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

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

Hall効果

太田英二、坂田亮著、半導体の電子物性光学、培風館 (2005)

Hall効果測定の条件 $J_y = 0$ $\left(\mu = \frac{e\tau}{m_e^*}\right)$ **より**

$$en\mu \begin{pmatrix} \frac{1}{1+\mu^2 {B_z}^2} & \frac{\mu B_z}{1+\mu^2 {B_z}^2} \\ -\frac{\mu B_z}{1+\mu^2 {B_z}^2} & \frac{1}{1+\mu^2 {B_z}^2} \end{pmatrix} \begin{pmatrix} E_x \\ E_y \end{pmatrix} = \frac{\sigma_0}{1+\mu^2 {B_z}^2} \begin{pmatrix} 1 & \mu B_z \\ -\mu B_z & 1 \end{pmatrix} \begin{pmatrix} E_x \\ E_y \end{pmatrix} = \begin{pmatrix} J_x \\ J_y \end{pmatrix}$$

$$E_x = \frac{1}{\sigma_0} \frac{J_x \sigma_{xx}}{\sigma_{xx}\sigma_{yy} - \sigma_{yx}\sigma_{yx}} = \frac{1}{\sigma_0} J_x$$

$$E_y = -\frac{1}{\sigma_0} \frac{J_x \sigma_{yz}}{\sigma_{xx}\sigma_{yy} - \sigma_{yx}\sigma_{yx}} = \frac{1}{\sigma_0} \mu B_z J_x = \frac{1}{en} B_z J_x = R_H B_z J_x$$

$$\boldsymbol{R_H = -\frac{E_y}{B_z J_x}} = -\frac{V_H}{W}\frac{Wd}{I_x}\frac{1}{B_z} = -\frac{\boldsymbol{V_H}}{\boldsymbol{I_x}}\frac{\boldsymbol{d}}{\boldsymbol{B_z}} = -\frac{1}{en}$$ **(for electron)**

Hall effect: Two-carrier model

太田英二、坂田亮著、半導体の電子物性光学、培風館 (2005)

Current $\boldsymbol{J} = -e\langle \boldsymbol{v} \rangle = \frac{e^2 n\tau}{m_e^*} \begin{pmatrix} \frac{1}{1+\left(\frac{e\tau}{m_e^*}\right)^2 {B_z}^2} & \frac{\left(\frac{e\tau}{m_e^*}\right) B_z}{1+\left(\frac{e\tau}{m_e^*}\right)^2 {B_z}^2} & 0 \\ -\frac{\left(\frac{e\tau}{m_e^*}\right) B_z}{1+\left(\frac{e\tau}{m_e^*}\right)^2 {B_z}^2} & \frac{1}{1+\left(\frac{e\tau}{m_e^*}\right)^2 {B_z}^2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} E_x \\ E_y \\ E_z \end{pmatrix}$

Two carrier model

$$\boldsymbol{J} = \boldsymbol{J_e} + \boldsymbol{J_h} = (\sigma_e + \sigma_h)\boldsymbol{E} + (-\sigma_e \mu_e + \sigma_h \mu_h)\boldsymbol{E} \times \boldsymbol{B}$$

$$\rho_{xx}(B) = \frac{1}{e} \frac{(n_h\mu_h + n_e\mu_e) + (n_h\mu_h + n_e\mu_e)\mu_h\mu_e B^2}{(n_h\mu_h + n_e\mu_e)^2 + (n_h - n_e)^2 {\mu_h}^2 {\mu_e}^2 B^2}$$

$$\rho_{yx}(B) = B \frac{1}{e} \frac{(n_h{\mu_h}^2 - n_e{\mu_e}^2) + (n_h - n_e){\mu_h}^2{\mu_e}^2 B^2}{(n_h\mu_h + n_e\mu_e)^2 + (n_h - n_e)^2 {\mu_h}^2 {\mu_e}^2 B^2}$$

Hall effect: Two-carrier model with $n_e \sim n_h$

太田英二、坂田亮著、半導体の電子物性光学、培風館 (2005)

Two carrier model

$$\rho_{xx}(B) = \frac{1}{e}\frac{(n_h\mu_h+n_e\mu_e)+(n_h\mu_h+n_e\mu_e)\mu_h\mu_eB^2}{(n_h\mu_h+n_e\mu_e)^2+(n_h-n_e)^2\mu_h{}^2\mu_e{}^2B^2}$$

$$\rho_{yx}(B) = B\frac{1}{e}\frac{(n_h\mu_h{}^2-n_e\mu_e{}^2)+(n_h-n_e)\mu_h{}^2\mu_e{}^2B^2}{(n_h\mu_h+n_e\mu_e)^2+(n_h-n_e)^2\mu_h{}^2\mu_e{}^2B^2}$$

$\boldsymbol{n_e \sim n_h \sim n}$

$$\rho_{xx}(B) = \frac{1}{en}\frac{1+\mu_h\mu_eB^2}{\mu_h+\mu_e}$$

$$\rho_{yx}(B) = B\frac{1}{en}\frac{\mu_h-\mu_e}{\mu_h+\mu_e}$$

Hall effect: Two-carrier model with $\mu_h \sim \mu_e$

太田英二、坂田亮著、半導体の電子物性光学、培風館 (2005)

Two carrier model

$$\rho_{xx}(B) = \frac{1}{e}\frac{(n_h\mu_h+n_e\mu_e)+(n_h\mu_h+n_e\mu_e)\mu_h\mu_e B^2}{(n_h\mu_h+n_e\mu_e)^2+(n_h-n_e)^2\mu_h{}^2\mu_e{}^2B^2}$$

$$\rho_{yx}(B) = B\frac{1}{e}\frac{(n_h\mu_h{}^2-n_e\mu_e{}^2)+(n_h-n_e)\mu_h{}^2\mu_e{}^2B^2}{(n_h\mu_h+n_e\mu_e)^2+(n_h-n_e)^2\mu_h{}^2\mu_e{}^2B^2}$$

$\boldsymbol{\mu_h \sim \mu_e \sim \mu}$

$$\boldsymbol{\rho_{xx}(B)} = \frac{1}{e\mu(n_h+n_e)}\frac{1+\mu^2B^2}{1+(n_h-n_e)^2/(n_h+n_e)^2\mu^2B^2}$$

$$\sim \rho_0(1+\mu^2B^2-(n_h-n_e)^2/(n_h+n_e)^2\mu^2B^2)$$

$$\boldsymbol{\sim \rho_0(1+\mu^2B^2)}$$

$$\rho_{yx}(B) = B\frac{1}{e(n_h+n_e)}\frac{1+\mu^2B^2}{1+(n_h-n_e)^2/(n_h+n_e)^2\mu^2B^2} \sim B\frac{1}{e(n_h+n_e)}$$

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):

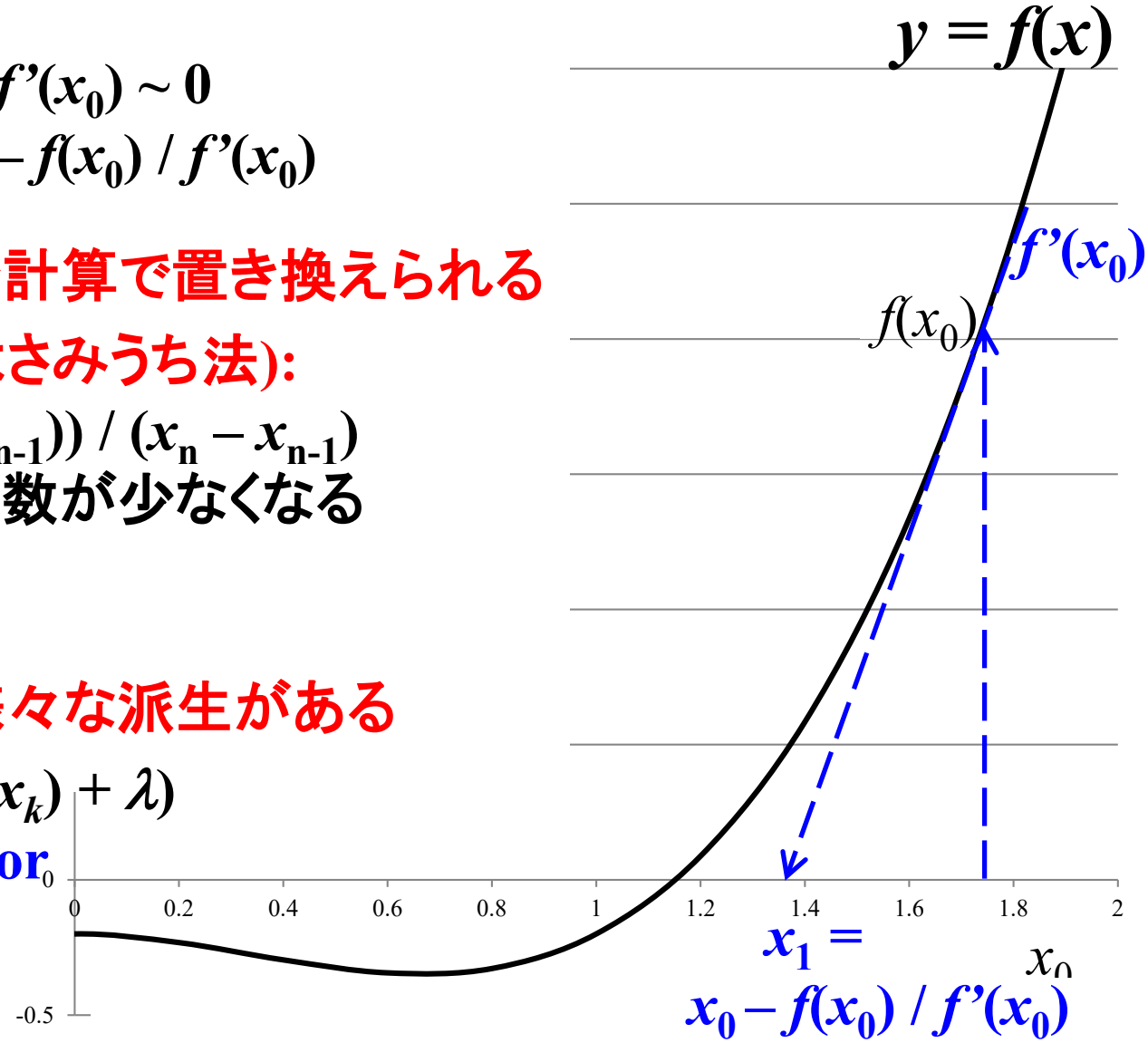
$$f'(x) = (f(x_n) - f(x_{n-1})) / (x_n - x_{n-1})$$

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

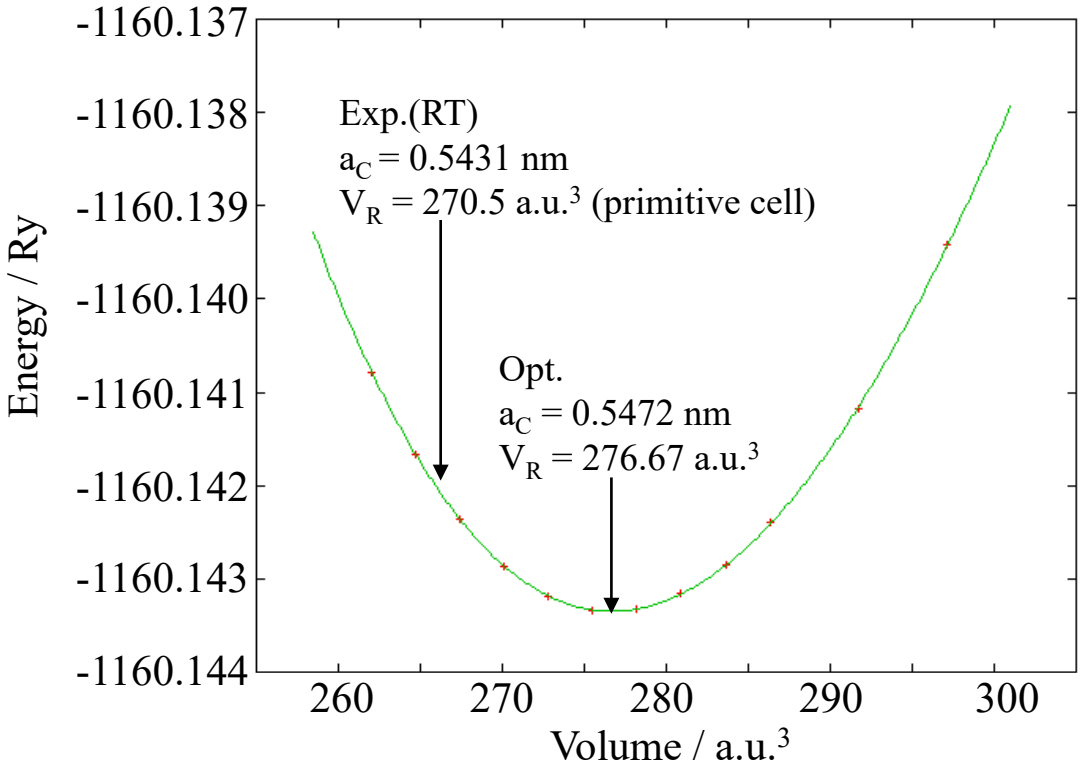


NL optimization of crystal structure: Illustrative approach

安定構造: 図解による解法

Calculate total energy by quantum calculations by varying a lattice parameter

***ex.* Si**

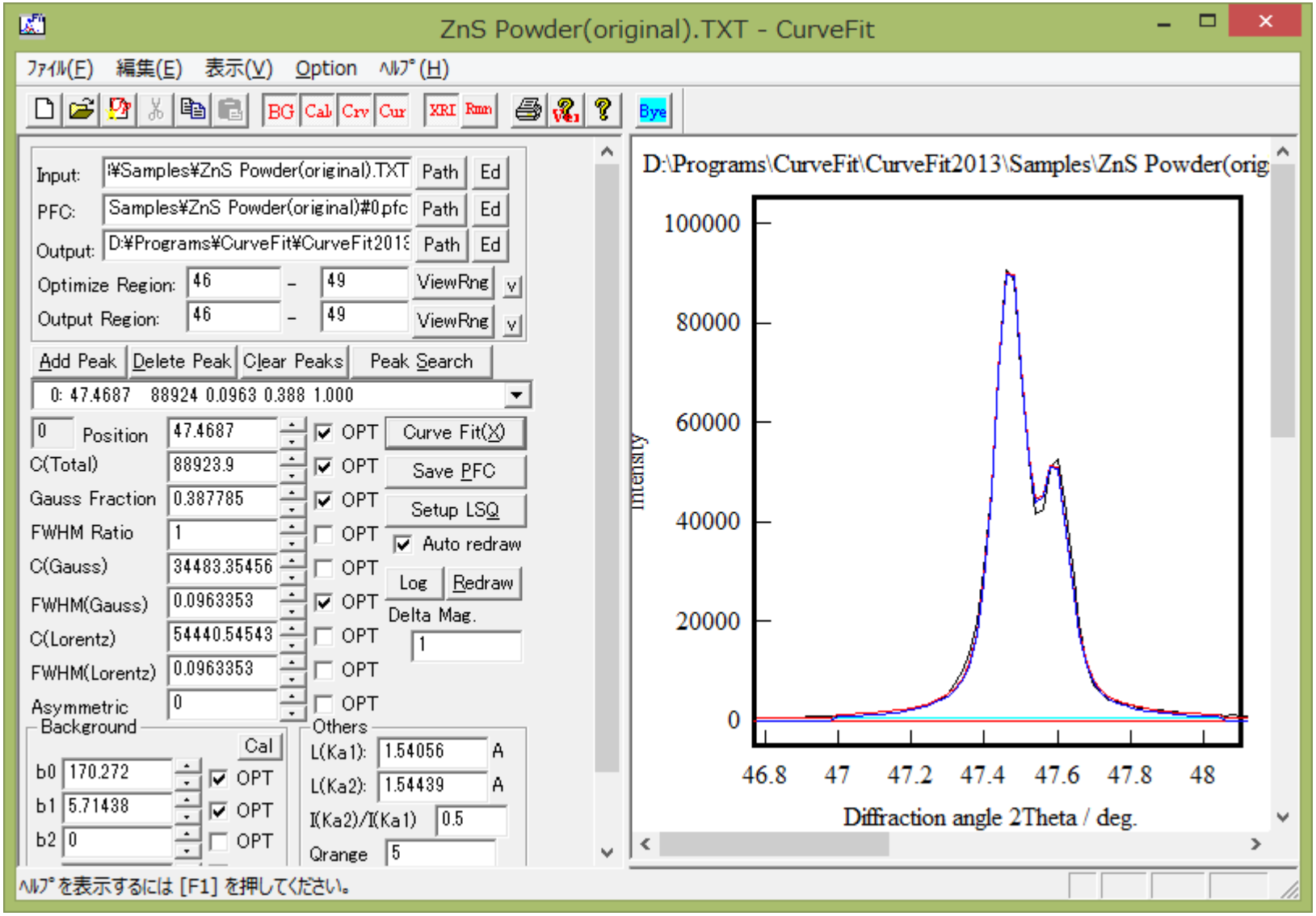


$$E = E_{\min} + 1/2 B_0 \left(V / V_0\right)^2$$

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

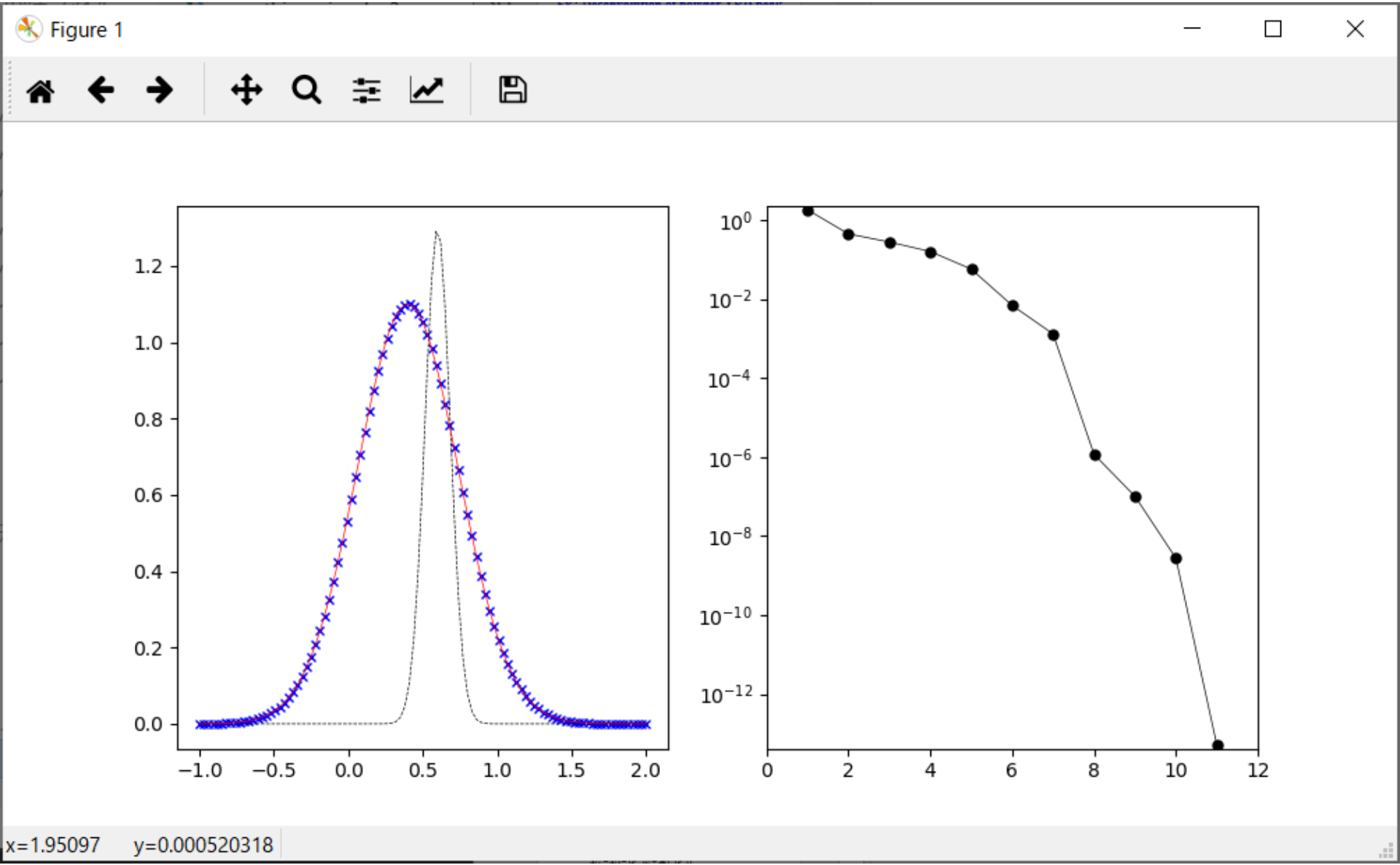
多変数最適化問題例: 粉末XRDのピーク分離

$K\alpha_1$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

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

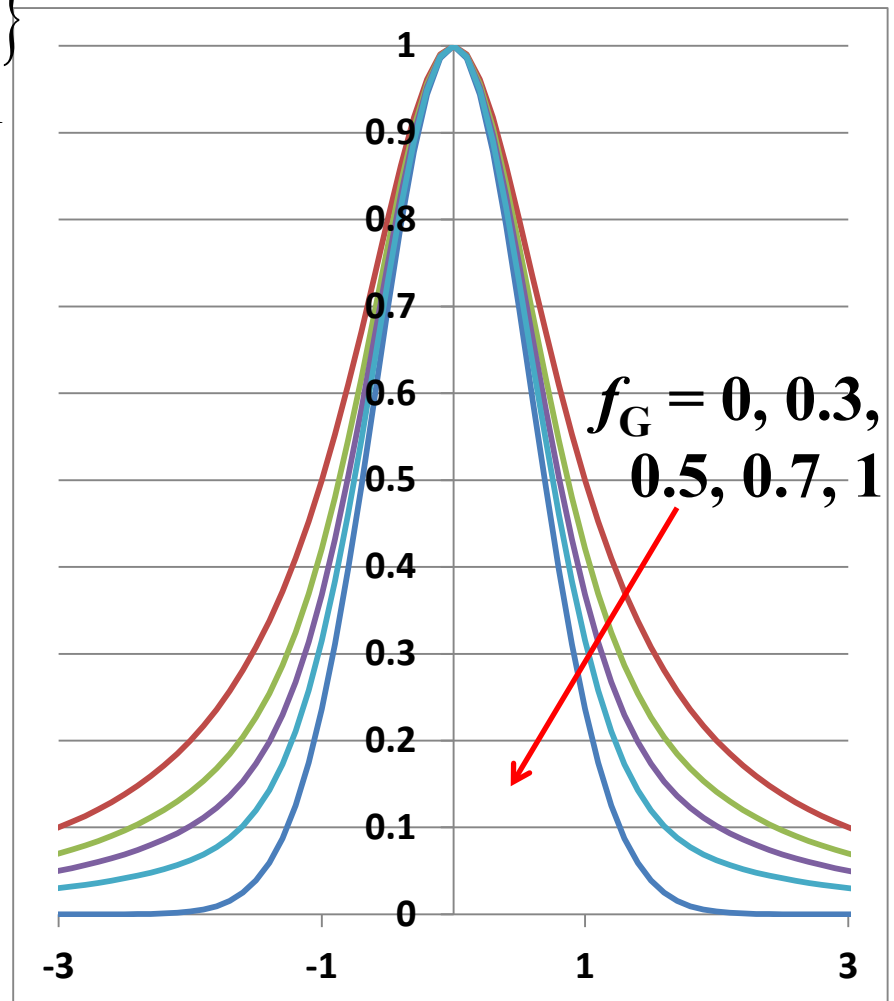
$$= \frac{a_V}{\pi} \int_{-\infty}^{\infty} \frac{\exp(-x'^2)}{a_V{}^2 + (x-x')^2} dx'$$

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.

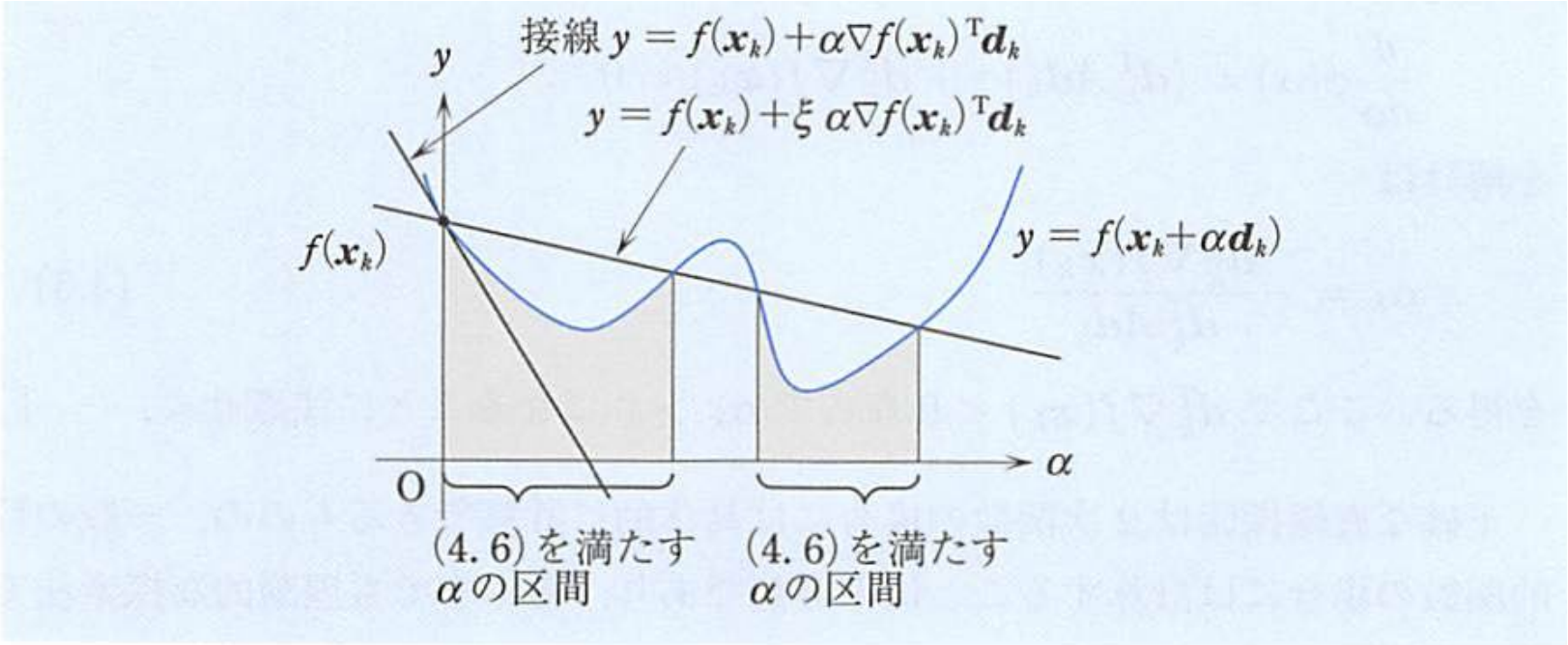


図 4.7　Armijo 条件 (4.6) を満たすステップ幅

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,

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

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

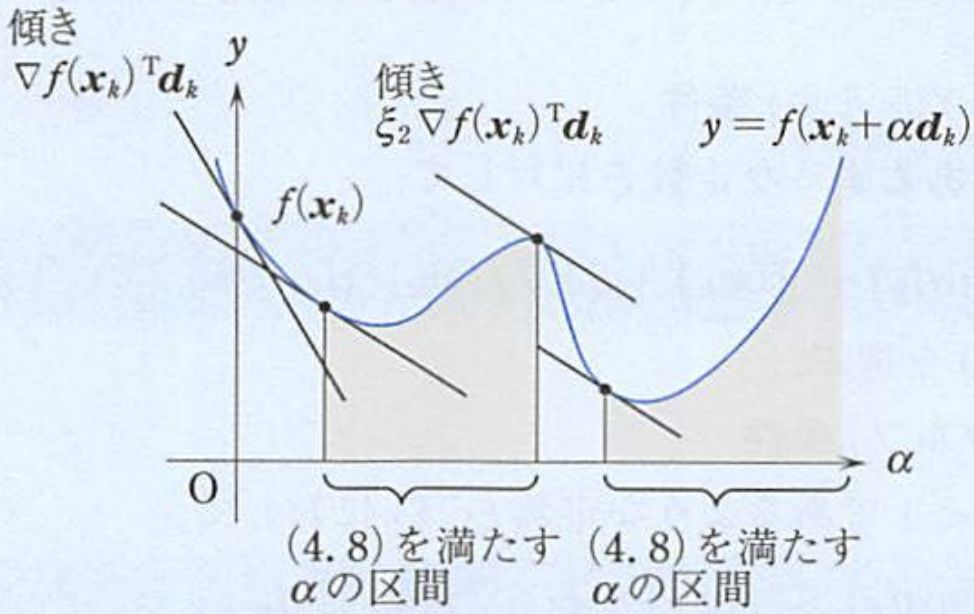


図 4.8　Wolfe 条件 (4.8) を満たすステップ幅

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)

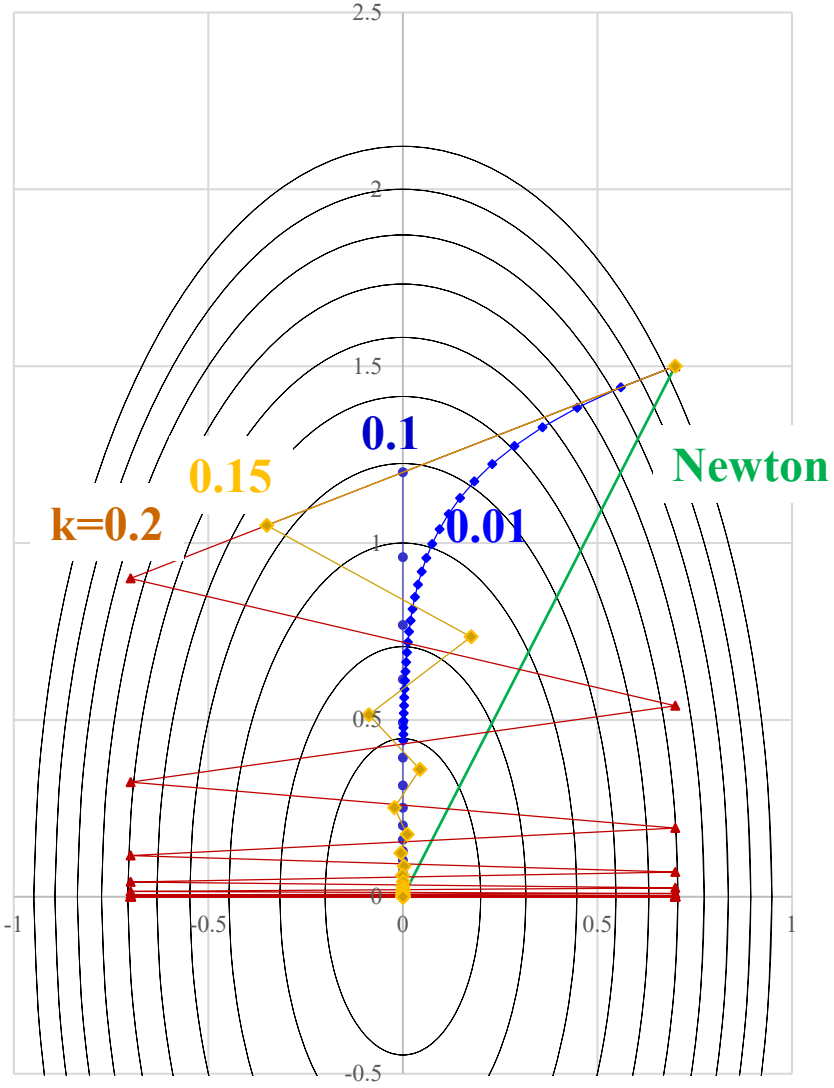
  0.2, 0.15: 振動しながら収束

  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

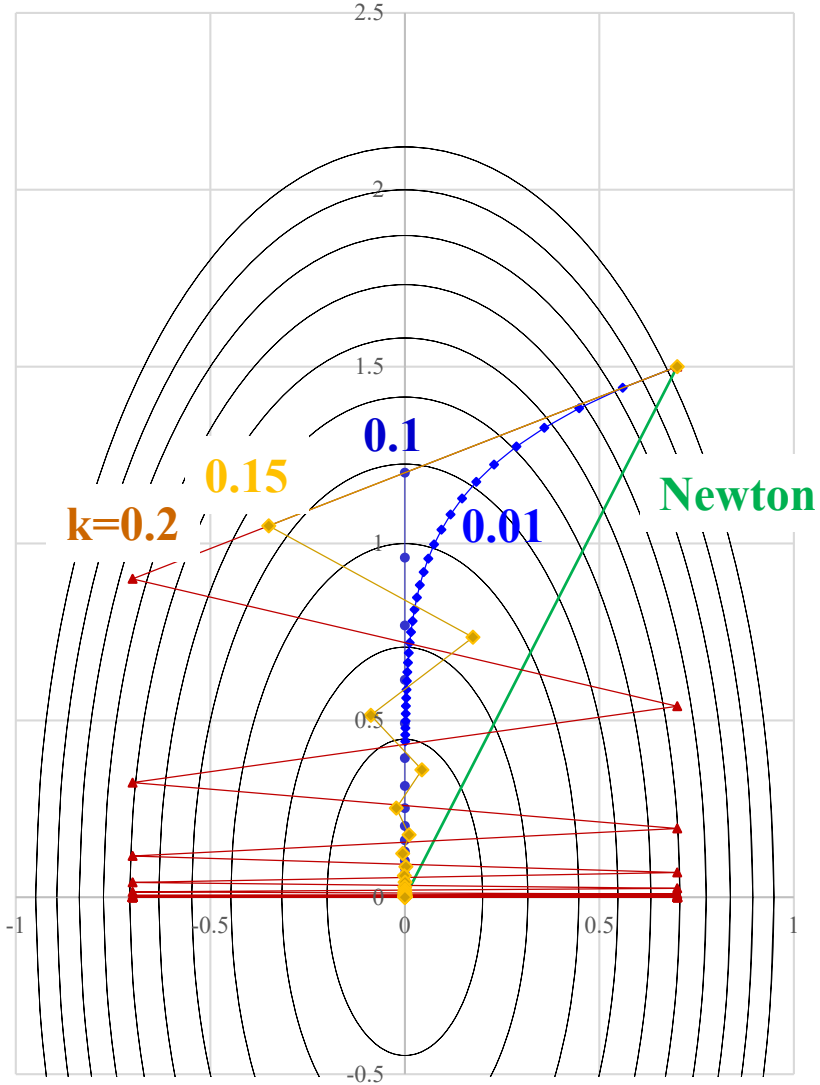


最急降下法 (Steepest Descend) 法

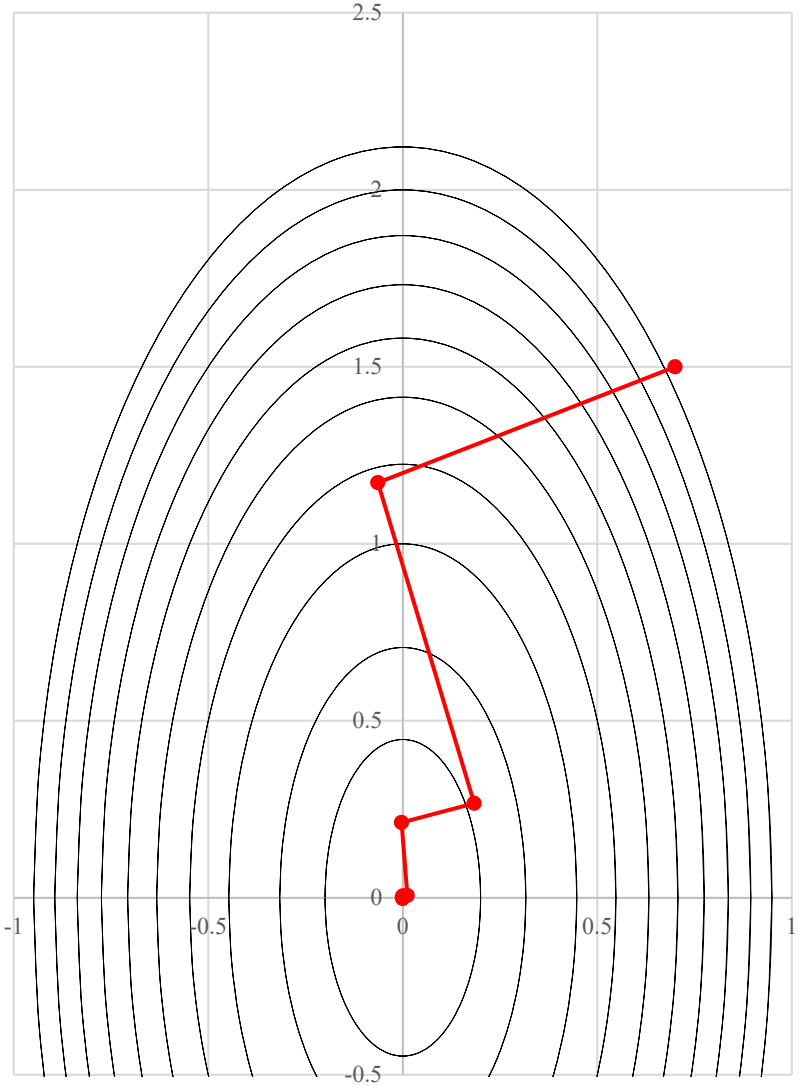
矢部博, 工学基礎 最適化とその応用, 数理工学社 (2006)

効率的な方法: 各ステップの α を直接探索法で決定する

直接探索を行わない場合



直接探索を行う場合



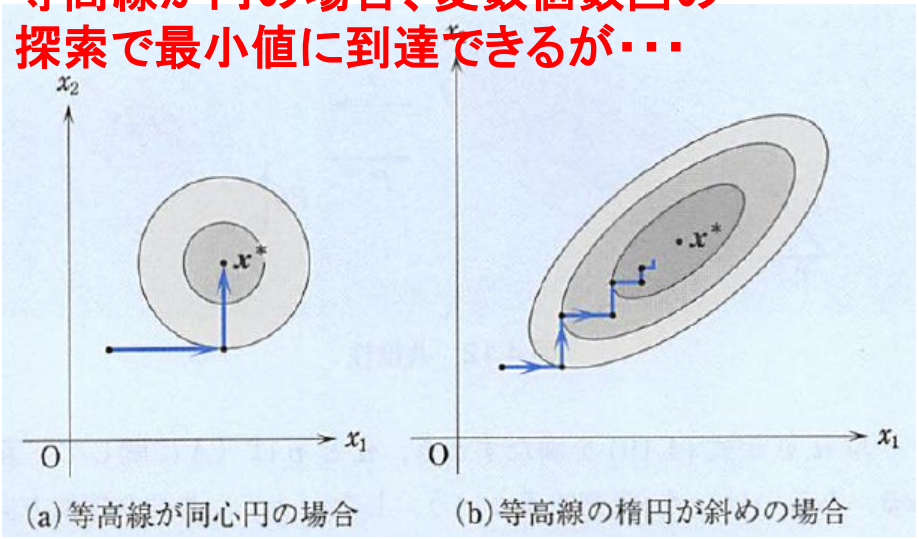
共役勾配 (Conjugate Gradient) 法

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

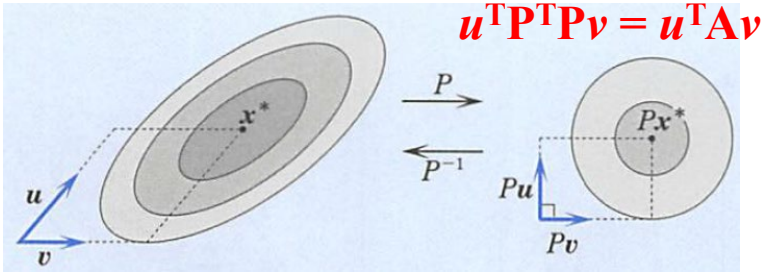
・共役な探索方向に沿って正確な直線探索を実行していけば、
有限回の反復で2次関数の最小解に到達することが期待される

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

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

と近似すると、

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

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

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

Levenberg-Marquart法

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

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

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

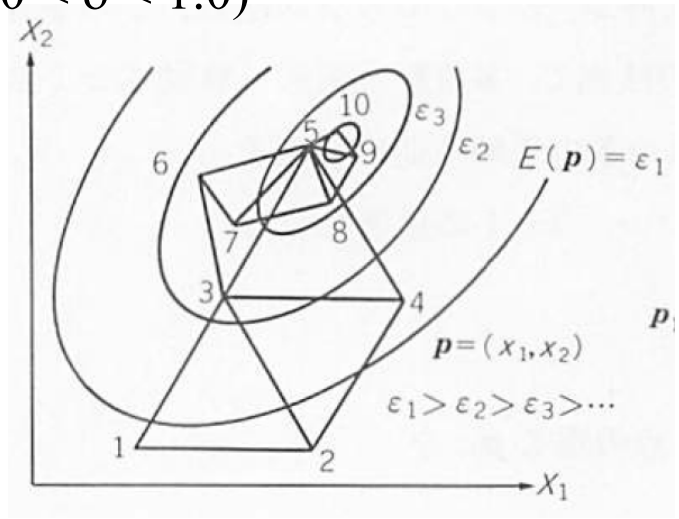
(i) 鏡映 : $\boldsymbol{x}_R = \boldsymbol{x}_1 + \alpha(\boldsymbol{x}_G - \boldsymbol{x}_1)$ (たとえば $\alpha = 2$)

(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 $0 < \delta < 1.0$)

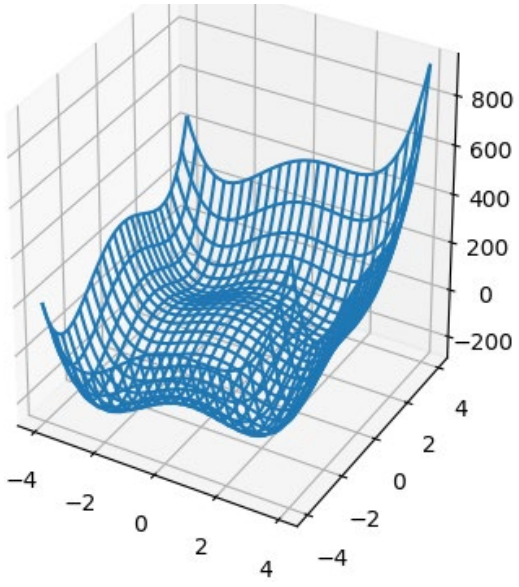
4. (i)~(iv)のうちで最初に $F(\boldsymbol{x}) < F(\boldsymbol{x}_1)$を満たす点を
   $\boldsymbol{x}_1$ と交換する。
   この操作を繰り返す



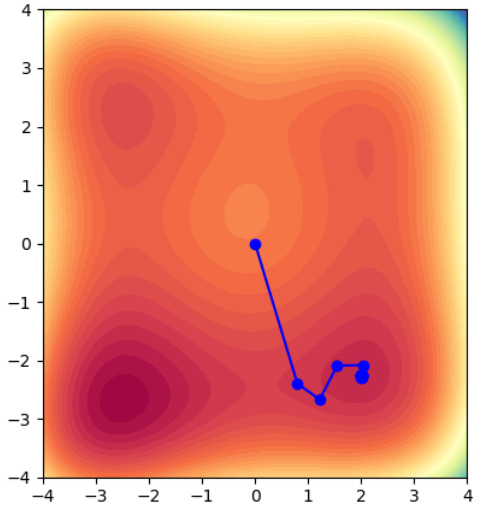
Comparison

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

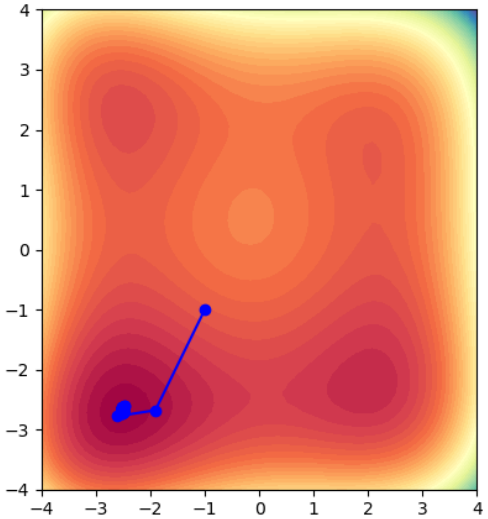
optimize-sd-cg2d-linesearch.py, optimize-newton-raphson2d.py



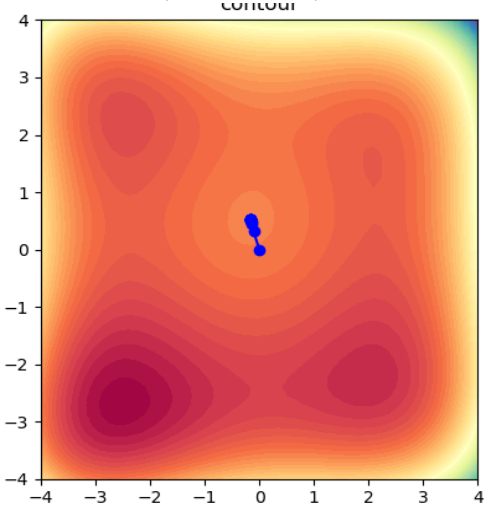
From (0.0 0.0) cg simple



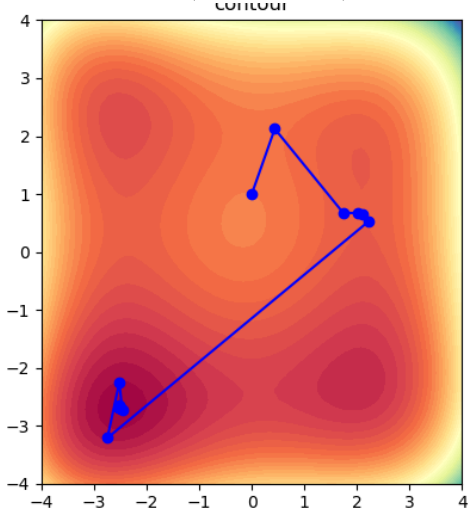
From (-1.0 -1.0) cg simple



From (0.0 0.0) Newton

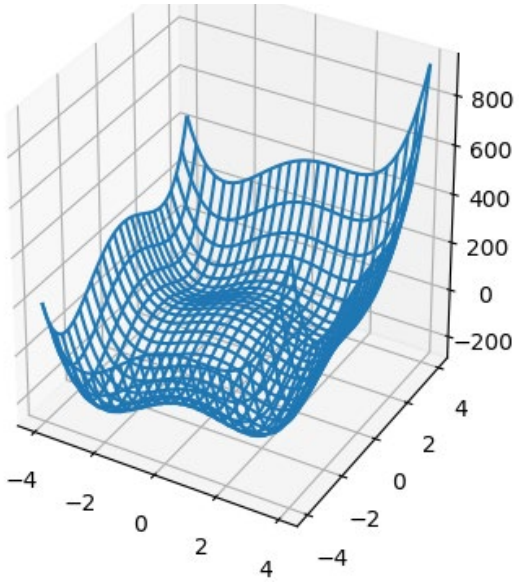


From (0.0 1.0) SD armijo

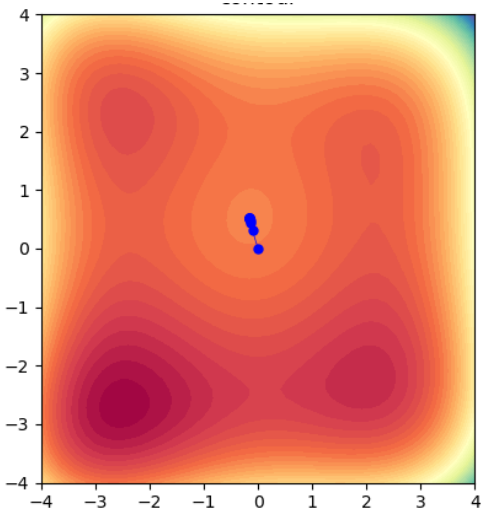


Comparison

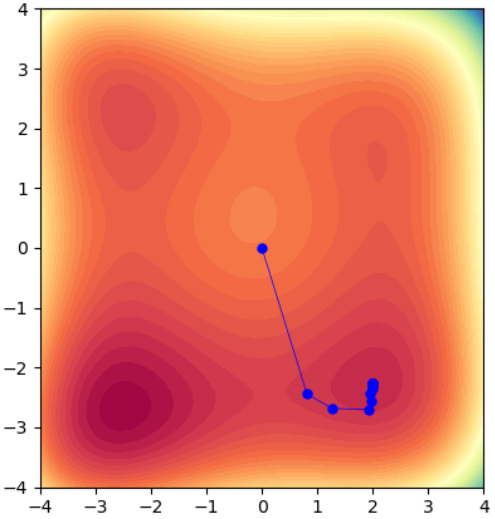
¥Programs¥python¥optimize.py in Programs.zip



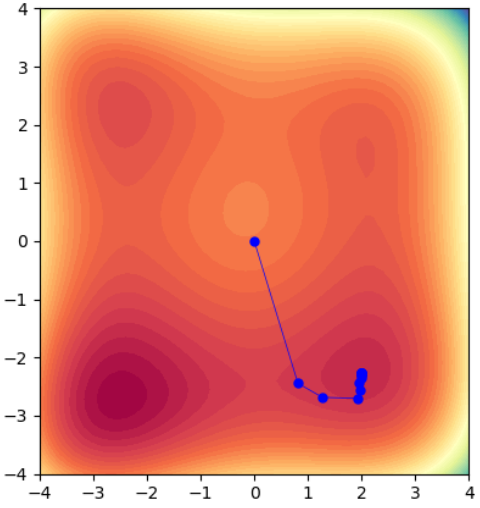
From (0.0 0.0) Newton



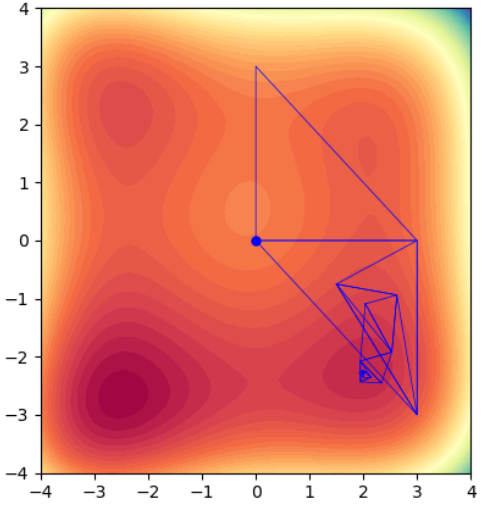
From (-1.0 -1.0) DFP golden



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

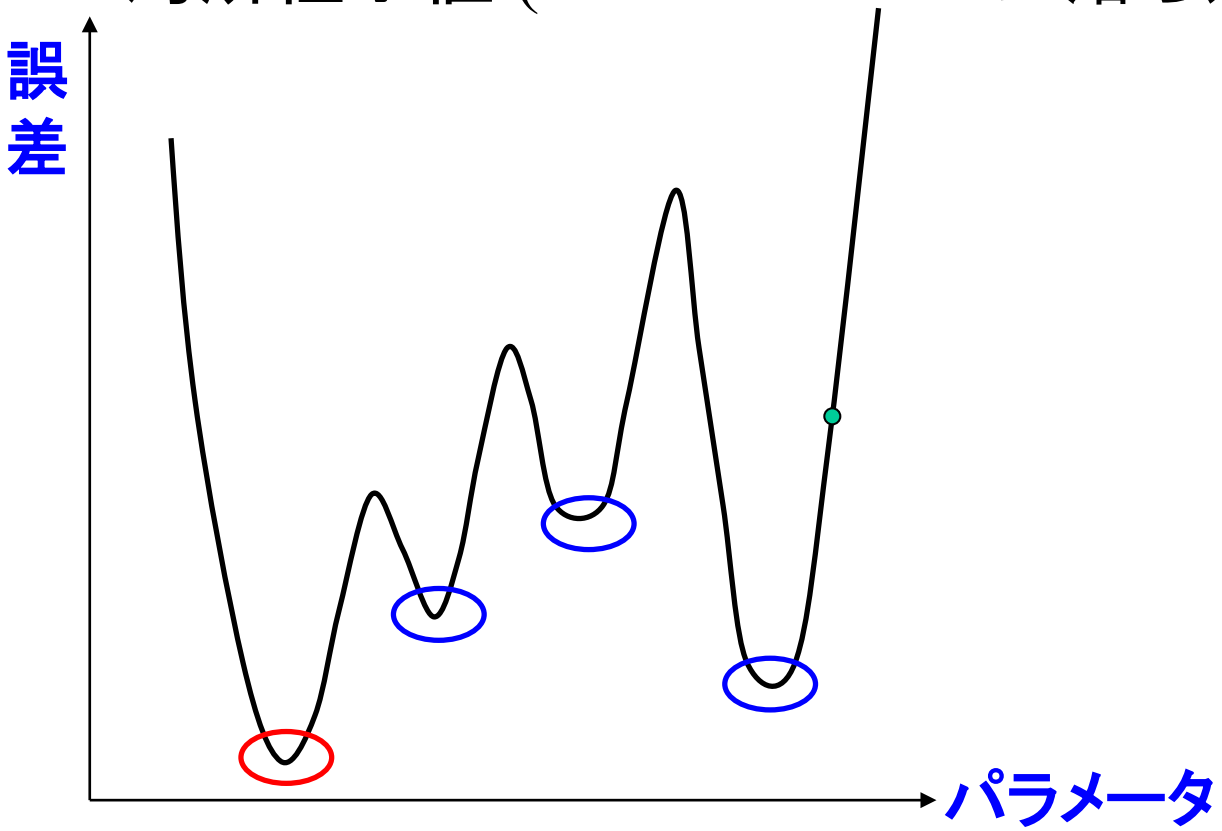
直接探索法

- **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)**