非線形最適化アルゴリズムの実装と動作例

奥野　貴之

本稿では，無制約非線形最適化問題に対するアルゴリズムの中でも基本である最急降下法，ニュートン法，ネステロフの加速勾配法 (NAG) について解説する．またそれらの Python による実装と実行結果の例についても述べる．

キーワード：最急降下法，ニュートン法，ネステロフの加速勾配法 (NAG)，Python

1. はじめに

本稿では，次の最適化問題

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \tag{1}$$

を考える．ただし，目的関数 $f\colon \mathbb{R}^n \to \mathbb{R}$ は連続的微分可能な実数値関数とする．同特集号の田中未来氏の記事 [1] では，この問題の大域的最適解と局所的最適解の基本的性質とそれらを求めるためのいくつかのアルゴリズム（解法）の解説を行った．本稿は田中氏の記事の続編であり，アルゴリズムの実装・振る舞いについてより詳しく解説を行うことを目的としている．しかしながら，独立した記事としても読むことができるように執筆したため田中氏の記事と重複した内容も多くある．

数値計算分野におけるアルゴリズムは直接法・反復法のどちらかに大別される．非線形最適化アルゴリズムのほぼすべては反復法として分類され，解きたい問題の解に妥当な条件の下で収束するような無限点列 $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3, \ldots$ を生成するように設計されており，現在点（アルゴリズムによっては過去の点）がもつ関数値などの情報を用いて次の点を生成することを繰り返すのである．

本稿では，問題 (1) のような制約条件が存在しない最適化問題に対する最急降下法，ニュートン法，ネステロフの加速勾配法について解説する．いずれも問題 (1) の停留点，すなわち $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$ である点 $\boldsymbol{x}$ を発見することを目的としたものである．関数 f が凸関数であるならば，停留点は問題 (1) の大域的最適解である．そうでないならば一般には最適解であることの保証はないものの，ほかの手法と組み合わせることによって問題 (1) の優れた解を見つけることができる．

本稿を通して，$\boldsymbol{x} \in \mathbb{R}^n$ は縦ベクトルを表し，$\boldsymbol{x}^\top$ でベクトル $\boldsymbol{x}$ の転置を表す．$\nabla f(\boldsymbol{x}) \in \mathbb{R}^n$ は f の勾配とし，その第 i 成分は f の x_i に関する偏微分 $\frac{\partial f(\boldsymbol{x})}{\partial x_i}$ である．$\nabla^2 f(\boldsymbol{x}) \in \mathbb{R}^{n\times n}$ は f のヘッセ行列とし，その第 (i,j) 成分は f の 2 階偏微分 $\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j}$ である．また $\boldsymbol{x} \in \mathbb{R}^n$ に対して $\|\boldsymbol{x}\| := \sqrt{\sum_{i=1}^n x_i^2}$ とする．

2. 最急降下法

2.1 最急降下法とステップサイズ

最急降下法はしばしば勾配法ともよばれ，その k 回目の反復において点 $\boldsymbol{x}^k$ は

$$\boldsymbol{x}^{k+1} := \boldsymbol{x}^k - \alpha_k \nabla f(\boldsymbol{x}^k)$$

で更新される．$\alpha_k > 0$ はステップサイズとよばれ，後述するように α_k の決定過程はアルゴリズムの性能に大きく関わる．$-\nabla f(\boldsymbol{x})$ は最急降下方向とよばれ，局所的に見て点 $\boldsymbol{x}$ から動いたときに目的関数 f の値が最も小さくなる方向である[1]．

最急降下方向に進めば目的関数値が減少することは保証されているものの，進み過ぎれば目的関数値が現在点よりも増加（悪化）してしまう可能性がある．そこで進み幅を調節するのが上の α_k の役目である．ところが α_k の定め方は決して自明ではない．関数値を確実に減少させるには，（その程度は問題に依存してしまうものの）できるだけ小さく設定すればよいが，それだと遅々として解に向かって進まない．かといって，速く進むために α_k を大きくとれば前述したように目的関数値が悪くなってしまう．そうした α_k をうまく自動的に決める方法としてバックトラッキング法またはアルミホ (Armijo) の直線探索がよく使われる．その機構と周辺の説明は文献 [2–4] に預けてしまうとし

おくの　たかゆき
成蹊大学理工学部理工学科
〒 180–8633 東京都武蔵野市吉祥寺北町 3–3–1
takayuki-okuno@st.seikei.ac.jp

[1] これはコーシー・シュワルツ不等式と f の 1 次のテイラー展開から正当化される．田中氏の記事 [1] も参照．

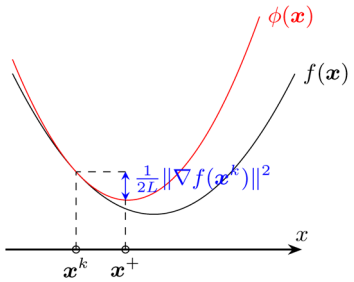

図 1　$\phi(\boldsymbol{x})$ と $f(\boldsymbol{x})$ のグラフのスケッチ

て，この手法はとても実用的で著者も普段から活用している．反面，数値的に不安定になりやすいうえに関数値の評価回数が多くなる傾向があるので，α_k の良い値を前もってわかっていることに越したことはない．

実は，ある正の定数 $L>0$ が存在して，関数 f が L-平滑となる場合は $\alpha_k \le 1/L$ とすればよいことが知られている．L-平滑性の定義は次のとおりである．

定義　2.1.　微分可能な関数 $h:\mathbb{R}^n \to \mathbb{R}$ に関して，$\|\nabla h(\boldsymbol{x}) - \nabla h(\boldsymbol{y})\| \le L\|\boldsymbol{x}-\boldsymbol{y}\|$ $(\boldsymbol{x},\boldsymbol{y}\in\mathbb{R}^n)$ である正の定数 $L>0$ が存在するとき h を L-平滑とよぶ．

たとえば，2 次関数は典型的な L-平滑な関数である．関数 f が L-平滑であるとき，$\alpha_k \le 1/L$ で定めれば目的関数値は必ず減少し，とくに $\boldsymbol{x}^+ := \boldsymbol{x}^k - L^{-1}\nabla f(\boldsymbol{x}^k)$ ならば

$$f(\boldsymbol{x}^+) \le f(\boldsymbol{x}^k) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}^k)\|^2 \qquad (2)$$

であることが知られている．この式を幾何学的に解釈してみよう．実は，2 次関数

$$\phi(\boldsymbol{x}) := f(\boldsymbol{x}^k) + \nabla f(\boldsymbol{x}^k)^\top(\boldsymbol{x}-\boldsymbol{x}^k) + \frac{L}{2}\|\boldsymbol{x}-\boldsymbol{x}^k\|^2$$

を定義すると図 1 のように $\phi(\boldsymbol{x})$ のグラフは $f(\boldsymbol{x})$ のグラフに $\boldsymbol{x}^k$ で接しつつ上側に位置している．そのため $f(\boldsymbol{x}^k)=\phi(\boldsymbol{x}^k) \ge \phi(\boldsymbol{x}^+) \ge f(\boldsymbol{x}^+)$ が成り立ち，簡単な計算で $\phi(\boldsymbol{x}^k) - \phi(\boldsymbol{x}^+) = \|\nabla f(\boldsymbol{x}^k)\|^2/(2L)$ となるので式 (2) が成り立つ．

2.2　理論的性質

つぎに最急降下法の性能に関する理論的な性質について述べよう．ただし，ここではこの 10 年で研究が盛んになっている f が L-平滑な関数である場合について述べる．より一般の関数についての結果はたとえば文献 [4]（4.5 節）を参照されたい．各 k について $\alpha_k = 1/L$ であるとし，$\boldsymbol{x}^*$ を大域的最適解とする．$f(\boldsymbol{x}^0), f(\boldsymbol{x}^1), f(\boldsymbol{x}^2), \dots$ が $f(\boldsymbol{x}^*)$ で下から抑えられるような単調減少列であることに注意すると $\boldsymbol{x}^+ = \boldsymbol{x}^{k+1}$ とおいた式 (2) から $\lim_{k\to\infty}\nabla f(\boldsymbol{x}^k) = \boldsymbol{0}$ がわかり[2]，最急降下法の大域的収束性が示された．最悪反復計算量については，K 回目の反復でつぎの式が成り立つ：

$$\min_{0\le k\le K}\|\nabla f(\boldsymbol{x}^k)\|^2 \le \frac{L}{K}(f(\boldsymbol{x}^0) - f(\boldsymbol{x}^*)). \qquad (3)$$

これは式 (2) において同じく $\boldsymbol{x}^+ = \boldsymbol{x}^{k+1}$ とおいて，$k=0,1,2,\dots,K-1$ に関する各不等式たちの両辺を足し合わせることで導出できる．そして，式 (3) は $\|\nabla f(\boldsymbol{x}^k)\| \le \varepsilon$ を達成するには，多くても $\varepsilon^{-2}L(f(\boldsymbol{x}^0) - f(\boldsymbol{x}^*))$ 回の反復を行えばよいことを意味する．関数 f が凸関数である場合には，K 回目に関して

$$f(\boldsymbol{x}^K) - f(\boldsymbol{x}^*) \le \frac{L}{2K}\|\boldsymbol{x}^0 - \boldsymbol{x}^*\|^2 \qquad (4)$$

が成り立ち，目的関数値は最適値に K^{-1} のオーダーで近づく．この性質は後述の加速勾配法との性能比較の中で重要となる．式 (4) の証明は比較的平易にできるものの日本語文献もあまりないようなので本稿末の付録に載せた．より詳細な解析については文献 [5] (Theorem 3.4) や文献 [6] (Chapter 4) を参考にされたい．

2.3　数値例

最急降下法の数値例を述べる．目的関数として $f(x_1,x_2) = 0.1x_1^4 + 2x_1^2 - 4x_1x_2 + 8x_2^2$ とする．この関数は凸関数であり，最適解は $(0,0)^\top$ である．それに対して，初期点を $(2,3)^\top$ とし，各反復の固定ステップサイズとして $10^{-1}, 10^{-2}, 10^{-3}$ をそれぞれとったときの各反復における目的関数の変化を表示したものが図 2 であり，$\|\nabla f(x_1,x_2)\|$ の様子を対数目盛で示したのが図 3 である．ただし，$\|\nabla f(x_1,x_2)\| \le 10^{-8}$ となるか反復回数が 100 回に達するかのいずれかのときに反復を打ち切っている．図 3 から，ステップサイズを大きくとっておけば収束が速いことがわかる．実際 $\alpha_k = 10^{-1}$ のときに 70 回程度で $\|\nabla f(x_1,x_2)\| \le 10^{-8}$ を達成している．ちなみに，これより α_k を大きめにとると目的関数値が増加しつづけ発散してしまった．図 3 からは $\|\nabla f(x_1,x_2)\|$ が直線的に 0 に収束していくのがわかる．これが 1 次収束性[3]の特徴である．

図 4 は f の等高線とともに生成された点列をプロットしたものである．ステップサイズ $\alpha_k = 10^{-1}$ の場合，最初は大きく進み，解に近づくにつれてその幅も

[2] ただし，多くの教科書では L-平滑性より緩い仮定下で同性質について述べていることには注意すべきである．
[3] 十分大きな k に対して $\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| \le c\|\boldsymbol{x}^k - \boldsymbol{x}^*\|$ となる定数 $0<c<1$ が存在すること．

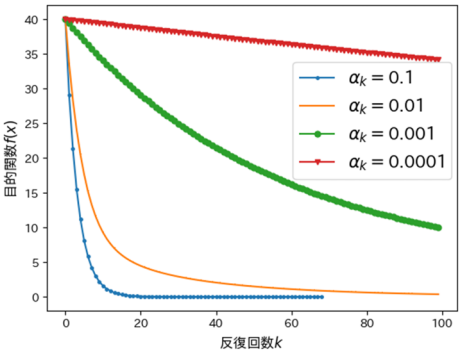

図 2　反復回数 vs 目的関数値（最急降下法）

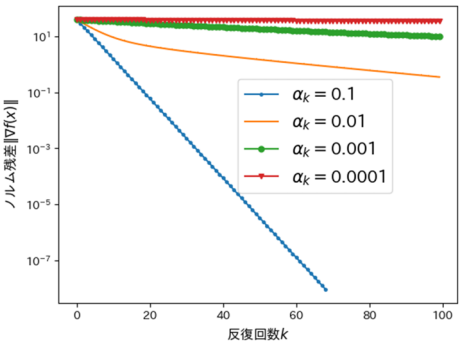

図 3　反復回数 vs $\|\nabla f(\boldsymbol{x}^k)\|$（最急降下法）

小さくなり収束していく様子がわかる．ほかのステップサイズに関しては，最適解へと続く一つの曲線をなぞりながら着実に進んでいるように見える．これは微分方程式 $\dot{\boldsymbol{x}}(t) = -\nabla f(\boldsymbol{x})$ で特徴づけられた曲線 $\boldsymbol{x}(t)$ であると予想する．具体的な結果は割愛するが，バックトラッキング法も実装したところパラメータの設定次第で $\alpha_k = 0.1$ の場合と比べて反復回数に関して比肩することも著しく劣る場合もあったが，やはり関数値の評価回数が多い傾向がみられた．しかし，ステップサイズを自動的に見積もれるのはやはり実用的である．最後に Python のコードの一例を図 5 に示す第 16 行でステップサイズ α_k の値を設定している．

3.　ニュートン法

3.1　ニュートン方向

ニュートン法は関数 f を $\boldsymbol{x}^k$ において 2 次のテイラー近似をした式

$$f(\boldsymbol{x}^k) + \nabla f(\boldsymbol{x}^k)^\top (\boldsymbol{x} - \boldsymbol{x}^k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^k)^\top \nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x} - \boldsymbol{x}^k) \qquad (5)$$

は $\boldsymbol{x}$ の 2 次関数であり，その停留点 $\boldsymbol{x}$ を次の点として $\boldsymbol{x}^k$ を更新する手法である．すなわち $\nabla^2 f(\boldsymbol{x}^k)\boldsymbol{d} = -\nabla f(\boldsymbol{x}^k)$ の解 $\boldsymbol{d}$ を用いて $\boldsymbol{x}^k + \boldsymbol{d}$ を次の点とする．$\nabla^2 f(\boldsymbol{x}^k)$ が半正定値対称行列であれば，$\boldsymbol{d}$ は関数 (5) の厳密な最小解である．$\nabla^2 f(\boldsymbol{x}^k)$ が逆行列をもつと仮定すれば，$\boldsymbol{d}$ は一意的に定まり $\boldsymbol{x}^k$ は

$$\boldsymbol{x}^{k+1} := \boldsymbol{x}^k - \nabla^2 f(\boldsymbol{x}^k)^{-1} \nabla f(\boldsymbol{x}^k)$$

と更新される．このときの $\boldsymbol{d}$ をニュートン方向とよび，この更新を繰り返すのが基本的なニュートン法である．最急降下方向 $-\nabla f(\boldsymbol{x}^k)$ が関数 (5) においてヘッセ行列を単位行列に置き換えてできる $\boldsymbol{d}$ であることに注意すれば，ニュートン方向は f の形状をより正確にとらえた方向であるといえる．ゆえに最急降下法よりも問題 (1) の解により迅速に収束することが期待される．実際，f が 2 次関数の場合には最急降下法が一般に複数回の反復を必要とするのに対してニュートン法は 1 回の反復で解に到達する．ただし，ニュートン方向を

図 4　点列の振る舞い（最急降下法）

```
1  import numpy as np
2
3  def val_f(x): #目的関数
4      val = 0.1*x[0]**4 + 2*x[0]**2 - 4*x[0]*x
           [1] + 8*x[1]**2
5      return val
6
7  def grad_f(x): #目的関数の勾配
8      grad = np.array([
9              0.4*x[0]**3 + 4*x[0] - 4*x[1],
10             -4*x[0] + 16*x[1]
11     ])
12     return grad
13
14 x = np.array([2., 3.]) #初期点xk
15 k = 0 #反復回数
16 alp = 1.0e-1    #ステップサイズの設定
17
18 while (k < 100):
19     x = x - alp * gradf #点xkの更新
20     gradf = grad_f(x)
21     normgf = np.linalg.norm(gradf, ord=2)
22     if(normgf <= 1.0e-8):
23       break
24     k += 1
25
26 print('x = '+str(x))
```

図 5　最急降下法のコード例

計算するためには連立 1 次方程式を解く必要があり，機械学習分野で現れる変数の数が多いような問題でその実行は難しい．これに対処するための古典的方法の一つとして，各連立 1 次方程式を共役勾配法で近似的に解く手法 [7] などが挙げられるが，これは長い研究・発展を経ており，Python の数値計算ソフトウエア Scipy [8] でも実装されている．

3.2 理論的性質

ニュートン法の特筆すべき性質は 2 次収束性[4]であり，最急降下法がもつ 1 次収束性よりも優れた収束性能をもつ．

ただし，ニュートン法はヘッセ行列が正則であるような $\boldsymbol{x}^*$ の十分近くに初期点 $\boldsymbol{x}^0$ をとれば収束が保証されているものの，大域的収束性は備えていない．つまり初期点によっては収束に失敗する．たとえば，$-x^6/6 + x^4/4 + 2x^2$ に対して $x = 1$ を初期点としてニュートン法を行うと 1 と -1 の間で無限に振動が起きて収束しない．最急降下法と同様にステップサイズを導入すればよいのではないかと思うかもしれないが，それはうまくいくとは限らない．なぜならばヘッセ行列 $\nabla^2 f(\boldsymbol{x}^k)$ が正定値であるとは限らず，その場合そこでのニュートン方向は目的関数値を減少させる方向（降下方向）とは限らないからである．ニュートン法に大域的収束性を備えさせるための古典的な手法として信頼領域法と準ニュートン法があるが，それは文献 [4] を参照されたい．近年では 3 次正則化をした関数 (5) の最小化を逐次的に行う 3 次ニュートン法 [9] も主流な方法の一つになりつつある．

3.3 数値例

最急降下法で用いた関数の最小化問題にニュートン法を適用した場合の数値例を示す．図 6 は，$\|\nabla f(\boldsymbol{x}^k)\|$ の変化を示したものであり，比較のため前節の最急降下法で最も性能がよかった $\alpha_k = 0.1$ の場合もあわせてプロットしている．ニュートン法は数回の反復で $\|\nabla f(\boldsymbol{x})\| \leq 10^{-11}$ を満たす解を見つけている．また最急降下法がほぼ直線的な収束を見せているのに対し，ニュートン法は減少の傾きが反復ごとに大きくなっている．これが 2 次収束性の特徴である[5]．最後にニュートン法の Python コードの例を図 7 に示す．ただし `val_f`, `grad_f` の定義までは最急降下法のコードと同一なので誌面の都合上省略している．

[4] 十分大きな k に対して $\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| \leq C\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2$ となる $C > 0$ が存在すること．

[5] より厳密には超 1 次収束性 [4] の特徴である．

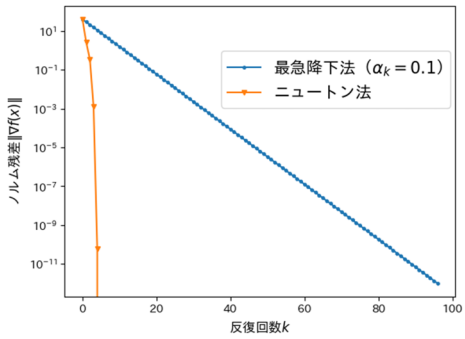

図 6 反復回数 vs $\|\nabla f(\boldsymbol{x}^k)\|$（ニュートン法）

```
def hess_f(x): #fのヘッセ行列
    hess = np.array([[1.2*x[0]**2 + 4,-4],
                     [-4,16]])
    return hess

x = np.array([2., 3.])
k = 0

while k < 100:
    val = val_f(x)
    grad = grad_f(x)
    norm_grad = np.linalg.norm(grad, ord=2)
    H = hess_f(x) #ヘッセ行列の計算
    d = np.linalg.solve(H, -grad) #ニュートン方向の計算
    x = x + d #点の更新
    k += 1
    if norm_grad <= 1.0e-12: #勾配ノルムが十分小さければ終了
        break

print('x = '+str(x))
```

図 7 ニュートン法のコード例

4. 加速勾配法

4.1 ポテンシャルからの NAG の導出

この節では f を L-平滑な凸関数に限定して加速勾配法[6]の中でもネステロフの加速勾配法（Nesterov's accelerated gradient method，以下 NAG）[11] について解説するが，非凸関数などより広いクラスへの拡張もされている [5]．さて，式 (4) からわかるように最急降下法（勾配法）では各反復での目的関数値 $f_k = f(\boldsymbol{x}^k)$ が最適値 $f_* = f(\boldsymbol{x}^*)$ に k^{-1} のオーダーで収束していく．NAG はそのオーダーを k^{-2} に改善したものであり，実用性にも優れている．ここでは NAG をポテンシャル（またはリャプノフ関数）とよばれる概念を用いて導いてみよう．

まずポテンシャルは最急降下法に対しても定義される．式 (4) の証明は 付録 A に掲載しているが，その中の $\mathcal{E}_k = k(f_k - f_*) + L\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2/2$ がポテンシャルにあたる．証明で示しているように，任意の整

[6] Polyak [10] によるヘビーボール法も有名である．

数 $k \geq 1$ について $\mathcal{E}_0 \geq \mathcal{E}_1 \geq \cdots \geq \mathcal{E}_{k-1} \geq \mathcal{E}_k$ が成り立つ．これに明らかな二つの式 $\mathcal{E}_k \geq k(f_k - f_*)$ と $\mathcal{E}_0 = L\|\boldsymbol{x}^0 - \boldsymbol{x}^*\|^2/2$ を組み合わせて k で両辺を割ることで結局，式 (4) を得るのである．このことから，次の夢と希望が生まれる．たとえば $\tilde{\mathcal{E}}_k := k^2(f_k - f_*) + L\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2/2$ のような量を新たなポテンシャルとして単調減少させるアルゴリズムならば k^{-2} の速度で最適値に収束するはずであると $\cdots$ それは正しい方針である．しかし，残念ながら $\tilde{\mathcal{E}}_k$ のような単純な修正ではそうした（少なくても勾配情報のみを用いた）アルゴリズムを生み出すことはできない．天下り的だが，$\boldsymbol{x}^k, \boldsymbol{z}^k \in \mathbb{R}^n$ が手元にあるとしたときにつぎの形の更新を行うアルゴリズムを考える：

$$\boldsymbol{y}^k := \tau_k \boldsymbol{x}^k + (1-\tau_k)\boldsymbol{z}^k, \tag{6}$$

$$\boldsymbol{x}^{k+1} := \boldsymbol{y}^k - \frac{1}{L}\nabla f(\boldsymbol{y}^k), \tag{7}$$

$$\boldsymbol{z}^{k+1} := \boldsymbol{z}^k - \frac{\beta_k}{L}\nabla f(\boldsymbol{y}^k). \tag{8}$$

ここで τ_k, β_k は各反復で設定される非負のパラメータである．$\tau_k \equiv 1$ ならば各 $\boldsymbol{x}^k$ はステップサイズを $1/L$ とした最急降下法が生成する点列にほかならない．そして NAG も実はこのアルゴリズムの一つである．このアルゴリズムに対して，$A_k \geq 0$ としてポテンシャル

$$\widehat{\mathcal{E}}_k := A_k\,(f_k - f_*) + \frac{L}{2}\|\boldsymbol{z}^k - \boldsymbol{x}^*\|^2$$

を定義する．実は，

$$\begin{aligned}\widehat{\mathcal{E}}_{k+1} - \widehat{\mathcal{E}}_k \leq & \frac{\beta_k^2 - A_{k+1}}{2L}\|\nabla f(\boldsymbol{y}^k)\|^2 \\ & + (A_{k+1} - A_k - \beta_k)\,\nabla f(\boldsymbol{y}^k)^\top(\boldsymbol{y}^k - \boldsymbol{x}^*) \\ & + \left(A_k - \frac{\tau_k}{1-\tau_k}\beta_k\right)\nabla f(\boldsymbol{y}^k)^\top(\boldsymbol{y}^k - \boldsymbol{x}^k)\end{aligned}$$

が成り立つ[7]．$\widehat{\mathcal{E}}_{k+1} \leq \widehat{\mathcal{E}}_k$ であるためには，最も単純なのは上の三項がつねに 0 になること，すなわち $\beta_k^2 - A_{k+1} = 0, A_{k+1} - A_k - \beta_k = 0, A_k - \frac{\tau_k}{1-\tau_k}\beta_k = 0$ となればよい．ここから

$$A_{k+1} = A_k + \frac{1 + \sqrt{4A_k + 1}}{2} \tag{9}$$

である．すると β_k, τ_k は

$$\beta_k = \frac{1 + \sqrt{4A_k + 1}}{2},\ \tau_k = \frac{A_k}{A_{k+1}} \tag{10}$$

と表せる．この A_k, β_k, τ_k を用いれば $\widehat{\mathcal{E}}_k$ が単調に減少することがわかる．さて，本題の NAG は $A_0 = 0, \boldsymbol{z}^0 = \boldsymbol{x}^0$ として式 (6),(7),(8) に式 (10) を組み合わせたものにほかならない．A_k は式 (9) で更新を行う．

なお，NAG にはさまざまな形 [6]（4.4 節）があってここで述べたものはその中の一つにすぎない．関数 f が μ-強凸性[8]をもつ場合に適した NAG も提案されている．それは，文献 [6] の Algorithm 14, 15, 16 を参照いただきたい．

4.2 理論的性質

$A_k(f_k - f_*) \leq \widehat{\mathcal{E}}_k \leq \widehat{\mathcal{E}}_0 = L\|\boldsymbol{z}^0 - \boldsymbol{x}^*\|/2$ であり，

$$f_k - f_* \leq \frac{L}{2A_k}\|\boldsymbol{z}^0 - \boldsymbol{x}^*\| \tag{11}$$

である．実は $A_0 = 0$ の下で $A_k \geq k^2/4$ を数学的帰納法で示すことができる．このことと式 (11) から

$$f_k - f_* \leq \frac{2L}{k^2}\|\boldsymbol{z}^0 - \boldsymbol{x}^*\| \tag{12}$$

となり，NAG は k^{-2} の速度で最適値に収束することがわかる．

4.3 数値例

最急降下法と同じ問題に対して同じ初期点から NAG を適用した．ただし，この関数は L-平滑な関数ではなく NAG の適用範囲外に見えるかもしれない．しかし，L-平滑性は最適解と点が生成される範囲で成り立っていれば十分で，L を少し大きめに見積もっていれば実用的には NAG の性能を享受できる[9]．今回は $L = 20$ とした．図 8 に実行結果を示す．各反復の $f_k - f_*$ を対数目盛りで表している．比較のためステップサイズを $0.1, 0.01$ とした最急降下法もプロットしている．図からわかるように，NAG は目的関数値を各反復で減少させるものではなく，玉の跳ね返りのような挙動を示す．再スタート法とよばれる技法が NAG に組み合わせてしばしば用いられるが，`NAG_Restart` はその振る舞いを示したものである．NAG における再スタート法は目的関数値の増加が起きたならば，$A_k = 0, \boldsymbol{z}^k = \boldsymbol{x}^k$ という初期化を行うものである．詳しくは文献 [12] を参照していただきたい．図 8 より目的関数値の増加が一度起こり，その後減少が続くことが繰り返されているのがわかる．ステップサイズを 0.1 とした最急降下法に NAG 単体は劣ってしまっているものの，再スタート付きの NAG は優れている．最後に図 9 に Python

[7] 次の不等式を用いる：$f(\boldsymbol{x}^{k+1}) \leq f(\boldsymbol{y}^k) - \frac{1}{2L}\|\nabla f(\boldsymbol{y}^k)\|^2$, $f(\boldsymbol{y}^k) - f(\boldsymbol{x}^*) \leq \nabla f(\boldsymbol{y}^k)^\top(\boldsymbol{y}^k - \boldsymbol{x}^*)$, $f(\boldsymbol{y}^k) - f(\boldsymbol{x}^k) \leq \nabla f(\boldsymbol{y}^k)^\top(\boldsymbol{y}^k - \boldsymbol{x}^k)$.

[8] μ-強凸性の定義については田中氏の記事 [1] の脚注 7 を参照していただきたい．

[9] バックトラッキング法を応用して L の値を各反復で見積もる手法もある．

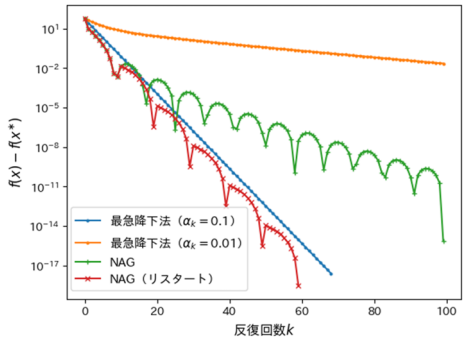

図 8　反復回数 vs $f(\boldsymbol{x}^k)-f(\boldsymbol{x}^*)$(NAG)

```
k = 0 #反復回数
x = np.array([2., 3.])   #初期点x0
z = x   #z0=x0
L = 20.   #Lの値
A = 0. #Akの初期値

while k < 100:
    grad = grad_f(x)
    norm_grad = np.linalg.norm(grad, ord=2)

    beta= (1+np.sqrt(4*A+1))/2     #
        betakの更新
    tau = A/(A+beta)     #taukの更新
    y = tau * x + (1-tau) * z   #ykの更新
    grady = grad_f(y)
    xpre = x      #xkの保存
    x = y - 1/L*grady   #xkの更新
    z -= beta*1/L*grady #zkの更新
    A += beta   #Akの更新
    k += 1

    if norm_grad <= 1.0e-12:
       break

    if val_f(x) > val_f(xpre): #再スタート
        A = 0
        z = x

print('x = '+str(x))
```

図 9　再スタート付き NAG のコード例

コードの例を示した．`beta`，`tau` がそれぞれ β_k，τ_k を意味している．24 行から 26 行で再スタート法を行っているが，ここをコメントアウトすれば通常の NAG である．

5.　最後に

本稿では，最急降下法，ニュートン法，ネステロフの加速勾配法に絞ってその構造と理論的性質，および Python によるコード例と実行例を示した．ここで紹介したくも誌面の都合上割愛したものとして，近接勾配法があるが，それに関してはたとえば日本語文献 [13]（4.3 節）が参考になるだろう．

参考文献

[1] 田中未来，“はじめよう非線形最適化の基礎，” オペレーションズ・リサーチ：経営の科学，**68**，pp. 607–615，2023.

[2] 金森敬文，鈴木大慈，竹内一郎，佐藤一誠，『機械学習のための連続最適化（機械学習プロフェッショナルシリーズ）』，講談社，2016.

[3] 山下信雄，『非線形計画法』，朝倉書店，2015.

[4] 矢部博，『工学基礎　最適化とその応用』，数理工学社，2006.

[5] G. Lan, *First-order and Stochastic Optimization Methods for Machine Learning*, Springer, 2020.

[6] A. Daspremont, D. Scieur and A. Taylor, *Acceleration Methods (Foundations and Trends(R) in Optimization)*, Now Publishers, 2021.

[7] R. S. Dembo and T. Steihaug, “Truncated-Newton algorithms for large-scale unconstrained optimization,” *Mathematical Programming*, **26**, pp. 190–212, 1983.

[8] P. Virtanen, R. Gommers, T. E. Oliphant, et al., “SciPy 1.0: Fundamental algorithms for scientific computing in Python,” *Nature methods*, **17**, pp. 261–272, 2020.

[9] Y. Nesterov and B. T. Polyak, “Cubic regularization of Newton method and its global performance,” *Mathematical Programming*, **108**, pp.177–205, 2006.

[10] B. T. Polyak, “Gradient methods for the minimisation of functionals,” *USSR Computational Mathematics and Mathematical Physics*, **3**, pp. 864–878, 1963.

[11] Y. Nesterov, “A gradient method of solving a convex programming problem with convergence rate $O(1/k^2)$,” *Soviet Mathematics Doklady*, **27**, pp. 372–376, 1983.

[12] B. O’Donoghue and E. Candes, “Adaptive restart for accelerated gradient schemes,” *Foundations of computational mathematics*, **15**, pp. 715–732, 2015.

[13] 寒野善博，『最適化手法入門』，講談社，2019.

付録 A.　式 (4) の証明

簡単のため $f_k := f(\boldsymbol{x}^k), f_* := f(\boldsymbol{x}^*)$ とおく．また $\mathcal{E}_k := k\,(f_k - f_*) + L\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2/2$ とおくと以下が成り立つ．

$$
\begin{aligned}
&k(f_{k+1}-f_*) \le k(f_k-f_*),\\
&f_{k+1}-f_* \le \nabla f(\boldsymbol{x}^k)^\top(\boldsymbol{x}^k-\boldsymbol{x}^*) - \frac{1}{2L}\|\nabla f(\boldsymbol{x}^k)\|^2,\\
&\frac{L}{2}\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^*\|^2 = \frac{L}{2}\|\boldsymbol{x}^k-\boldsymbol{x}^*\|^2\\
&\qquad\qquad - \nabla f(\boldsymbol{x}^k)^\top(\boldsymbol{x}^k-\boldsymbol{x}^*) + \frac{1}{2L}\|\nabla f(\boldsymbol{x}^k)\|^2.
\end{aligned}
$$

2 番目の不等式は $f_{k+1}-f_* \le f_k - f_* - (2L)^{-1}\|\nabla f(\boldsymbol{x}^k)\|^2$ と f の凸性から導かれる不等式 $f_k - f_* \ge \nabla f(x^k)^\top(x^k - x^*)$ を用いることで導くことができる．3 番目の等式は $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - L^{-1}\nabla f(\boldsymbol{x}^k)$ から得ることができる．これらの各辺を足すと $\mathcal{E}_{k+1} \le \mathcal{E}_k$ である．よって $\mathcal{E}_k \le \mathcal{E}_0$ だから $k\,(f_k - f_*) \le L\|\boldsymbol{x}^0 - \boldsymbol{x}^*\|^2/2$ となり式 (4) を得る．