = 0$

$$x_{l,1} = x_{l,0} - (\partial f_k(x_l)/\partial x_{k'})^{-1}(f_k) = x_{l,0} - (F''_{kk'})^{-1}(F'_k)$$

$$F''_{kk'} = \frac{\partial^2 F(\mathbf{x})}{\partial x_k \partial x_{k'}}$$

Hessian (ヘッセ) 行列

(ヘッセ行列の固有値をヘッシアンと呼ぶ)

Hessian行列は正定値であるとは限らない (極大値、鞍点)

＝＞ F''が降下方向を与えるとは限らない

F''を正定値行列で置き換え、発散を抑える

$$x_{l,1} = x_{l,0} - (F''_{kk'} + \lambda I)^{-1}(F'_k)$$

λ: Dumping Factor

Quasi-Newton method (準Newton法)

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

最小化関数 $F(x_l)$

繰り返し計算: $x_l{}^{(i+1)} = x_l{}^{(i)} - \left(\partial^2 F/\partial x_k \partial x_{k'}\right)^{-1}(\partial F/\partial x_k)$

$F''_{kk'} = \partial^2 F/\partial x_k \partial x_{k'}$: **Hessian (ヘッセ) 行列**

Newton法の問題:

(1) Hessian行列は二次行列のため、計算に時間がかかる

(2) Hessian行列の固有値は負になることもある => 極大値を探索

(3) 発散しやすい

準Newton法:

(1,2) Hessian行列の計算を過去の一次微分の値を使って近似

(3) 探索方向 $-(\partial^2 F/\partial x_k \partial x_{k'})^{-1}(\partial F/\partial x_k)$ に沿って線形探索を行う

Davidon-Fletcher-Powell (DFP) 法

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

$$F\left(x_l^{(k)} + \alpha d\right) = F\left(x_l^{(k)}\right) + \nabla F(x_l^{(k)})^T d + \frac{1}{2} d^T B^{(k)} d \sim 0$$

$B^{(k)} d = -\nabla F(x_l^{(k)})$ で探索方向 d を決める

DFP法: 準Newton法のはじまり

$$s^{(k)} = x^{(k+1)} - x^{(k)}, y^{(k)} = \nabla F\left(x_l^{(k+1)}\right) - \nabla F(x_l^{(k)})$$

$$B^{(k+1)} = B^{(k)} + \frac{\left(y^{(k)} - B^{(k)} s^{(k)}\right) \cdot y^{(k)T} + y^{(k)} \cdot \left(y^{(k)} - B^{(k)} s^{(k)}\right)^T}{s^{(k)T} \cdot y^{(k)}}$$

$$- \frac{s^{(k)T} \cdot \left(y^{(k)} - B^{(k)} s^{(k)}\right)}{\left(s^{(k)T} \cdot y^{(k)}\right)^2} y^{(k)} \cdot y^{(k)T}$$

$$= B^{(k)} - \frac{B^{(k)} s^{(k)} \cdot y^{(k)T} + y^{(k)} \cdot \left(B^{(k)} s^{(k)}\right)^T}{s^{(k)T} \cdot y^{(k)}} + \left(1 + \frac{s^{(k)T} B^{(k)} s^{(k)}}{s^{(k)T} \cdot y^{(k)}}\right)$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS) 法

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

BFGS法：準Newton法でも最も有効と認められている

$$s^{(k)} = x^{(k+1)} - x^{(k)}, y^{(k)} = \nabla F\left(x_l{}^{(k+1)}\right) - \nabla F\left(x_l{}^{(k)}\right)$$

$$B^{(k+1)} = B^{(k)} - \frac{B^{(k)}s^{(k)}\left(B^{(k)}s^{(k)}\right)^T}{s^{(k)T}B^{(k)}s^{(k)}} + \frac{y^{(k)}y^{(k)T}}{s^{(k)T}.y^{(k)}}$$

アルゴリズム：

STEP 0: 初期値 $x^{(0)}$、初期行列 $B^{(0)}$ (通常は単位行列) を与える。

STEP 1: $B^{(k)}d = -\nabla F\left(x_l{}^{(k)}\right)$ から探索方向 $d^{(k)}$ を求める

STEP 2: 直線探索によって、$d^{(k)}$方向のステップ幅 $\alpha^{(k)}$ を決める

STEP 3: $x^{(k+1)} = x^{(k)} + \alpha^{(k)}d^{(k)}$ とする

STEP 4: 収束したら計算終了。

収束条件が満たされていなければSTEP 5へ

STEP 5: $s^{(k)}, y^{(k)}$を計算し、$B^{(k+1)}$ を求め、STEP 1へ

Line search (直線探索法): Armijo条件

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

(i) **Armijo** (アルミホ) 条件

$0 < \xi < 1$ であるような定数 ξ に対して,

$$f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) + \xi \alpha \nabla f(\boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{d}_k \tag{4.6}$$

を満たす $\alpha > 0$ を選ぶ.

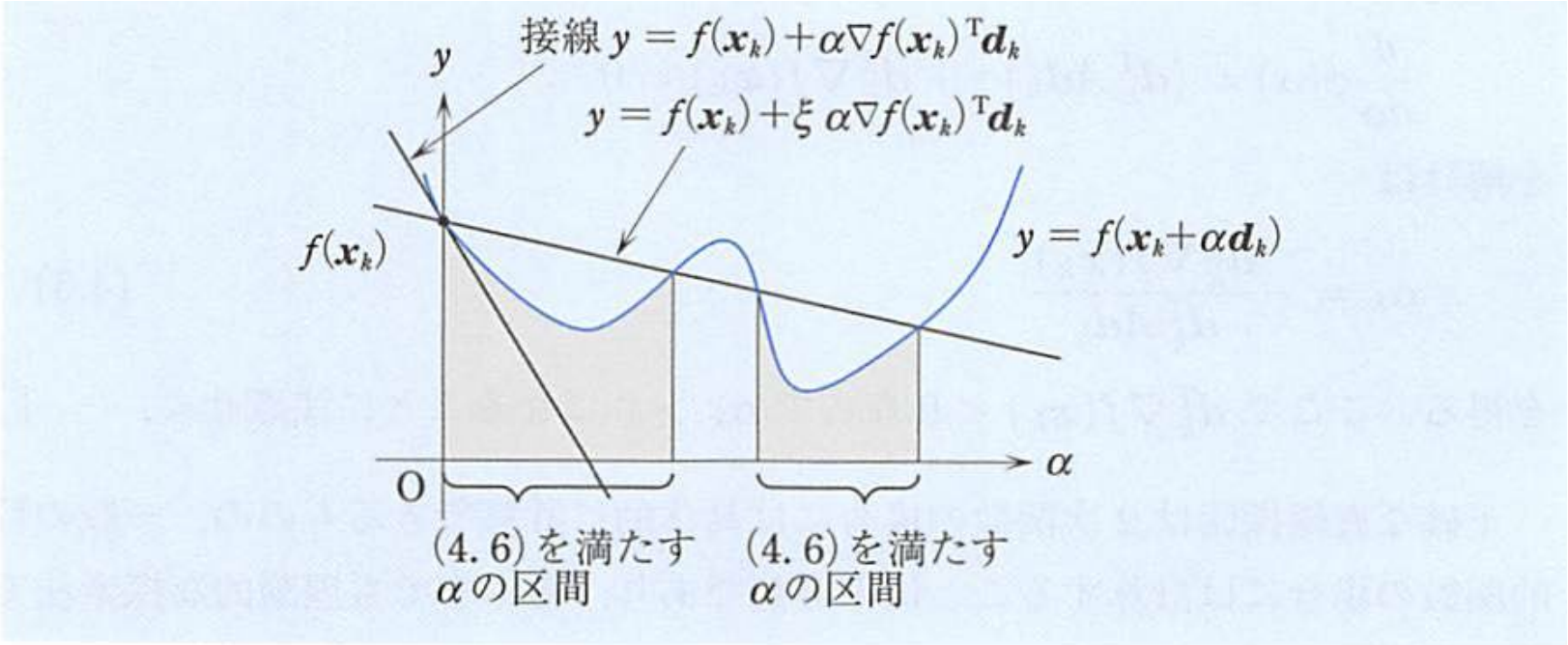


図 4.7　Armijo 条件 (4.6) を満たすステップ幅

Line search (直線探索法): Wolfe条件

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

(ii) **Wolfe**(ウルフ) 条件

$0 < \xi_1 < \xi_2 < 1$ であるような定数 ξ_1, ξ_2 に対して,

$$f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) + \xi_1 \alpha \nabla f(\boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{d}_k \tag{4.7}$$

$$\xi_2 \nabla f(\boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{d}_k \leq \nabla f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k)^{\mathrm{T}} \boldsymbol{d}_k \tag{4.8}$$

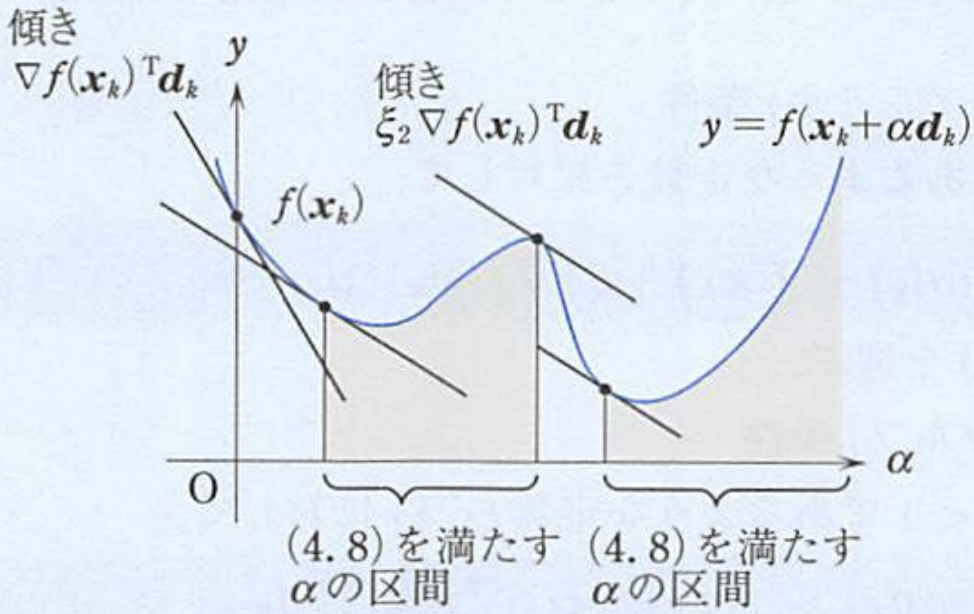


図 4.8　Wolfe 条件 (4.8) を満たすステップ幅

最急降下法 (Steepest Descend) 法

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

残差関数の傾きのみから最小値を探索する。勾配法としては最も単純。

- SD法: ベクトル $-(df / dx_i)dx_i$ 方向へ進めば S^2は小さくなる

  $x_i^{(i+1)} = x_i^{(i)} - \alpha(df / dx_i)$

  α は適当に決める。

右の例:

$S^2 = f(x_i) = 5x_1^2 + x_2^2$, 初期値 $x_1 = 0.7, x_2 = 1.5$

- Newton法:

  楕円問題の場合は一度目の計算で最適値に到達

- SD法:

  α = 0.3: 発散 (グラフにプロットしていない)

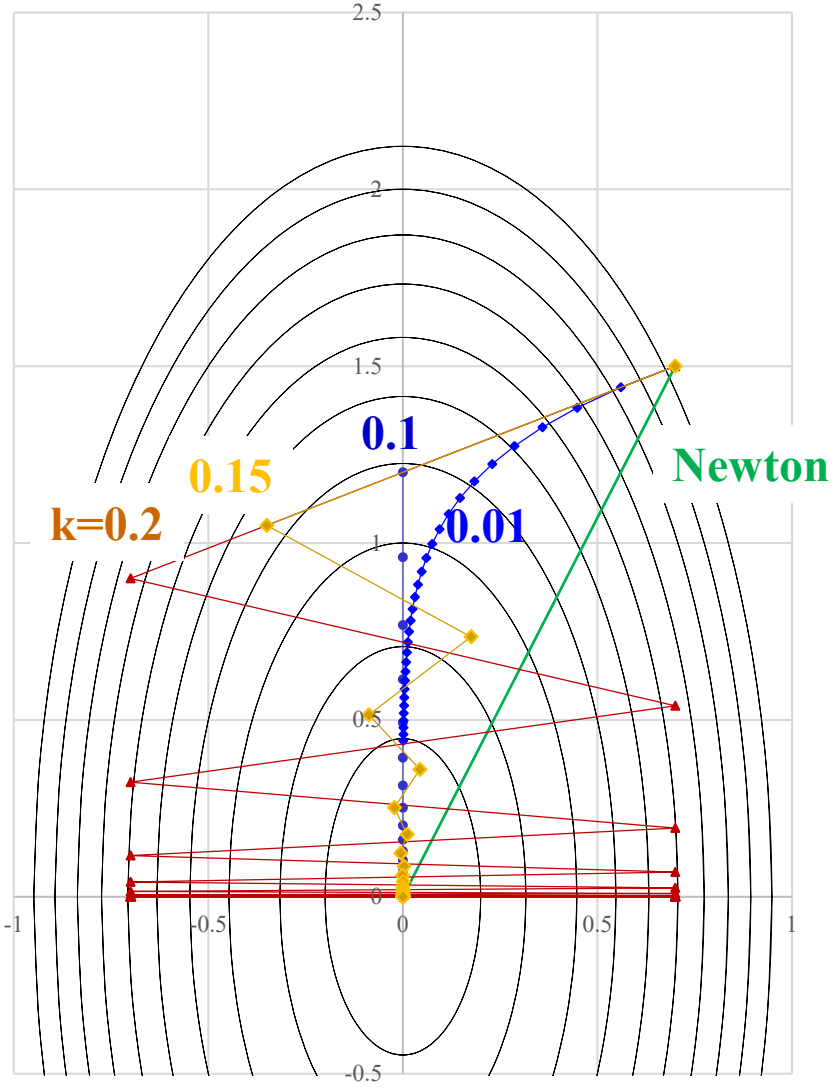
  0.2, 0.15: 振動しながら収束

  0.1: 最初の1度目で x_1の最適値に到達

  0.01: 振動せず、緩やかに収束

「S^2が大きく非対称な場合、最急勾配方向は最小値方向とは大きく異なることがある」

=> 共役勾配法

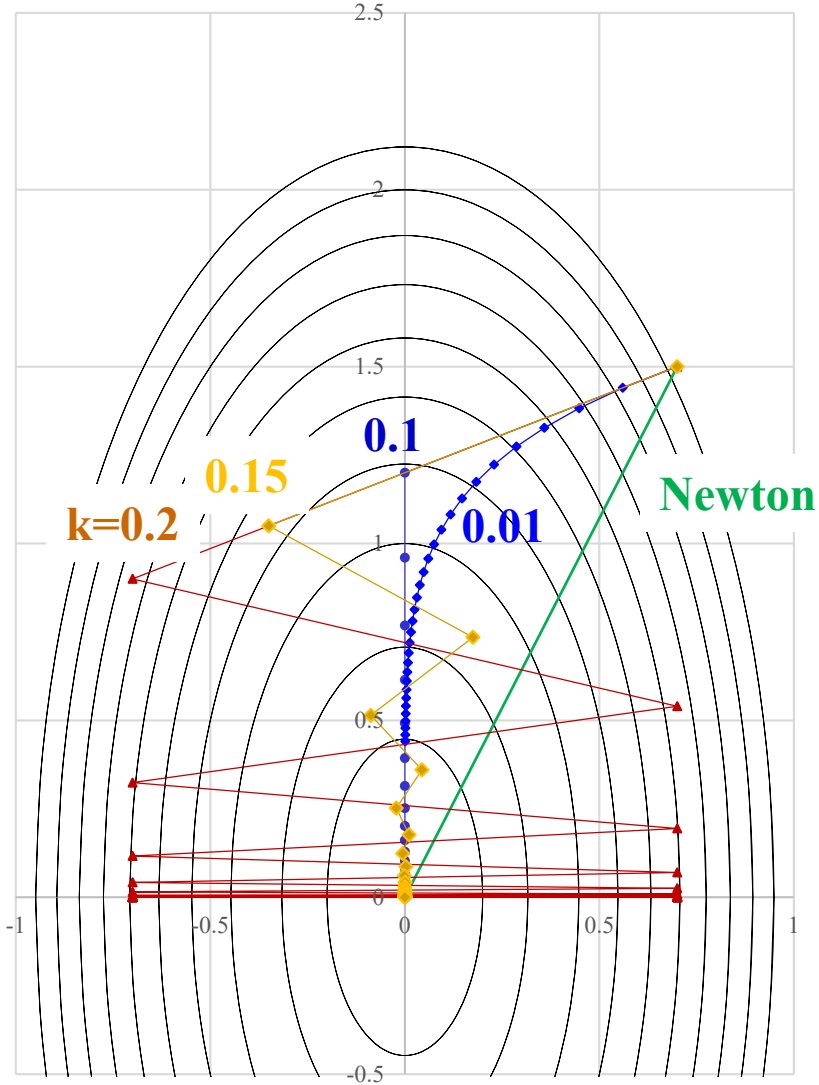


最急降下法 (Steepest Descend) 法

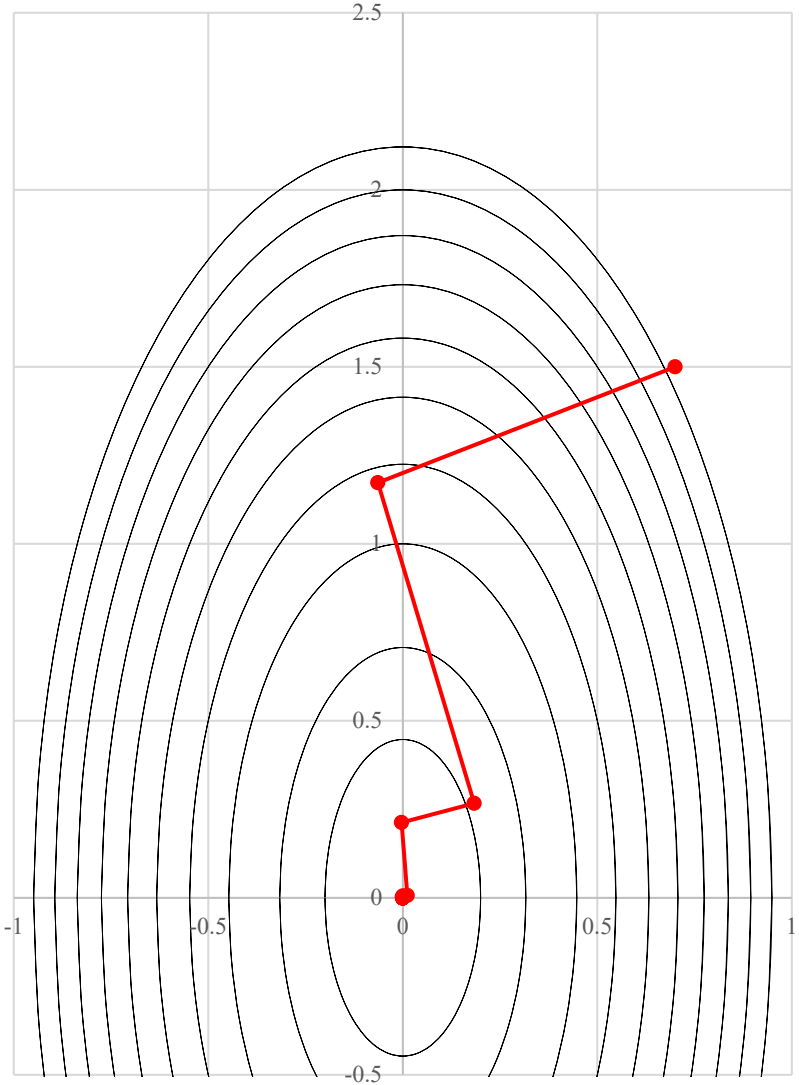
矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

効率的な方法: 各ステップの α を直線探索法で決定する

直線探索を行わない場合



直線探索を行う場合



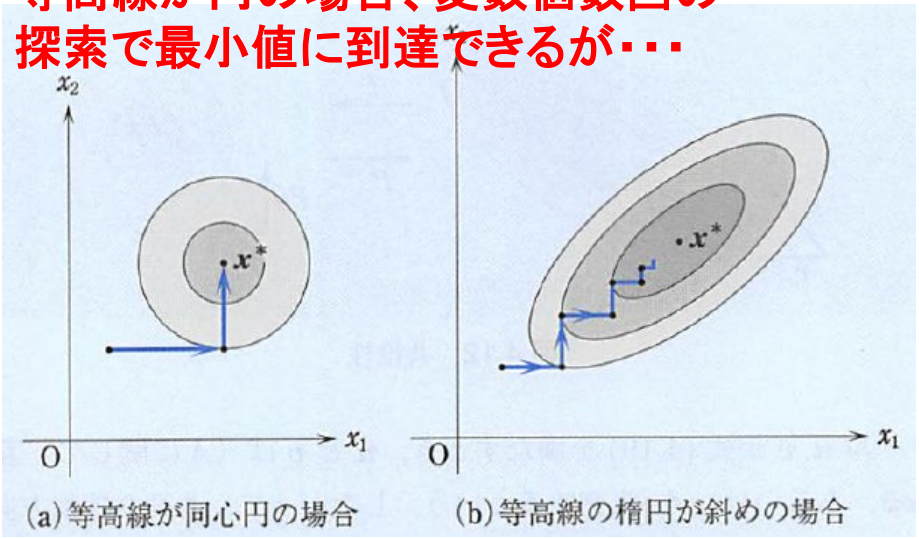
共役勾配 (Conjugate Gradient) 法

矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

行列 A に対してベクトル u, v が $u^tAv = 0$ であるとき、u と v は互いに共役の関係にあるという

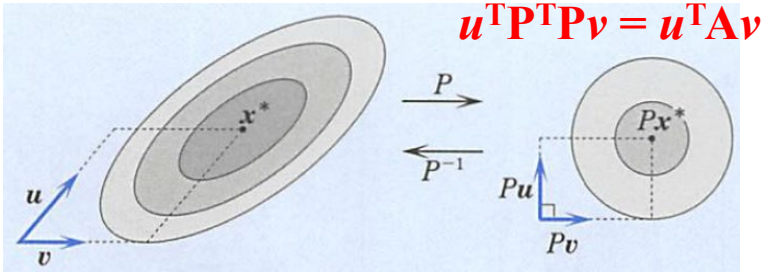
・共役な探索方向に沿って正確な直線探索を実行していけば、
有限回の反復で2次関数の最小解に到達することが期待される

等高線が円の場合、変数個数回の探索で最小値に到達できるが・・・



共役なベクトルと楕円ー円変換

$u^{\mathrm{T}}\mathrm{P}^{\mathrm{T}}\mathrm{P}v = u^{\mathrm{T}}\mathrm{A}v = 0$



1. 初期値 x_0 を与える
2. 初期探索方向 d を再急降下方向にとる
$$\mathbf{d} = -\nabla f$$
3. 直接探索法に従ってα_kを決め、
$$x_{k+1} = x_k + \alpha_k d_k$$
を計算する
4. 探索方向を
$$\mathbf{d}_k = \mathbf{d}_{k-1} - \frac{\mathbf{d}_{k-1} \cdot \nabla f(\mathbf{x}_k)}{\mathbf{d}_{k-1} \cdot \mathbf{d}_{k-1}} \nabla f(\mathbf{x}_k)$$
に更新する
5. 3,4を繰り返して収束させる

4.では、$\mathbf{d}_k$は $\mathbf{d}_i$ ($i = 1, \ldots, k$-1) のすべてに直交するため、このループは有限回しか実行できない => 時々$\mathbf{d}_k$をリセット

Marquart法

N個の変数 x_i をもつm個の関数 $f_j(x_i)$の自乗和

$$F(x_i)=\sum_{j=1}^{m} f_j(x_i)^2$$

の最小値(最大値)を求める

$$f_j(x_i+\delta x_i)\sim f_j(x_i)+\left(\frac{\partial f_j}{\partial x_k}\right)(\delta x_i)=f_j(x_i)+\mathbf{A}\boldsymbol{\delta}\mathbf{x_i} \qquad A_{jk}=\frac{\partial f_j}{\partial x_k}$$

と近似すると、

$$F(x_i+\delta x_i)\sim F(x_i)^2+2\sum_{j,k} f_j A_{jk}\delta x_k+\sum_{j,k,k'} A_{jk}A_{ik'}\delta x_k\delta x_{k'}$$

$$\frac{\partial F(x_i)}{\partial \delta x_k}\sim 2\sum_{j}\left(f_j A_{jk}+A_{jk}A_{jk}\delta x_j\right)=0$$

$$\boldsymbol{\delta}\mathbf{x}=-\left(\mathbf{A}^{\mathbf{t}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathbf{t}}\left(f_j\right)$$ **Gauss-Newton法**

Levenberg-Marquart法

$$\boldsymbol{\delta}\mathbf{x}=-\left(\mathbf{A}^{\mathbf{t}}\mathbf{A}+\lambda I\right)^{-1}\mathbf{A}^{\mathbf{t}}\left(f_j\right)$$ **λの最適値がよくわからない**

$$\boldsymbol{\delta}\mathbf{x}=-\left(\mathbf{A}^{\mathbf{t}}\mathbf{A}+\lambda\,\mathrm{diag}(\mathbf{A}^{\mathbf{t}}\mathbf{A})\right)^{-1}\mathbf{A}^{\mathbf{t}}\left(f_j\right)$$ **A行列の対角和に比例させる**

直線探索法

単体 (Simplex) 法 (Amoeba法)

南茂夫 編著、科学計測のための波形データ処理、CQ出版 (1986年)

単体 (Simplex): n次元空間で (n+1) 個の頂点が作る図形

$F(x_i)$の最小値を求める

1. $(n+1)$個の初期値で頂点 $F(\boldsymbol{x}_i)$ $(i = 1,2,\cdots,n+1)$ を
   $F(\boldsymbol{x}_i) > F(\boldsymbol{x}_{i'})$ $(i < i')$ となるように並べ替える
2. 最大値を取る頂点 $\boldsymbol{x}_i$ 以外が作る重心を $x_G = \sum_{i=2} x_i / n$ とする

3. 直線 $\boldsymbol{x}_1$- $\boldsymbol{x}_G$ 上で新しい点を次の順に求めて F を計算する

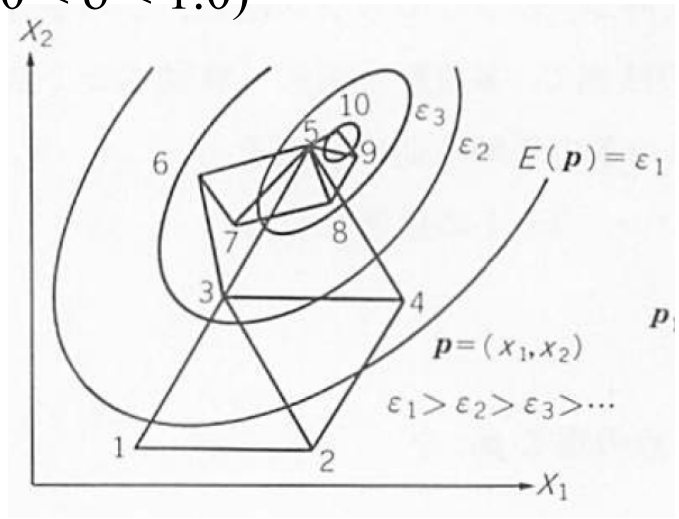
(i) 鏡映 : $\boldsymbol{x}_R = \boldsymbol{x}_1 + \alpha(\boldsymbol{x}_G - \boldsymbol{x}_1)$ (たとえば $\alpha = 2$)

(ii) 拡大 : $\boldsymbol{x}_E = \boldsymbol{x}_1 + \beta(\boldsymbol{x}_G - \boldsymbol{x}_1)$ (たとえば $\beta > 2$)

(iii) 縮小鏡映 : $\boldsymbol{x}_{CR} = \boldsymbol{x}_1 + \gamma(\boldsymbol{x}_G - \boldsymbol{x}_1)$ (たとえば $1.0 < \gamma < 2.0$)

(iv) 縮小 : $\boldsymbol{x}_{CW} = \boldsymbol{x}_1 + \delta(\boldsymbol{x}_G - \boldsymbol{x}_1)$ (たとえば $0 < \delta < 1.0$)

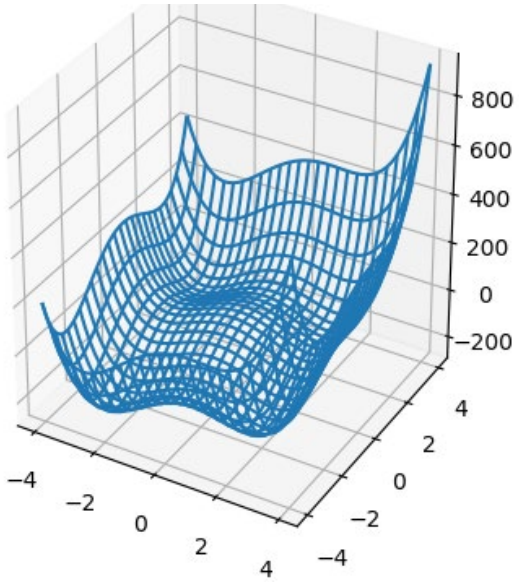
4. (i)~(iv)のうちで最初に $F(\boldsymbol{x}) < F(\boldsymbol{x}_1)$を満たす点を
   $\boldsymbol{x}_1$ と交換する。
   この操作を繰り返す



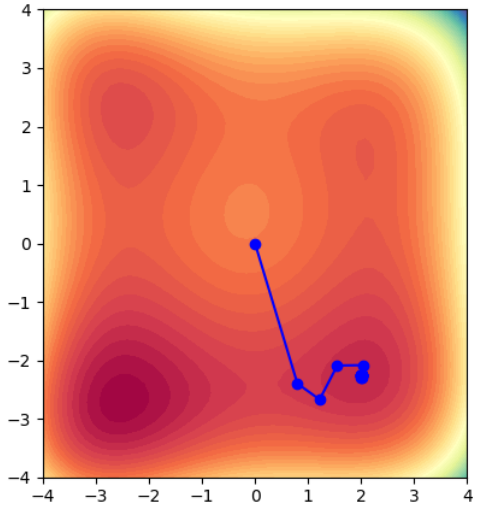
Comparison

http://conf.msl.titech.ac.jp/Lecture/python/index-numericalanalysis.html

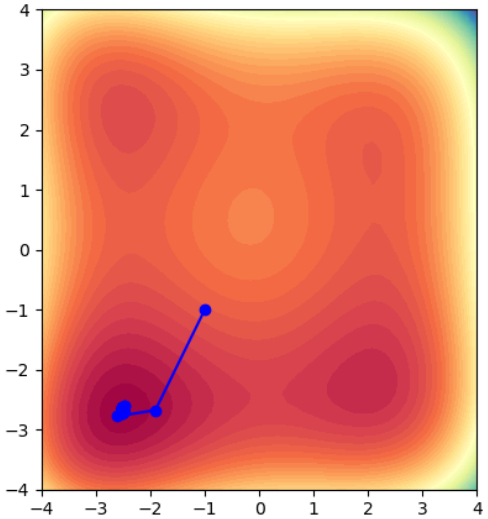
optimize-sd-cg2d-linesearch.py, optimize-newton-raphson2d.py



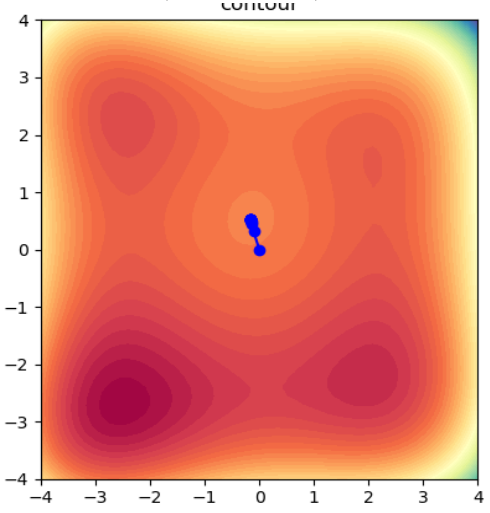
From (0.0 0.0) cg simple



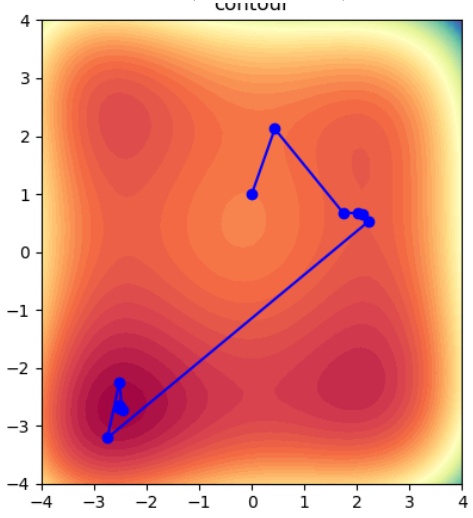
From (-1.0 -1.0) cg simple



From (0.0 0.0) Newton

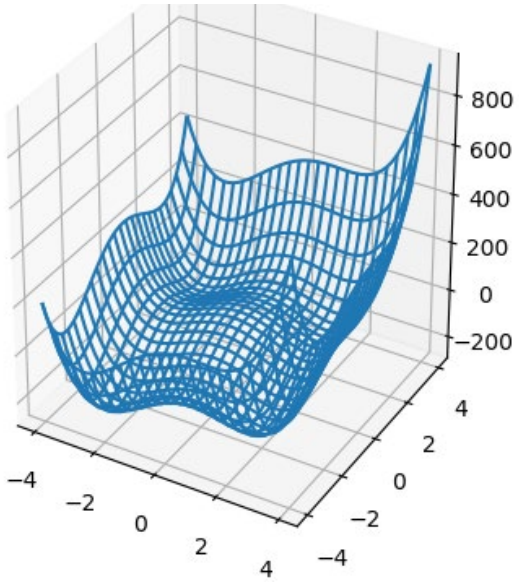


From (0.0 1.0) SD armijo

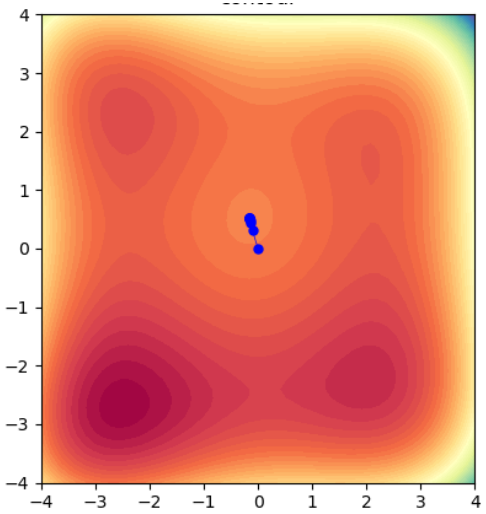


Comparison

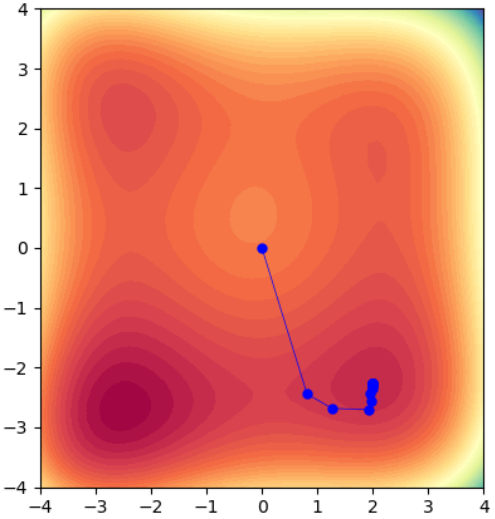
¥Programs¥python¥optimize.py in Programs.zip



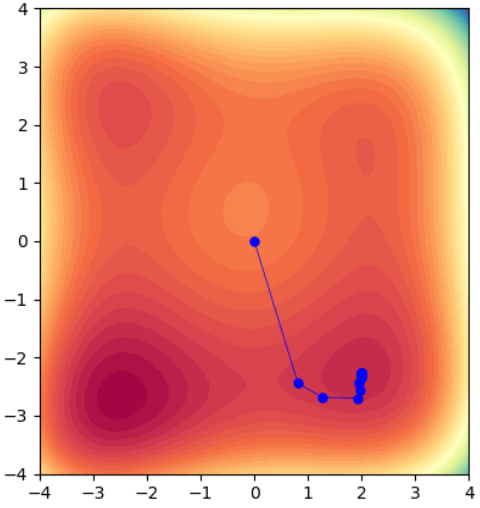
From (0.0 0.0) Newton



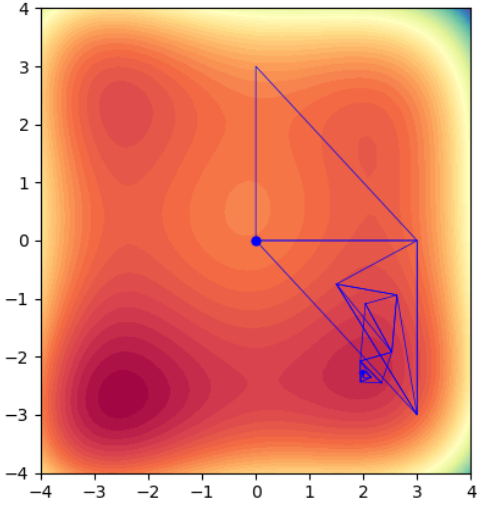
From (-1.0 -1.0) DFP golden



From (0.0 0.0) BFGS golden



From (0.0 1.0) Simplex



Main algorism:

Newton, DFP, BFGS

SD, CG

Simplex

Line search:

Golden, Armijo

非線形最小化問題の解法

$F(x)$ の最小値(最大値)を求める

勾配法

- **Newton-Raphson法:**
  二次微分行列を使って最小値を探索
- **準Newton法**
  二次微分行列を一次微分ベクトルから近似して最小値を探索
- **最急降下法 (Steepest Descent):**
  一次微分から傾きの方向のみで最小値を探索
- **共役勾配 (Conjugate Gradient) 法:**
  変数の変位ベクトルの共役方向へ最小値を探索
- **Marquart法**
  $f_j(x_i)$ の一次微分の行列を使って最小値を探索

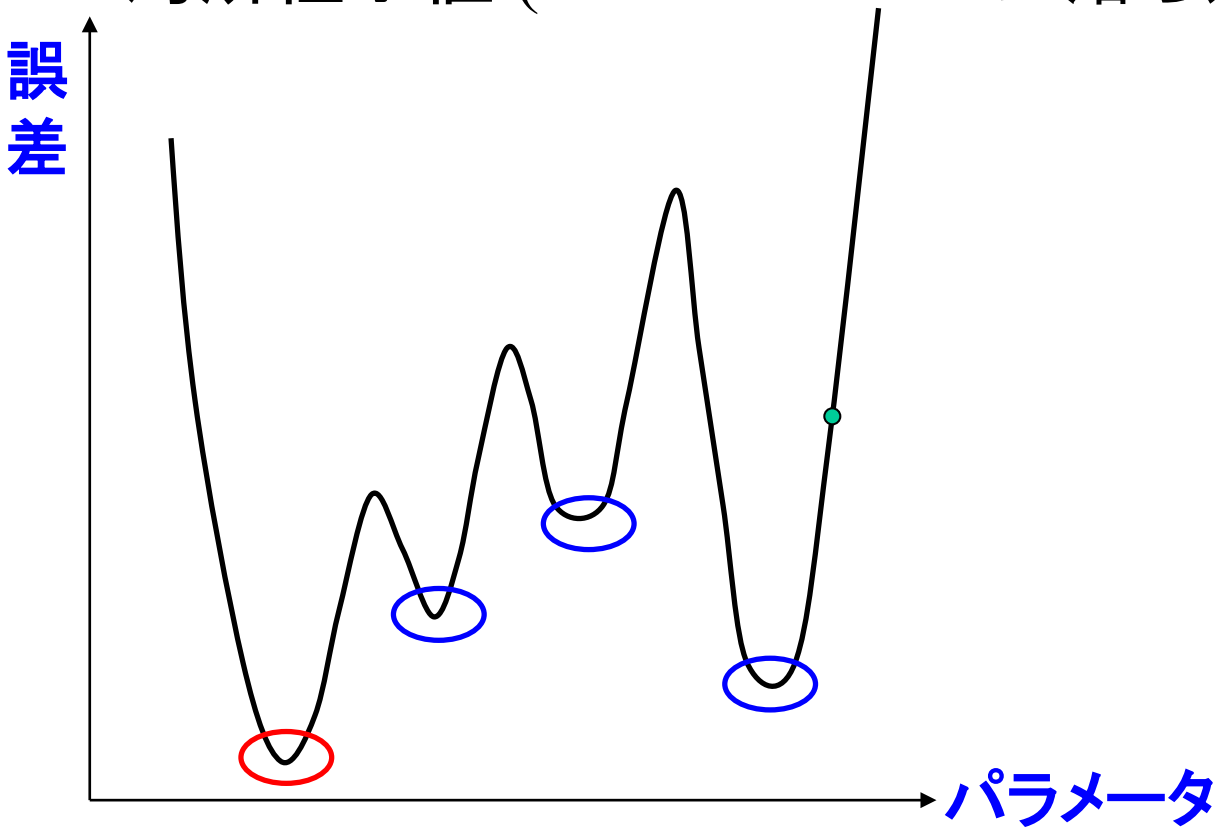
直接探索法

- **単体 (Simplex) 法 (Amoeba法)**
  一定のルールに従い、試行錯誤で最小値を探索

非線形 (多値) 方程式の注意

- 解が複数ある場合も
- ほとんどの場合、1度の計算で最適解を求めることは無理
  - 収束したことを確認する
  - 大域最小値を求める

⇔局所極小値 (local minimumに落ち込む)



非線形最適化アルゴルリズムの傾向

	A	B
収束速度	×	○
収束安定性	○	×
安定収束範囲	○	×
使い方	第一段階	第二段階

A: 単体 (Simplex) 法

A,B: 共役勾配法 (Conjugate Gradient: CG)

B: 最急降下法 (Steepest Descent: SD)

B: Newton-Raphson法　・準Newton法

- **Davidson-Fletcher-Powell (DFP)**
- **Broyden-Fletcher-Goldfarb-Shanno (BFGS)**

matplotlib での特殊文字表示: BMシフト

課題

バースタイン・モスシフト (縮退半導体の E_F) ΔE_gをキャリア濃度 Ne の関数としてグラフに描け。有効質量は自由電子の質量とし、横軸 Neは対数プロットせよ

$$\Delta E_g^{BM} = \frac{h^2}{m_{de}} \left(\frac{3N_e}{16\sqrt{2}\pi} \right)^{2/3}$$

PowerPoint 等 のプレゼンテーションファイルにして提出

期限: 今日の17:00までに

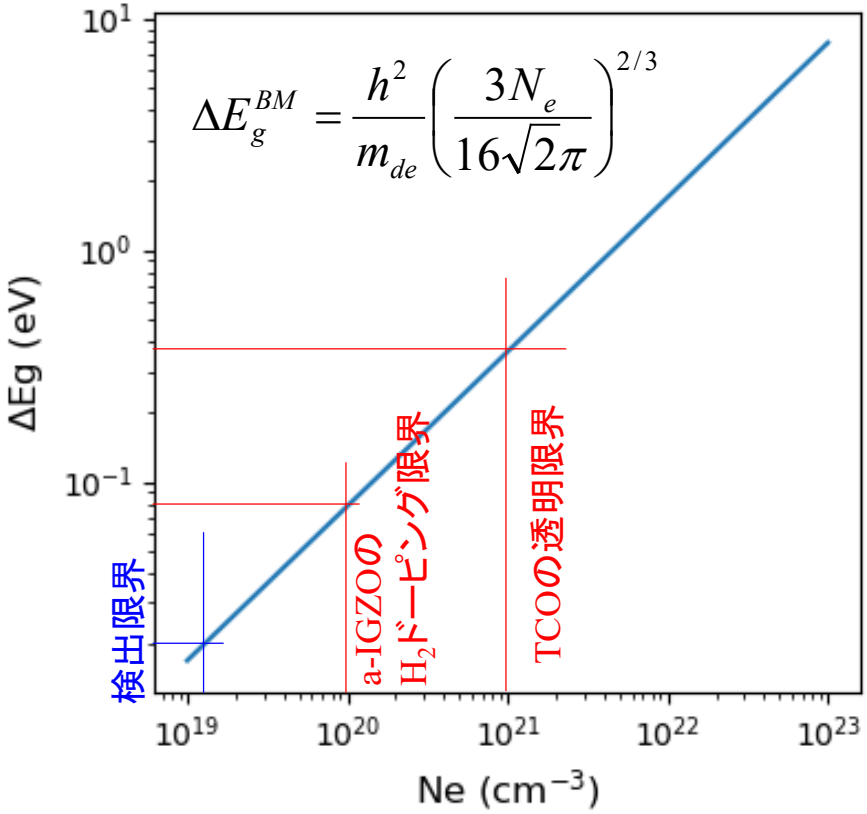
できたところまでで可

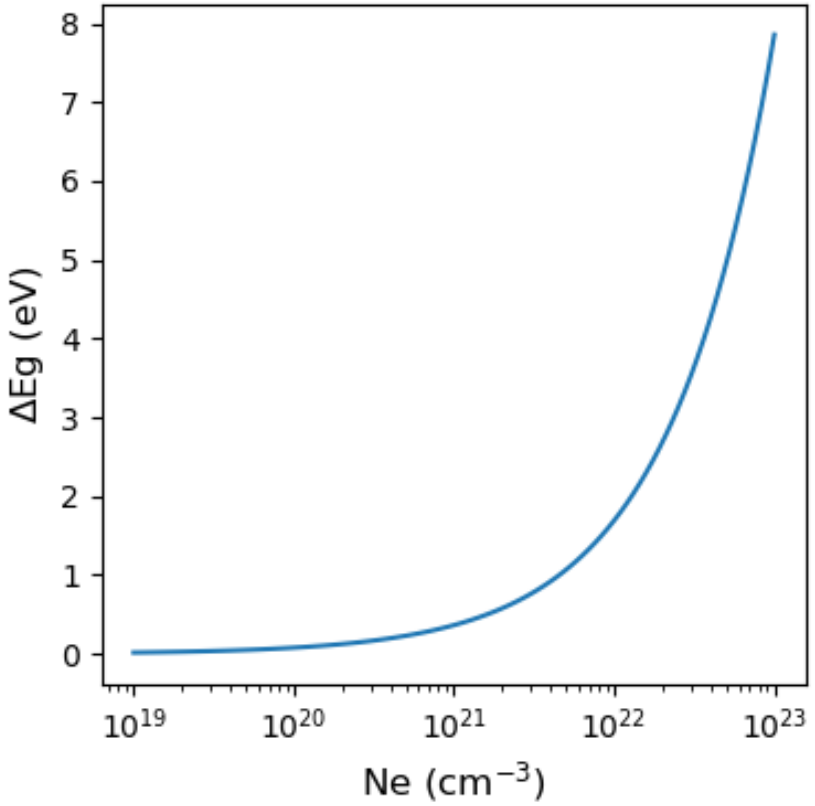
解答

電子濃度範囲:

TCOの透明限界: ~10^{21} cm^{-3}

金属の電子濃度　: 5x10^{22} cm^{-3} ~ 10^{23} cm^{-3}





プログラム (抜粋)

BMshift.py

```
import numpy as np
from numpy import sqrt, exp, sin, cos, tan, pi
from matplotlib import pyplot as plt

pi   = 3.14159265358979323846
h    = 6.6260755e-34    # Js";
e    = 1.60218e-19      # C    浮動小数点定数はe表記を使う

# parameters
logNemin = 19.0 # in log10(cm-3)     単位を明記
logNemax = 23.0 # in log10(cm-3)
nlogNe   = 101

def main():
    global logNemin,logNemax, nlogNe

    logNestep = (logNemax - logNemin) / (nlogNe - 1)
    Ne  = []
    dEg = []
    for i in range(nlogNe):
        logNe = logNemin + i * logNestep
        n  = 10.0**logNe * 1.0e6          # in m^-3   単位を明記
        de = (h*h / me) * pow(3.0 * n / 16.0 / sqrt(2) / pi,
2.0/3.0) / e # in eV
        Ne.append(n * 1.0e-6)
        dEg.append(de)

    print("")
    print("plot")
    fig = plt.figure(figsize = (8, 4))
    ax1 = fig.add_subplot(1, 2, 1)
    ax2 = fig.add_subplot(1, 2, 2)

    ax1.plot(Ne, dEg)
    ax1.set_xscale("log")
    ax1.set_yscale("log")
# matplotlibの表示文字列には、TeX 形式が使える
#  $ ~ $ で囲われた範囲が TeX形式
#      ギリシャ文字: ¥alpha, ¥Alpha など
#      上付き文字: ^{ と } で囲う
#      下付き文字: _{ と } で囲う
    ax1.set_xlabel("Ne (cm$^{-3}$)", fontsize = fontsize)
    ax1.set_ylabel("$¥Delta$Eg (eV)", fontsize = fontsize)
    ax2.plot(Ne, dEg)
    ax2.set_xscale("log")
    ax2.set_xlabel("Ne (cm$^{-3}$)", fontsize = fontsize)
    ax2.set_ylabel("$¥Delta$Eg (eV)", fontsize = fontsize)
    plt.tight_layout()

    plt.pause(0.1)
    input()

if __name__ == "__main__":
    main()
```

課題

バンド構造 band.csv から、数値微分により $d^2E(k)/dk_2$ を求め、有効質量 m_e^* と k の関係をグラフに描け。

格子定数は $a = 4.0$ Å とする。

異なる精度の数値微分をし、有効質量の精度の比較をするとbetter。

PowerPoint 等 のプレゼンテーションファイルにして提出
期限：今日の17：00までに
できたところまでで可

数値微分: 有効質量

7. 図4 有効質量

LCAOバンド

$$E(k)=\varepsilon_1-2|h_{12}|\cos(ka)\sim\varepsilon_1-2|h_{12}|+|h_{12}|a^2k^2+O((ka)^4)$$

自由電子

$$E(k)=E_0+\frac{|\mathbf{P}|^2}{2m}=E_0+\frac{\hbar^2}{2m}|\mathbf{k}|^2$$

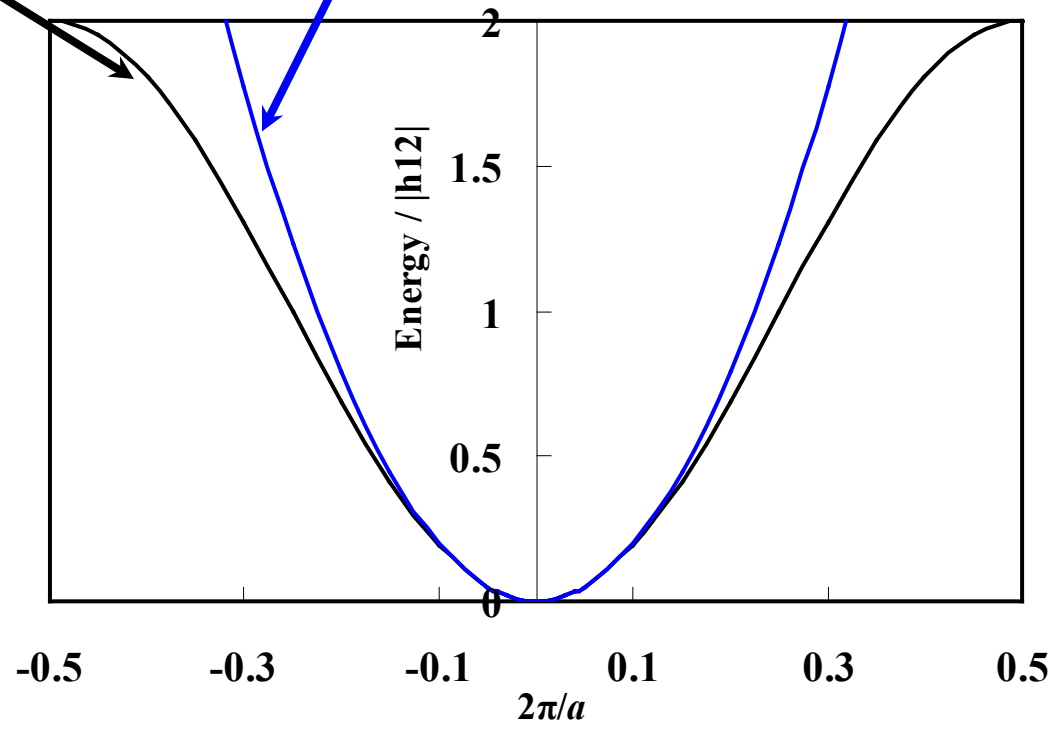
$$\frac{1}{m^*}=\frac{1}{\hbar^2}\frac{\partial^2E_n(\mathbf{k})}{\partial k^2}$$

$$m^*=\frac{\hbar^2}{2|h_{12}|a^2}$$

大きな混成($|h_{12}|$)に
より質量m^*は小さくなる

バンド幅 $W=4|h_{12}|$

$$m_e^*=2\hbar^2/Wa^2$$



有効質量

k は逆格子の内部座標: 一般に [-½ ½] の範囲で表示される
単位変換 $k_{real} = (2\pi/a)\, k$

$$m^* = \hbar^2 \left(\frac{\partial^2 E_n(\mathbf{k})}{\partial k_{real}^{\ 2}} \right)^{-1} = \hbar^2 \left(\frac{2\pi}{a} \right)^2 \left(\frac{\partial^2 E_n(\mathbf{k})}{\partial k^2} \right)^{-1}$$

一般に、有効質量は 電子の静止質量 $m_e^{\ 0}$ との比であらわす

$$m^* / m_e^{\ 0} = \hbar^2 \left(\frac{\partial^2 E_n(\mathbf{k})}{\partial k_{real}^{\ 2}} \right)^{-1} / m_e^{\ 0} = \hbar^2 \left(\frac{2\pi}{a} \right)^2 \left(\frac{\partial^2 E_n(\mathbf{k})}{\partial k^2} \right)^{-1} / m_e^{\ 0}$$

数値計算: 微分

$\frac{df(x)}{dx}$ をコンピュータでどのように計算するか

微分 d を差分 Δ で置き換える

$$\frac{df(x)}{dx} \sim \frac{\Delta f(x)}{\Delta x} = \frac{f(x+h)-f(x)}{(x+h)-x} = \frac{f(x+h)-f(x)}{h}$$

h を 小さくすれば精度が上がる ⇔ 桁落ち誤差

32bit浮動小数点 (~7桁) : 扱う最小数値の 5桁下が限界
64bit浮動小数点 (~16桁): 扱う最小数値の 14桁下が限界

$$f(x+h) = f(x) + \frac{df(x)}{dx}h + \frac{1}{2}\frac{d^2 f(x)}{dx^2}h^2 + O(h^3)$$

$$\frac{f(x+h)-f(x)}{h} = \frac{df(x)}{dx} + \boxed{\frac{1}{2}\frac{d^2 f(x)}{dx^2}h + O(h^2)}$$

差分誤差

数値微分: 平均を取って精度を上げる

$$\frac{df(x)}{dx} \sim \frac{f(x+h)-f(x)}{h}$$

誤差: $$\frac{f(x+h)-f(x)}{h} = \frac{df(x)}{dx} + \frac{1}{2}\frac{d^2 f(x)}{dx^2}h + \frac{1}{3!}\frac{d^3 f(x)}{dx^3}h^2 + O(h^4)$$

$$\frac{df(x)}{dx} \sim \left[\frac{f(x+h)-f(x)}{h} + \frac{f(x)-f(x-h)}{h}\right]/2 = \frac{f(x+h)-f(x-h)}{2h}$$

$$f(x+h) = f(x) + \frac{df(x)}{dx}h + \frac{1}{2}\frac{d^2 f(x)}{dx^2}h^2 + \frac{1}{3!}\frac{d^3 f(x)}{dx^3}h^3 + O(h^4)$$

$$f(x-h) = f(x) - \frac{df(x)}{dx}h + \frac{1}{2}\frac{d^2 f(x)}{dx^2}h^2 - \frac{1}{3!}\frac{d^3 f(x)}{dx^3}h^3 + O(h^4)$$

誤差: $$\frac{f(x+h)-f(x-h)}{2h} = \frac{df(x)}{dx} + \frac{1}{3!}\frac{d^3 f(x)}{dx^3}h^3 + O(h^4)$$

二階微分

一階微分を前身差分で計算してから二階微分を計算すると・・・

$$\frac{d^2x(t)}{dt^2} = \frac{\frac{dx}{dt}(t+\Delta t) - \frac{dx}{dt}(t)}{\Delta t}$$

$$\sim \frac{\frac{x(t+\Delta t)-x(t)}{\Delta t} - \frac{x(t)-x(t-\Delta t)}{\Delta t}}{\Delta t} = \frac{x(t+2\Delta t)-2x(t+\Delta t)+x(t)}{\Delta t^2}$$

偶数階微分では、結果が $t+\Delta t$、$t-\Delta t$ について対称になる式を取れる

$$\frac{d^2x(t)}{dt^2} \sim \frac{\frac{x(t+\Delta t)-x(t)}{\Delta t} - \frac{x(t)-x(t-\Delta t)}{\Delta t}}{\Delta t}$$

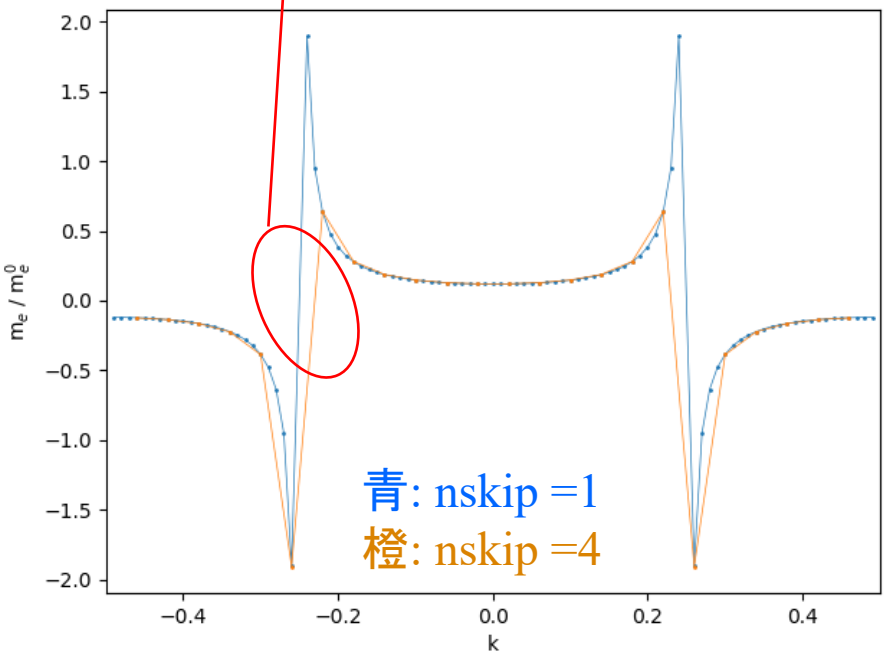
$$= \frac{x(t+\Delta t)-2x(t)+x(t-\Delta t)}{\Delta t^2}$$

2つの式では、横軸がひとつずれるので注意！

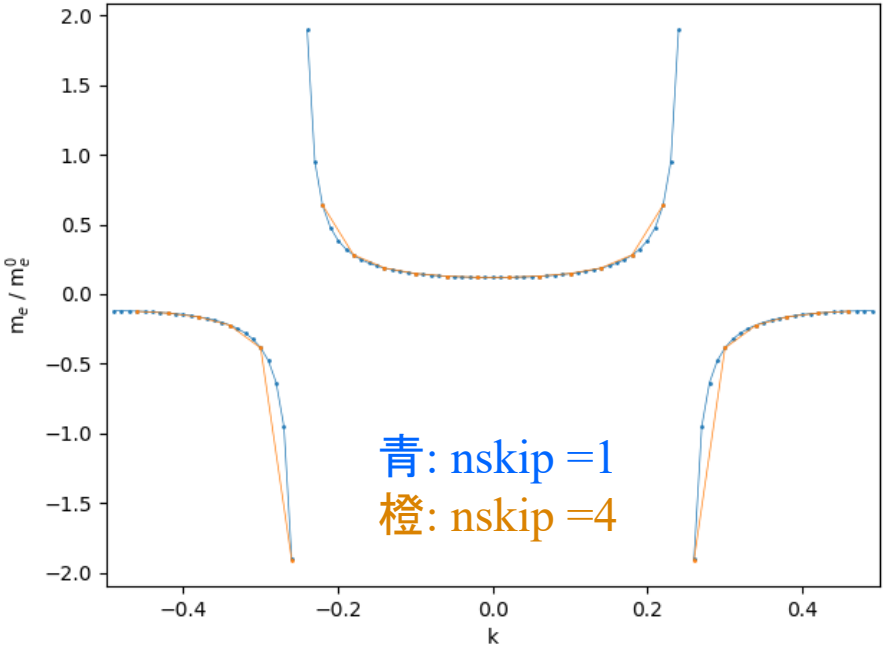
プログラム (抜粋)

BMshift.py

描いてはいけない線



データに None (未定義値) を挿入することで
描いてはいけない線を消した



プログラム (抜粋)

BMshift.py

```
#====================================
# parameters
#====================================
#a   = 4.0 # A
a = 4.0e-10 #m

infile = 'band.csv'
#有効質量の符号が変わる点をつなぐかどうかのフラグ
cutline = 1

def read_csv(fname):
    x = []
    y = []
    with open(fname) as f:
        fin = csv.reader(f)
        xlabel, ylabel, = next(fin)
        for row in fin:
            try:
                x.append(float(row[0]))
                y.append(float(row[1]))
            except:
                print("Warning: Invalid float data [{}] or [{}]".format(row[0],
row[1]))

    return xlabel, ylabel, x, y

def main():
    klabel, Elabel, k, E = read_csv(infile)
#入力データで使う変数を計算
    nk = len(k)
    dk = k[1] - k[0]
#共通の定数は先に計算
    km = hbar * hbar * (pi2 / a)**2.0
#微分の精度を比較するため、h = nskip*dk にする
    nskip = 1
    xk  = []
    ymc = []
#符号の変化を検出するため、符号変数を用意
signprev = None
for i in range(nskip, nk - nskip, nskip):
#2階微分を計算
        d2Edk2c = (E[i+nskip] + E[i-nskip] - 2 * E[i]) * e / pow(nskip *
dk, 2.0)
#2回微分はゼロになることがあるので、まずは1/m*を計算
        minv = d2Edk2c / km
        print(i, E[i-1], E[i], E[i+1], minv)
#1/m*が1/meより非常に小さければ、m*は計算しない
        if abs(minv) <= 1.0e20:  # << 1.0/me ~ 1e30
#符号が反転する場所でグラフの線を切断するときは
#Noneデータを追加する。
            if cutline:
                xk.append(k[i])
                ymc.append(None)
#反転した符号を記録
         signprev = -signprev
            continue
        else:
            m = km / d2Edk2c

        if signprev is None:
            signprev = m
        elif signprev * m < 0.0:
            if cutline:
```

プログラム (抜粋)

BMshift.py

```
#共通の定数は先に計算
   km = hbar * hbar * (pi2 / a)**2.0
#微分の精度を比較するため、h = nskip*dk にする
    nskip = 1
    xk  = []
    ymc = []
#符号の変化を検出するため、符号変数を用意
signprev = None
for i in range(nskip, nk - nskip, nskip):
#2階微分を計算
        d2Edk2c = (E[i+nskip] + E[i-nskip] - 2 * E[i]) * e / pow(nskip *
dk, 2.0)
#2回微分はゼロになることがあるので、まずは1/m*を計算
        minv = d2Edk2c / km
        print(i, E[i-1], E[i], E[i+1], minv)
#1/m*が1/meより非常に小さければ、m*は計算しない
        if abs(minv) <= 1.0e20:  # << 1.0/me ~ 1e30
#符号が反転する場所でグラフの線を切断するときは
#Noneデータを追加する。
            if cutline:
                xk.append(k[i])
                ymc.append(None)
#反転した符号を記録
         signprev = -signprev
            continue
        else:
            m = km / d2Edk2c

#符号が反転する場所でグラフの線を切断するときは
#Noneデータを追加する。
        if signprev is None:
#signprevが 初期値 None である場合は 符号の最初の値を代入
            signprev = m
        elif signprev * m < 0.0:
            if cutline:
                xk.append(k[i])
                ymc.append(None)
#反転した符号を記録
            signprev = m

        xk.append(k[i])
        ymc.append(m / me)

    plt.plot(xk,  ymc,  linewidth = 0.5, marker = 'o', markersize = 1.0,
label = 'nskip = 1')
    plt.xlabel(klabel)
    plt.ylabel("m$_e$ / m$_e^0$")
    plt.xlim([-0.5, 0.5])
#    plt.ylim([-0.5, 0.5])
    plt.tight_layout()

    plt.pause(0.1)
    print("Press ENTER to exit>>", end = '')
    input()


if __name__ == "__main__":
    main()
```

最小二乗法: 電界効果移動度

課題

1. 厚さ 100 nmの a-SiO_2 の単位面積当たり静電容量 C_{OX} を求めよ。
   a-SiO_2 の 比誘電率は $\varepsilon_r = 11.9$ とする。

2. TransferCurve.xlsxのデータから、飽和移動度を求めよ
   電極幅 W = 300 μm, L = 50 μm とする。
   飽和移動度を求める際の Vg、Vd は各自で選ぶこと。
   その値を選んだ理由も説明せよ。

PowerPoint 等 のプレゼンテーションファイルにして提出
期限：今日の17：00までに
できたところまでで可

FET特性の解析: 飽和領域

$$I_{DS} = \frac{W}{L}\mu C_{OX}\left[(V_{GS} - V_{th})V_{DS} - \frac{{V_{DS}}^2}{2}\right]$$

$V_{DS} > V_p = V_{GS} - V_{th}$

$$I_{DS} = \frac{W}{2L}\mu C_{OX}(V_{GS} - V_{th})^2$$

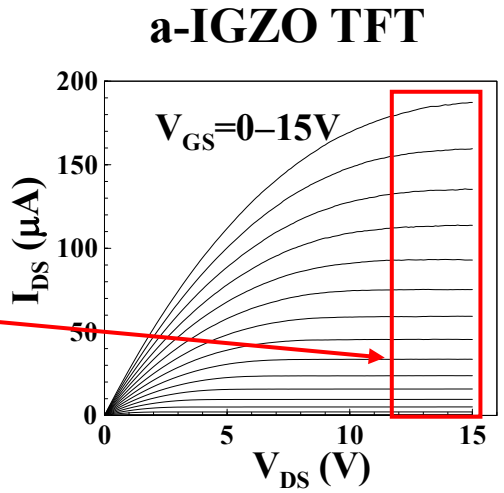
$$I_{DS}{}^{1/2} = \sqrt{\frac{W}{2L}\mu C_{OX}}(V_{GS} - V_{th})$$

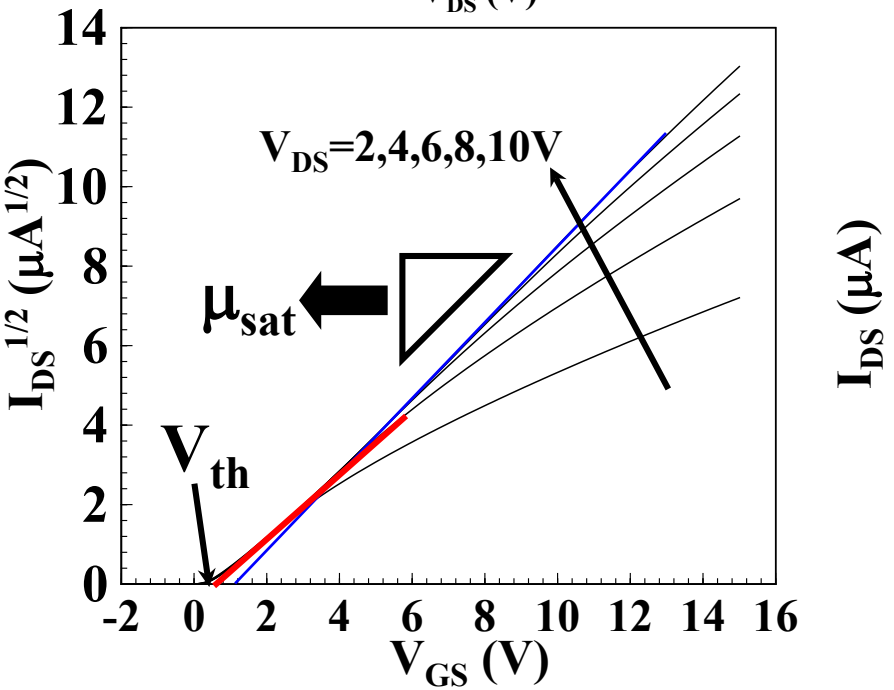
$I_{DS}{}^{1/2}$ vs. V_{GS}をプロット

V_{GS}軸切片: V_{th}

傾き: 飽和移動度

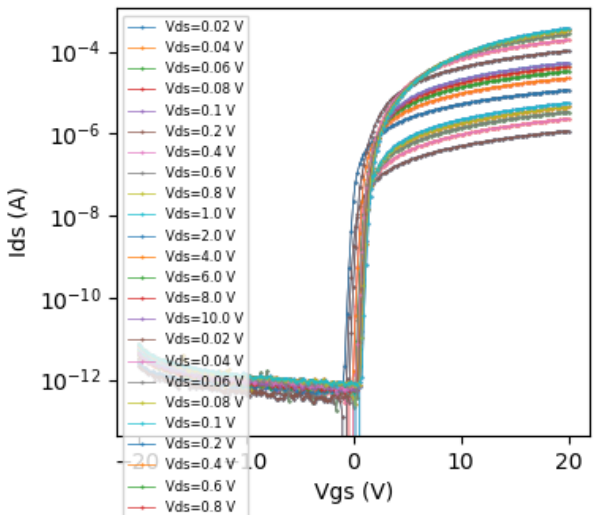
Saturation mobility, μ_{sat}

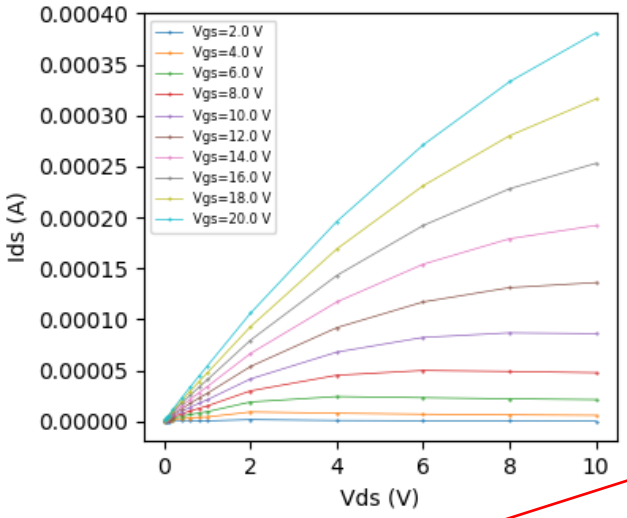


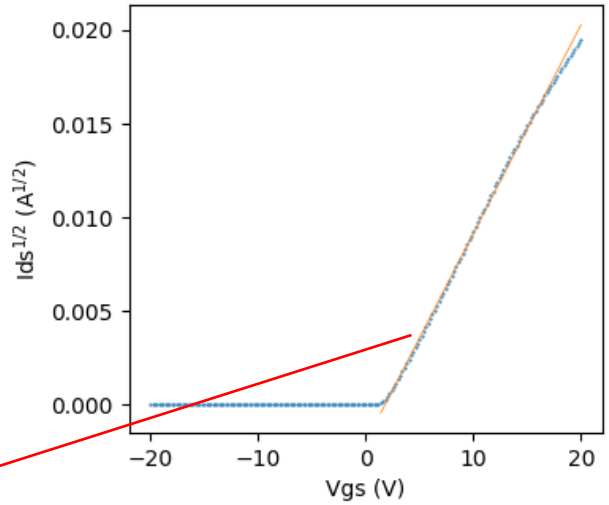


プログラム

TFTiv.py







直線に見える 1.9 ~ 10.0 V で、numpy.polyfit() で一次多項式にフィッティング

ai = np.polyfit(xfit, yfit, 1)

y = ai[1] + ai[0]x

Vth = -ai[1] / ai[0] = 1.794 V

$\frac{dIg^{\frac{1}{2}}}{dVgs}$ = ai[0] = 0.001114 $A^{1/2}/V$

$$I_{DS}{}^{1/2} = \sqrt{\frac{W}{2L}\mu C_{OX}}(V_{GS} - V_{th})$$

$$\boldsymbol{\mu_{sat} = \frac{(dIgs^{1/2}/dVgs)^2}{\frac{W}{2L}C_{ox}} = 0.000392\ m^2/Vs = 3.92\ cm^2/Vs}$$

プログラム (抜粋)

TFTiv.py

```
import re      # 正規表現モジュールを読み込む

#==================================
# parameters
#==================================
infile = 'TransferCurve.csv'

dg  = 100e-9 #m
erg = 11.9

W = 300.0e-6 #m
L = 50.0e-6

# Ids^(1/2)-VgsプロットをするVds
Vds0 = 10.0

# 余計な文字が含まれている文字列から、
# 浮動小数点に変換できる最初の文字列を切り出し、
# 浮動小数点に変換して返す
def pfloat(str, defval = None):
# 文字列から、浮動小数点に使える文字が連続している部分を切り出す
    m = re.search(r'([+¥-eE¥d¥.]+)', str)
# 一致した文字列を取得
    valstr = m.group()
    try:
        return float(valstr)
    except:
        return defval
```

```
def read_csv(fname):
    print("")
    with open(fname) as f:
        fin = csv.reader(f)

        labels = next(fin)
        xlabel = labels[0]

# label行が 空文字 の場合、データとしては読み込まない
        ylabels = []
        for i in range(1, len(labels)):
            if labels[i] == '':
                break
            ylabels.append(labels[i])
        ny = len(ylabels)

        x     = []
        ylist = []
        for i in range(ny):
            ylist.append([])

        for row in fin:
            x.append(pfloat(row[0]))
            for i in range(1, ny+1):
                v = pfloat(row[i])
                if v is not None:
                    ylist[i-1].append(v)
                else:
                    ylist[i-1].append(None)

    return xlabel, ylabels, x, ylist
```

プログラム (抜粋)

TFTiv.py

```
def main():
    Cox = erg * e0 / dg  #(F/m^2)
    print("")
    print("Cox = {:12.6g} [F/m^2]".format(Cox))

    xlabel, ylabels, Vgs, IdsVgs = read_csv(infile)
    nVgs = len(IdsVgs[0])
    nVds = len(ylabels)
    Vds = []
    for i in range(nVds):
        Vds.append(pfloat(ylabels[i]))
    print("")
    print("nVds=", nVds)
    print("nVgs=", nVgs)
    print("xlabel : ", xlabel)
    print("ylabels: ", ylabels)
    print("Vds: ", Vds)

# 出力特性 Ids - Vds をプロットするためのデータを作る
    IdsVds = np.empty([nVgs, nVds])
    for ig in range(nVgs):
        for id in range(nVds):
            IdsVds[ig][id] = IdsVgs[id][ig]

# sqrt(Ids) - Vgsプロットをする Vds0 のデータ番号 iVds を探す
# Vds[i] が小さい方から順に、Vds0 <= Vds[i] となる i を探すが、
# 浮動小数点誤差があることを考慮し、Vds0 - 1.0e-3 <= Vds[i] とする
    for i in range(nVds):
        if Vds0 - 1.0e-3 <= Vds[i]:
            iVds = i
            break

# sqrt(Ids) データを作る
    print("")
    print("Vds used: {} V (iVds = {})".format(Vds[iVds], iVds))
    sqrtIds = []
    for ig in range(nVgs):
        sqrtIds.append(sqrt(IdsVgs[iVds][ig]))

# 最小二乗法のデータ
    xfit = []
    yfit = []
    for i in range(nVgs):
        if xfitmin <= Vgs[i] <= xfitmax:
            xfit.append(Vgs[i])
            yfit.append(sqrtIds[i])
    print("")
    print("Least squares fitting:")
    print("Vgs range: {} - {} V".format(xfitmin, xfitmax))
    print("Vgs=", xfit)
    print("Igs^(1/2)=", yfit)
    ai = np.polyfit(xfit, yfit, 1)
# y = ai[1] + ai[0]x
    Vth  = -ai[1] / ai[0]
    grad = ai[0]
    mu = grad * grad / (W * Cox / 2.0 / L)
    print("Vth = {:6.4g} V".format(Vth))
    print("dIgs^1/2/dVgs = {:12.4g} A^(1/2)/V".format(grad))
    print("mu_sat = {} m^2/Vs = {} cm^2/Vs".format(mu, 1.0e4 * mu))
```

プログラム (抜粋)

TFTiv.py

```
# 最小二乗の結果を確認する直線
# フィッティング範囲より広くプロットする
    xcal = []
    ycal = []
    xx = xfitmin - 0.5
    xcal.append(xx)
    ycal.append(ai[1] + ai[0] * xx)
    xx = max(Vgs)
    xcal.append(xx)
    ycal.append(ai[1] + ai[0] * xx)

# 伝達特性  Ids-Vgs のグラフ
    for id in range(nVds):
        ax1.plot(Vgs, IdsVgs[id],  linewidth = 0.5, marker = 'o',
markersize = 0.5,
                 label = 'Vds={} V'.format(Vds[id]))
    for id in range(nVds):
        ax1.plot(Vgs, IdsVgs[id],  linewidth = 0.5, marker = 'o',
markersize = 0.5,
                 label = 'Vds={} V'.format(Vds[id]))
    ax1.set_xlabel('Vgs (V)')
    ax1.set_ylabel("Ids (A)")
    ax1.set_yscale('log')
# 凡例のアンカーを左上に設定
     ax1.legend(loc = 'upper left', fontsize = legend_fontsize)
```

```
# 出力特性  Ids-Vds のグラフ
# 20点の Vgs を選び、そのうち、線形でIdsが見えるデータのみ表示する
    nskip = int(nVgs / 20.0 + 1.0e-6)
    for ig in range(0, nVgs, nskip):
# 線形プロットで見えないほどIdsが小さいデータは表示しない
        if max(IdsVds[ig]) < 1.0e-7:
            continue
        ax2.plot(Vds, IdsVds[ig],  linewidth = 0.5, marker = 'o', markersize
= 0.5,
                 label = 'Vgs={} V'.format(Vgs[ig]))
    ax2.set_xlabel('Vds (V)')
    ax2.set_ylabel("Ids (A)")
    ax2.legend(loc = 'upper left', fontsize = legend_fontsize)

# Ids^(1/2) - Vgs グラフ
    ax3.plot(Vgs, sqrtIds,  linestyle = 'none', marker = 'o', markersize =
0.5)
    ax3.plot(xcal, ycal,   linestyle = '-', linewidth = 0.5)
    ax3.set_xlabel('Vgs (V)')
    ax3.set_ylabel("Ids$^{1/2}$ (A$^{1/2}$)")

    plt.tight_layout()

    plt.pause(0.1)
    print("Press ENTER to exit>>", end = '')
    input()
```