数値解析　レポート

641163 的矢知樹

計算環境

CPU core i7-6500U　memory 8GB　Python3.6.3

問題 1

べき乗法により固有ベクトルを求める。

$$A_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 5 & -1 \\ 0 & 3 & 0 \end{pmatrix}$$

のときの各成分の推移を表したグラフが図 1 である。A_1の固有多項式は$\lambda^3 - 6\lambda^2 + 8\lambda - 3 = 0$、解は$\lambda = 1, \frac{5 \pm \sqrt{25-12}}{2}$である。絶対値最大の固有値が一つであり固有ベクトルに収束する。

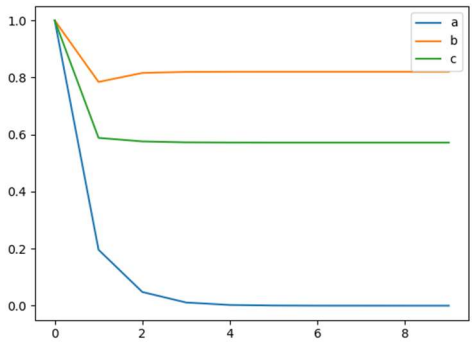
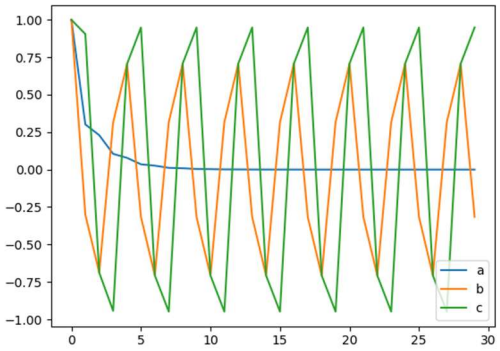


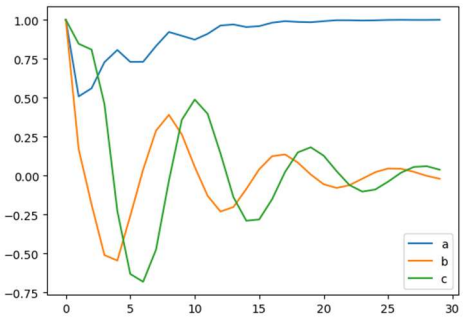
図 2 は

$$A_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 3 & 0 \end{pmatrix}$$

に対してべき乗法を適用した結果である。A_2の固有値は$\lambda = \pm i\sqrt{3}, 1$であり、絶対値最大の固有値が二つ存在するので収束しない。



A_2の固有値を+2 する、つまり$A = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & 3 & 2 \end{pmatrix}$とすると、3 が絶対値最大の固有値となる。べき乗法を適用すると右図のようになる。



固有ベクトル$v = {}^t(1 \quad 0 \quad 0)$に収束している。最大固有値と二番目に大きい固有値の比$\frac{|3|}{|2+i\sqrt{3}|} \approx 1.13$が、一に近いため収束が遅いことが見受けられる。

問題２

問題で与えられた、$\mathrm{N}\times\mathrm{N}$行列で対角成分c対角成分からひとつずれた要素は-1 である行列を$\mathrm{A_N^c}$と表記することにする。$\mathrm{A_N^c}$の条件数やスペクトル半径は固有値が代数的に計算できるため、容易に計算することが出来る。いかに考察に用いる命題を述べる

命題1. $\mathrm{A_N^c}$の固有値は$\mathrm{c}+2\cos\left(\frac{\mathrm{k\pi}}{\mathrm{N+1}}\right)$である$(\mathrm{k}=1\ldots\mathrm{N})$

証明

第二種チェビシェフ多項式を$\mathrm{U_n}(x)$で表すとする。

$\mathrm{A_N^0}$の固有多項式は$U_N(\frac{\lambda}{2})$であることを示す。帰納法を用いる. $\frac{\lambda}{2}=\mathrm{x}$とする。

$\mathrm{N}=1$のとき、$\det([2x])=2x$より良い。

$\mathrm{N}=2$のとき、$\det\begin{pmatrix}2x & 1\\ 1 & 2x\end{pmatrix}=4\mathrm{x}^2-1$より良い。

第二種チェビシェフ多項式は、$\mathrm{U_{n+2}}(x)=2x\,U_{n+1}(x)-U_n(x)$という漸化式を満たすが、一行目に関して余因子展開をすることで、A_N^0の固有多項式も同じ漸化式を満たすことが分かる。

第二種チェビシェフ多項式は$U_{n-1}(\cos(x))=\frac{\sin(nx)}{\sin(x)}$という関係を満たしており、固有多項式の根は $2\cos(\mathrm{x})\quad(\mathrm{s.t.}\sin\big((n+1)x\big)=0)$となる。
よって示された.

命題 2. 次の式が成立する

$$\mathrm{cond}(\mathrm{A_N^c})\approx\begin{cases}\frac{4}{\pi^2}\mathrm{N}^2\cdots(c=2)\\ \frac{c+2}{c-2}\cdots(c>2)\end{cases}$$

証明

固有値は$\mathrm{c}+\ 2\cos(\frac{\mathrm{k\pi}}{\mathrm{N+1}})$である。条件数は正規行列であれば$\left|\frac{\lambda_{\max}}{\lambda_{\min}}\right|$である。

$\mathrm{c}=2$のとき，$\lambda_{\max}\approx 4$，$\lambda_{\min}=2\left(1-\cos\left(\frac{\pi}{\mathrm{N+1}}\right)\right)=4\sin^2\left(\frac{\pi}{2(N+1)}\right)\approx\frac{\pi}{(N+1)^2}$より良い.

$\mathrm{c}\gg 2$のとき，$\lambda_{\max}\approx\mathrm{c}+2$，$\lambda_{\min}\approx\mathrm{c}-2$となり良い。

最適な SOR 法のハイパーパラメータωは、Jacobi 法の反復行列のスペクトル半径rが分かれば，$\omega=\frac{2}{1+\sqrt{1-\mathrm{r}^2}}$と書けることが知られている。ここで最適とはスペクトル半径が最小のものである。

命題 3. $\mathrm{A_N^c}=\mathrm{L}+\mathrm{D}+\mathrm{U}$として、$H=-D^{-1}(L+U)$とする。

$$\rho(\mathrm{H})=\frac{2}{\mathrm{c}}\cos\left(\frac{\pi}{N+1}\right)$$

証明

$H = \frac{1}{c}A_N^0$であるので,固有値は$\frac{2}{c}\cos\left(\frac{k\pi}{N+1}\right)$であり，スペクトル半径はすぐに上の式であると分かる．

この命題より Jacobi 法のスペクトル半径が分かり、SOR 法の最適なパラメータが分かる．

命題 4 . SOR 法の最適なハイパーパラメータは,

$$\omega = \frac{2}{1+\sqrt{1-\left(\frac{2}{c}*\cos\left(\frac{\pi}{N+1}\right)\right)^2}}$$

なおこのときのスペクトル半径は$\omega - 1$となる

事実

Gauss-Seidel 法のスペクトル半径は Jacobi 法の半径の二乗である。

実験

CG 法・Jacobi 法・Gauss-Seidel 法・SOR 法・Red-Black 法を python により実装した. Red-Black 法は SOR 法の並列性を高めるために成分を並べる順番を変えて偶数番目を過去の値を用いてすべて計算した後に奇数番目の値を今の値を用いて計算するものである．今回計算に python ライブラリである numpy を使用し、numpy は BLAS の実装として intel 製 CPU に最適化された intel MLK を用いており、効率的に並列計算することが期待できる。

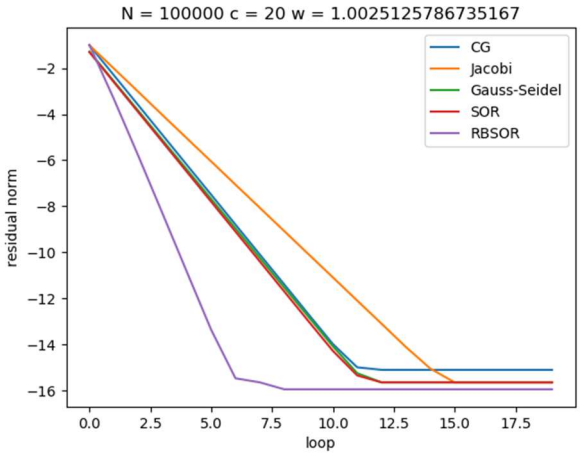
各手法について反復させたときの残差ベクトルの inf ノルムの減少する様子をグラフにプロットした。縦軸は常用対数である。

ベクトル b は$[-1,1]$の乱数列とした。SOR 法のパラメータは最適なものとした。

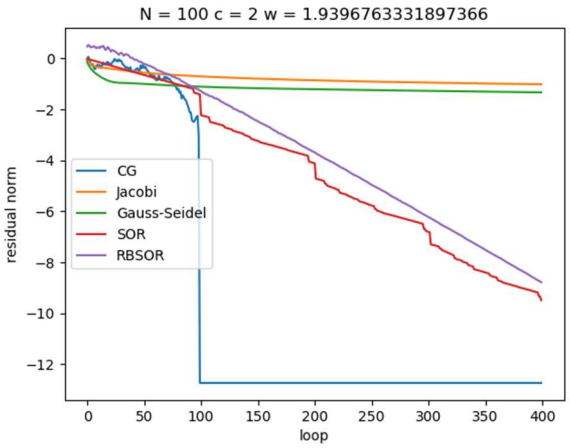
実験結果

$c = 20$のとき，非常に早く収束をし，残差ノルムが減らなくなる．これは数値を倍精度浮動小数点で持っているため丸め誤差の影響である．

収束の速度としては Red-Black 法が最速で、Jacobi 法が最も遅く、その他三手法がほぼ同じものとなった。



c=2 のとき、N=100 でループを 400 回行った。CG 法は、100 回目に大きく残差が減少した。Jacobi 法, Gauss-Seidel 法はほとんど収束せず、SOR 法, Red-Black 法は非常にゆっくりと収束する。



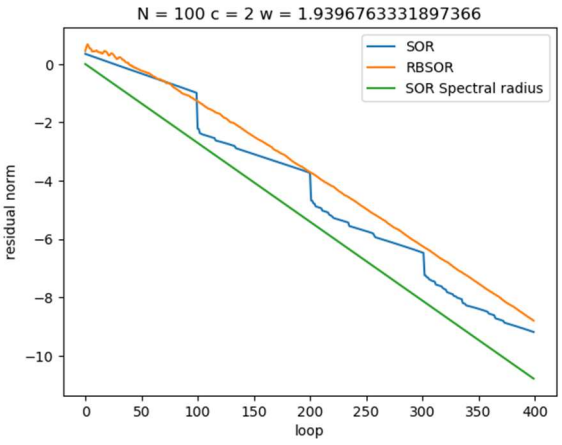
考察

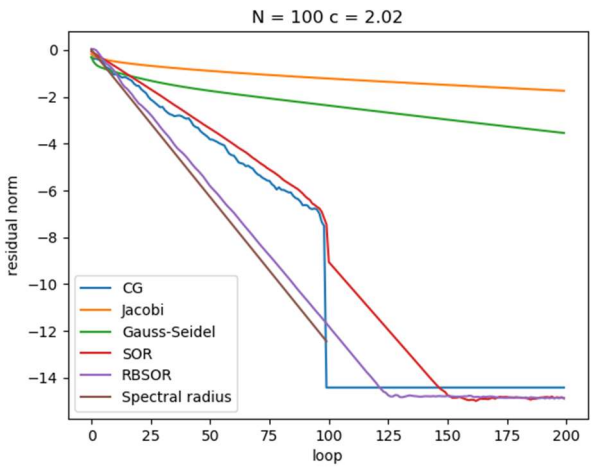
CG 法について。

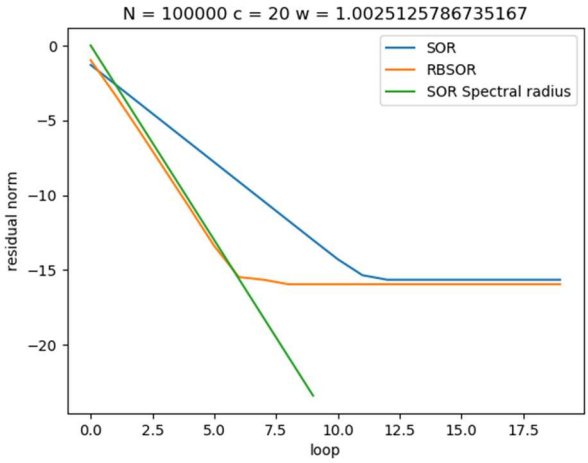
$c = 2$ のとき、CG 法は行列のサイズと同じ回数反復すると必ず真の解に収束するという直接法的側面が表れている。しかし反復法としてはほとんど使い物になっていない。これは条件数が大きいためである。$c = 2$のとき、条件数は N の二乗オーダーで増えることを示した。条件数が大きいと二つの要因で収束が遅くなる。CG 法は$x^t Ax$で入れたノルムが、$\left(\frac{1-\sqrt{cond}}{1+\sqrt{cond}}\right)^2$で小さくなっていく。条件数が大きくなるとノルムの収束は遅くなり、入っているノルムが尖がっているものになって inf ノルムや 2 ノルムとの差が大きくなってしまう。C=20 のとき、条件数は$\frac{11}{9}$で 1 に近く非常に条件が良い。SOR 法より収束速度が遅いが、SOR 法はパラメータが最適に調整されており、何も調整しないでこの収束性能は優秀な手法であるといえる。

Jacobi 法、Gauss-Seidel 法について
Gauss-Seidel 法のスペクトル半径は Jacobi 法のスペクトル半径の二乗であり、実験で得られた収束速度はそのスペクトル半径と一致した。C=2 のときスペクトル半径がほぼ 1 になってしまうことは上の命題から容易に分かる。

SOR 法、Red-Black 法について
これらの手法の収束速度は面白い挙動を示したので考察する。
反復法の収束速度は、反復行列のスペクトル半径の大きさによって計算することが出来る。よって各反復法の反復行列のスペクトル半径と実際の収束速度を比較してみることにする。残差ノルムの収束の速さと傾きがスペクトル半径の対数である直線を並べた。







SOR 法は$N = 100, c = 2$のとき 100 ループごとに急に残差が減少し、$N = 100000, c = 20$のときは Red-Black 法に比較して収束が遅い。$c = 2.02$のときは 100 回ループするまで遅く、その後 Red-Black 法と同じ速度で収束している。以下この差異を考察する。

次の事実よりこの差異は固有値分布によるものではない。

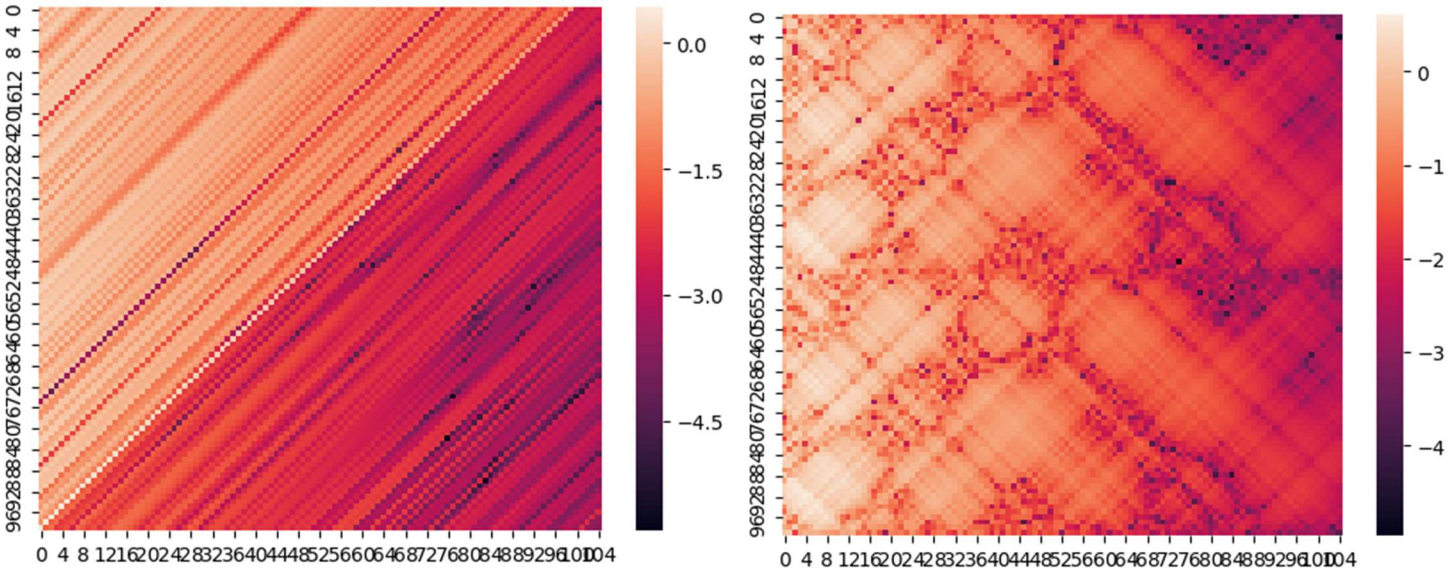
事実

反復行列の固有値分布は、SOR 法と Red-Black 法で等しい。

理由

SOR 法の反復行列の固有値は Jacobi 法の反復行列の固有値から計算でき、Red-Black 法は基底の順番を入れ替えた SOR 法であり、基底の順番を入れ替えても Jacobi 法の反復行列の固有値は変わらないため。

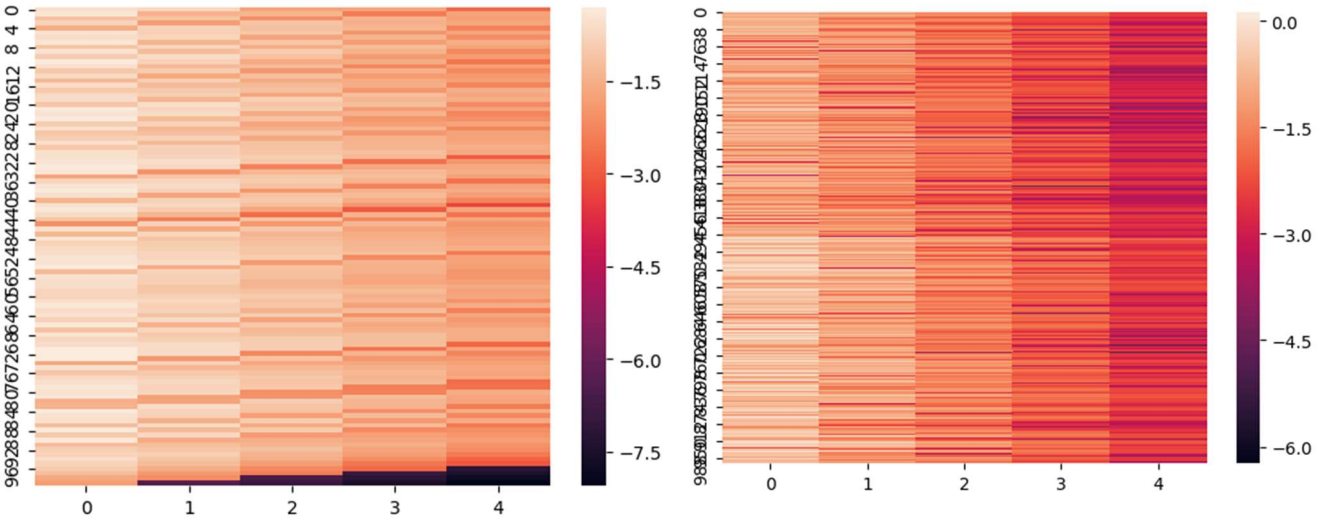
反復法は適切なノルムを入れるとスペクトル半径の速度で残差ノルムを減少させられることが知られており、反復行列と inf ノルムの相性が問題である。ここで残差ベクトルの成分ごとの大きさをヒートマップで見てみる。



この図は$N = 100, c = 2$のときの、残差ベクトルの減少の様子を成分ごとに見たものである。色で大きさを表す。一番左の列が初期残差であり、右に行くにつれて残差がだんだん小さくなる。一番右の列が 105 回反復した後である。左のヒートマップが SOR 法のもので、右が Red-Black 法である

SOR 法において、各成分が均一に残差が減少するわけではなく、残差が一つづつずれていくようなものであるため、最初はあまり誤差が減らず、100 回ループを回すと大きく残差が減ることが推察できる。Red-Black 方は残差ノルムの各成分が同等の速度で小さくなっている。

$N = 100, c = 2.5$のときに 5 回ループを回した時のものも比較してみる。左が SOR 法, 右が Red-Blach 法である。

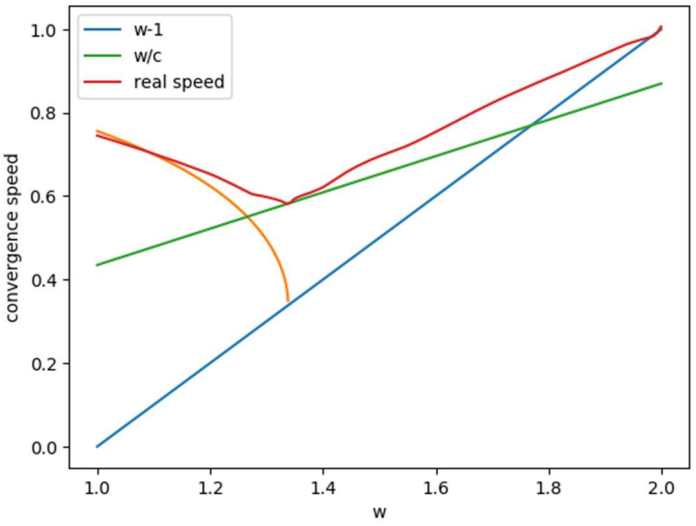


SOR 法では一部の成分の残差が急激に減少しているが、Red-Black 法では均一に誤差が減少している。適切なノルムを入れれば両者は同じ収束速度であっても、2 ノルムや inf ノルムを考えると、Red-Black 法の方が収束性が良いといえるであろう。

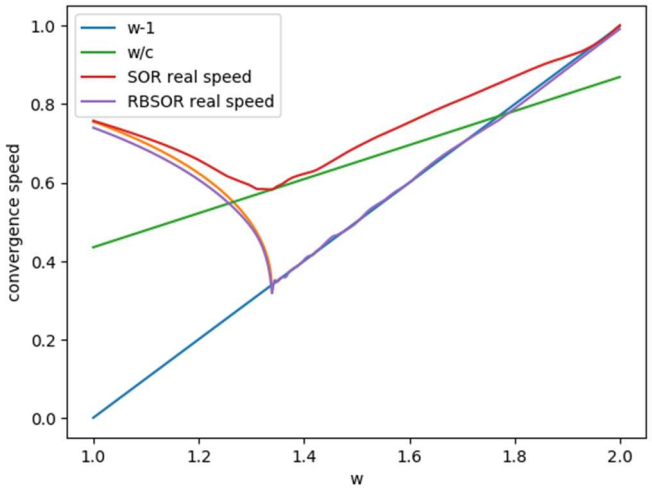
SOR 法における誤差が一つづつずれるときの倍率は反復行列の対角成分から一つずれた部分（添え字で書けば$\mathrm{A_{i,(i+1)}}$と書ける成分）を見ればよい。
SOR 法の反復行列は

$$(\mathrm{I}+\omega \mathrm{D}^{-1}L)^{-1}\big((1-\omega)I-\omega D^{-1}U\big)$$

とかけることから、Dが対角成分が$\frac{1}{c}$の行列であることに注意すれば、対角成分の隣には$\frac{\omega}{c}$が並んでいることが分かる。よって SOR 法において残差は$\frac{\omega}{c}$されて隣に移る。この値がスペクトル半径より大きいとき、収束速度は十分な回数（行列の一辺の大きさ程度）の反復を繰り返すまで収束が遅くなる。
実際に$\mathrm{N}=100,\mathrm{c}=2.3$のときに確かめたのが左のグラフとなる。10 回反復から 25 回反復までのノルムの減少の速度を線形回帰で求めた。オレンジと青の直線は Jacobi 法の反復行列のスペクトル半径から求めた SOR 法の最大固有値である。二次方程式の解として現れるため、放物線と直線を組み合わせたものとなる。



Red-Black 法がほぼスペクトル半径と同じ収束速度で収束している一方、SOR 法は緑の直線を下回ることが出来ていない。
ただωの最適値はほぼ同じになるようである。



以上より、ラプラス方程式を離散化したような偏った行列に対して SOR 法を適用するときは Red-Black 法のような手法を取った方が性能が良いであろうと思われる。並列性も高く一石二鳥である。

今回の実装では Red-Black 法は numpy の離散畳み込み演算を利用することで高速に実装できたが、SOR 法は for 文を回す必要があり、低速な実装となってしまった。

様々な実験をしたため書いたコード量は多くなってしまったが、アルゴリズムのコア部分の実装を次ページから印刷する。

```
from math import sqrt,log,floor,cos,pi
import numpy as np
from numpy.random import rand
from matplotlib import pyplot
from numpy.linalg import norm

def residual(v,c,b):
    return log( norm(np.convolve(v,np.array([-1,c,-1]),"same")-
b,np.inf)/norm(b,np.inf),10)

def optw(N,c):
    return 2 / (1 + sqrt(1-((2/c)*cos(pi /(N+1)))**2))


"""CG method"""
def CG_method(N,c,b,loop):
    convv = np.array([-1,c,-1])
    result = []
    x = np.zeros(N)
    r = b - np.convolve(x,convv,"same")
    p = r
    for k in range(loop):
        Ap = np.convolve(p,convv,"same")
        a = np.dot(r,r)/np.dot(p,Ap)
        Nx = x + a*p
        Nr = r - a*np.convolve(p,convv,"same")
        beta = np.dot(Nr,Nr)/np.dot(r,r)
        Np = Nr + beta*p

        result.append(residual(Nx,c,b))

        x = Nx
        r = Nr
        p = Np
    return result


"""Jacobi method"""
def Jacobi(N,c,b,loop):
```

```
    result2 = []
    x = np.zeros(N)
    conv = np.array([-1,0,-1])
    for k in range(loop):
        x = (b-np.convolve(x,conv,"same"))/c
        result2.append(residual(x,c,b))
    return result2

"""Gauss-Seidel"""
def Gauss_Seidel(N,c,b,loop):
    result3 = []

    x = np.zeros(N)
    for k in range(loop):
        x[0] = (x[1]+b[0])/c
        for i in range(1,N-1):
            x[i] = (x[i-1]+x[i+1]+b[i])/c
        x[N-1] = (x[N-2]+b[N-1])/c

        result3.append(residual(x,c,b))
    return result3

"""SOR"""
def SOR(N,c,b,loop,w):
    result4 = []
    x = np.zeros(N)
    for k in range(loop):
        y = (x[1]+b[0])/c
        x[0] = x[0] + w*(y-x[0])
        for i in range(1,N-1):
            y = (x[i-1]+x[i+1]+b[i])/c
            x[i] = x[i] + w*(y-x[i])
        y = (x[N-2]+b[N-1])/c
        x[N-1] = x[N-1] + w*(y-x[N-1])

        result4.append(residual(x,c,b))
    return result4
```

```
"""RBSOR"""
convE = np.array([0,-1,-1])
convO = np.array([-1,-1,0])
def EOresidual(E,O,bE,bO,c,b):
    x =  norm(c*E + np.convolve(O,convE,"same") - bE,np.inf)
    y =  norm(c*O + np.convolve(E,convO,"same") - bO,np.inf)
    return log(max(x,y)/norm(b,np.inf),10)
def RBSOR(N,c,b,loop,w):
    result5 = []
    bO = np.array([b[i*2+1] for i in range(N//2)])
    bE = np.array([b[i*2] for i in range(N//2)])

    even = np.zeros(N//2)
    odd = np.zeros(N//2)

    for k in range(loop):
        odd  = odd  + w*((bO - np.convolve(even,convO,"same"))/c - odd )
        even = even + w*((bE - np.convolve(odd, convE,"same"))/c - even)

        vec = np.vstack((c*even+np.convolve( odd,convE,"same")-bE
                        ,c*odd +np.convolve(even,convO,"same")-bO))
        result5.append(EOresidual(even,odd,bE,bO,c,b))
    return result5
```