NA 課題 1

05-251008 金田燈和

2025 年 11 月 19 日

$f(x) = x^3 - 3x + 3$ とする．$f(x) = 0$ の根を計算する．初期値 x_1 を -3 から 3 の範囲で動かして，以下のアルゴリズムを使う．アルゴリズムの収束は，更新の量が 10^{-8} 以下かつ $f(x)$ の絶対値が 10^{-8} 以下になったことで判断する．ただし，x の更新回数は 1000 を上限とする．

1. 単純なニュートン法について，x_1 の関数として，収束までに x を更新する関数をグラフに表せ．
   図 1 に示した．

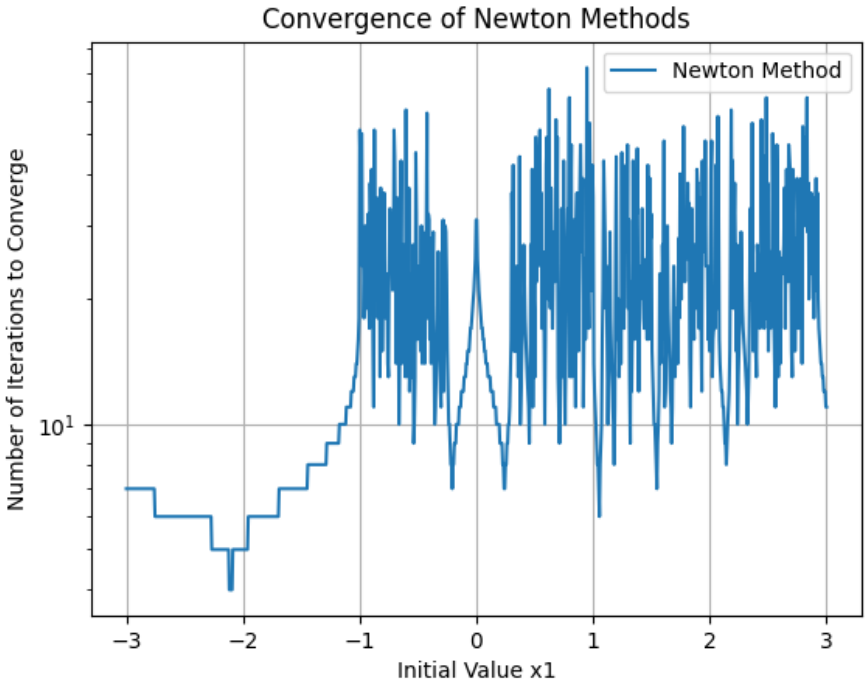

図 1　単純なニュートン法

2. $\beta = 0.2, \lambda = 2.0$ とした減速ニュートン法について，x_1 の関数として，収束までに x を更新する回数をグラフに表せ．

図 2 に示した.

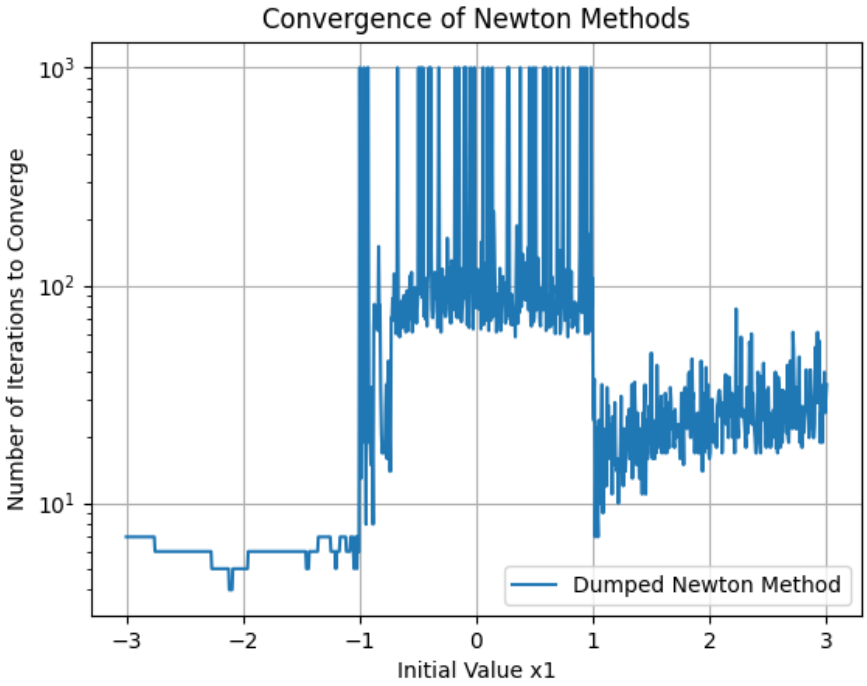

図 2　減速ニュートン法

3. 平野の変形ニュートン法について，x_1 の関数として，収束までの x を更新する回数をグラフに表せ．パラメータは任意に選択してよいが，x_1 に対しては定数とせよ．
   図 3 に示した．なおパラメータについては，$\beta = 0.2, \delta = 1.0$ とした．

以上の 3 つのグラフを重ねたものを図 4 に示す.

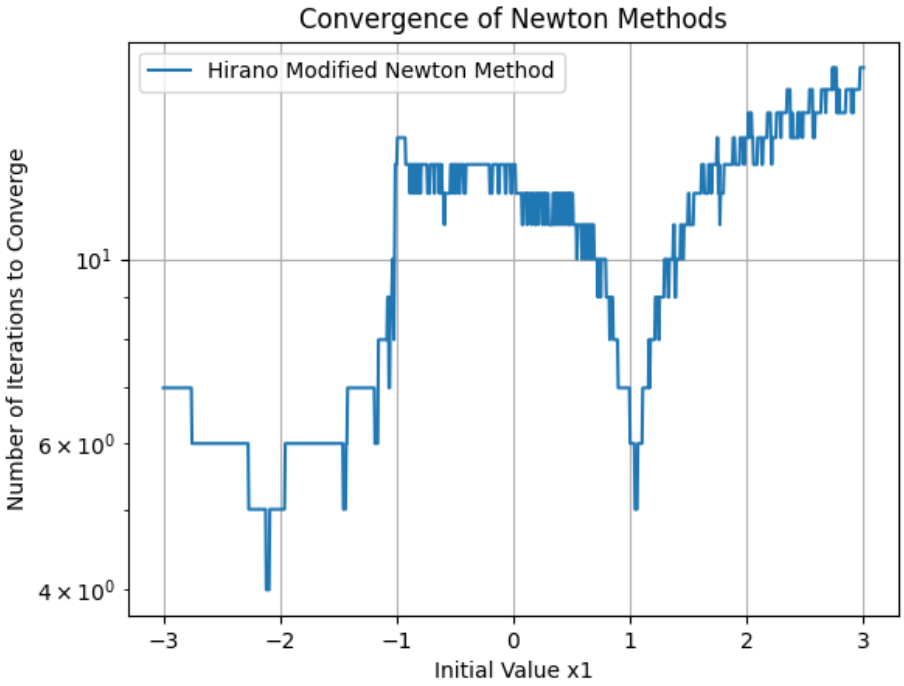

図 3　平野の変形ニュートン法

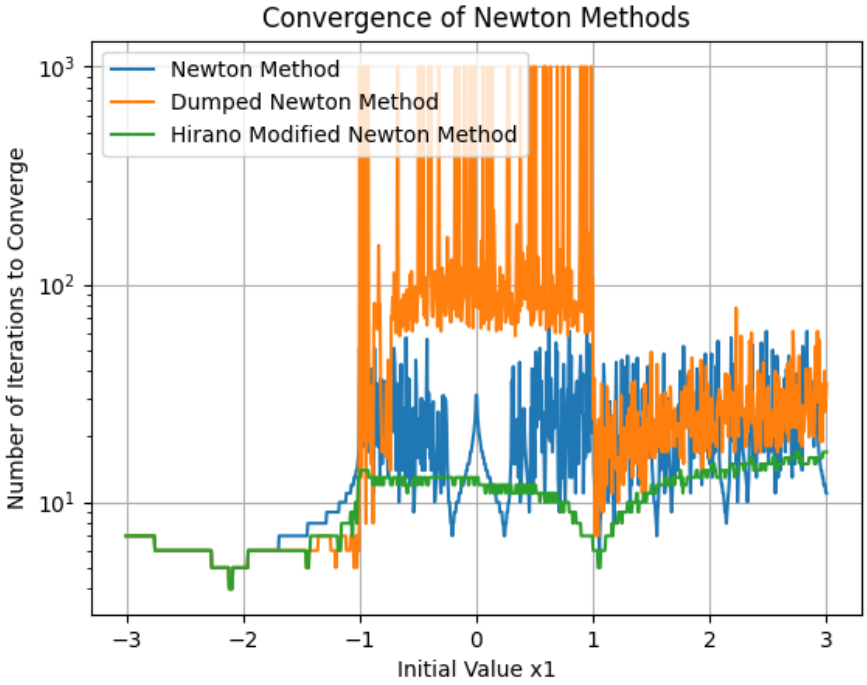

図 4　3 つのニュートン法の比較

実装コード

```python
import numpy as np

beta = 0.2
lambda_ = 2
delta = 1.0

def f(x):
    return x**3 - 3*x + 3

def f_prime(x):
    ans = 3*x**2 - 3
    if ans == 0:
        return f_prime(x + 1e-6)
    return ans

def f_primes(x,n):
    if n<=0:
        return f(x)
    if n==1:
        return f_prime(x)
    if n==2:
        ans=6*x
    if n==3:
        ans=6
    if n>3:
        ans=0
    if ans == 0:
        return f_primes(x + 1e-6,n)
    return ans

def newton_step(x):
    return  x - f(x) / f_prime(x)

def dumped_newton_step(x):
    i = 0
    e = np.abs(f(x))
    while True:
        x_new = x - lambda_**(-i) * f(x) / f_prime(x)
        if np.abs(f(x_new)) <= (1 - (1 - beta) * lambda_**(-i)) * e:
            return x_new
        i += 1

def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

def hirano_newton_step(x):
    a=[f_primes(x,i) / factorial(i) for i in range(4)]
    mu=1
```

```
    e=np.abs(a[0])
    while True:
        xi=[np.pow(-mu*a[0]/a[i]+0j,1/i) for i in range(1,4)]
        min_xi=min(xi,key=lambda x:np.abs(x))
        d=sum([a[i]*np.pow(min_xi,i) for i in range(4)])
        if np.abs(d)<= (1 - (1 - beta) * mu) * e:
            return x + min_xi
        mu/=delta+1

def newton_method(step, x0, max_iter=1000, tol=1e-8):
    trajectory = [x0]
    x = x0

    for _ in range(max_iter):
        x_new = step(x)
        trajectory.append(x_new)

        if np.abs(f(x_new)) <= tol and np.abs(x_new - x) <= tol:
            break
        x = x_new

    return len(trajectory)

import matplotlib.pyplot as plt

x1_values = np.linspace(-3, 3, 1000)
newton_iters = [newton_method(newton_step,x1) for x1 in x1_values]
dumped_newton_iters = [newton_method(dumped_newton_step,x1) for x1 in
    x1_values]
hirano_newton_step_iters = [newton_method(hirano_newton_step,x1) for x1 in
    x1_values]

plt.plot(x1_values, newton_iters, label='Newton Method')
plt.plot(x1_values, dumped_newton_iters, label='Dumped Newton Method')
plt.plot(x1_values, hirano_newton_step_iters, label='Hirano Modified Newton
    Method')
plt.xlabel('Initial Value x1')
plt.ylabel('Number of Iterations to Converge')
plt.yscale('log')
plt.legend()
plt.title('Convergence of Newton Methods')
plt.grid()
# plt.show()
plt.savefig('all_newton_methods_convergence.png')
```