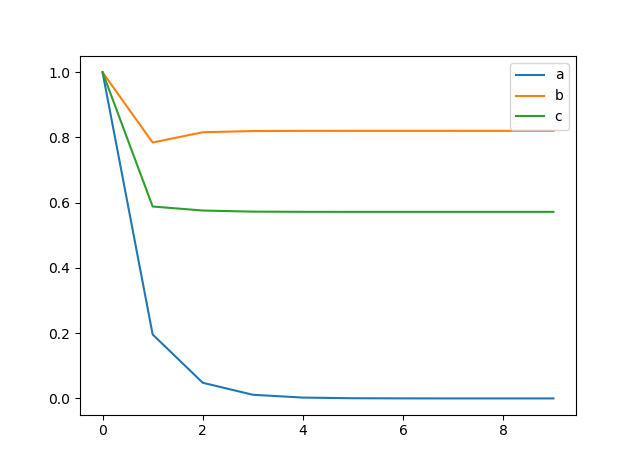
数値解析　レポート

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。

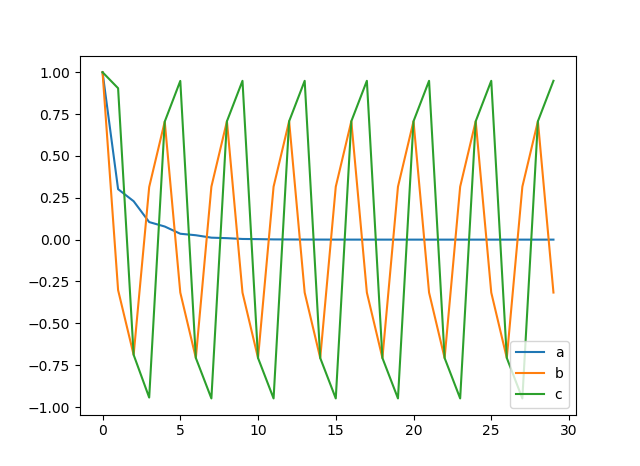
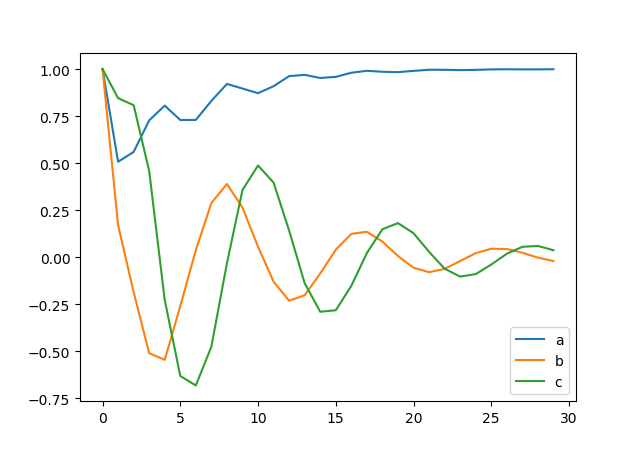
のときの各成分の推移を表したグラフが図1である。の固有多項式は、解はである。絶対値最大の固有値が一つであり固有ベクトルに収束する。

図２は

に対してべき乗法を適用した結果である。の固有値はであり、絶対値最大の固有値が二つ存在するので収束しない。

の固有値をする、つまりとすると、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、行列で対角成分対角成分からひとつずれた要素は-1である行列をと表記することにする。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

命題の固有値はである()

証明

第二種チェビシェフ多項式をで表すとする。

の固有多項式はであることを示す。帰納法を用いる. とする。

のとき、より良い。

のとき、より良い。

第二種チェビシェフ多項式は、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、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という関係を満たしており、固有多項式の根は2となる。

よって示された.

命題2. 次の式が成立する

証明

固有値はである。条件数は正規行列であればである。

のとき, , より良い.

のとき, , となり良い。

最適なSOR法のハイパーパラメータは、Jacobi法の反復行列のスペクトル半径が分かれば, 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. として、とする。

証明

であるので,固有値はであり, 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,

なおこのときのスペクトル半径はとなる

事実

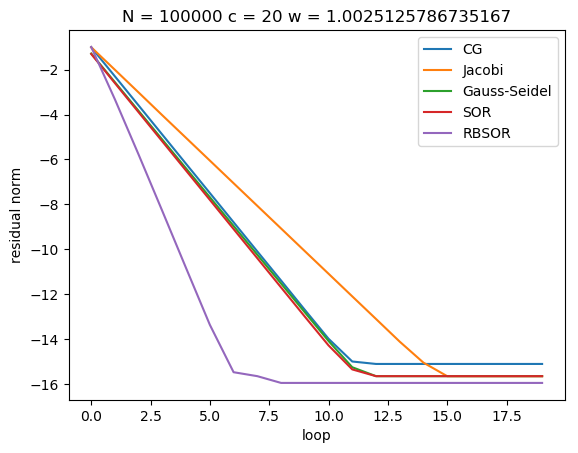
Gauss-Seidel法のスペクトル半径はJacobi法の半径の二乗である。

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。

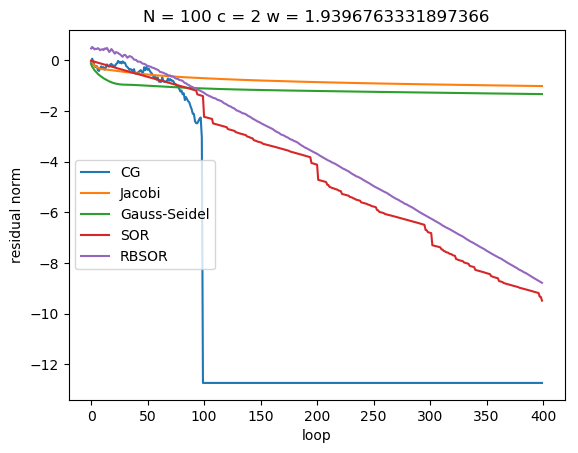
各手法について反復させたときの残差ベクトルのinfノルムの減少する様子をグラフにプロットした。縦軸は常用対数である。

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

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。

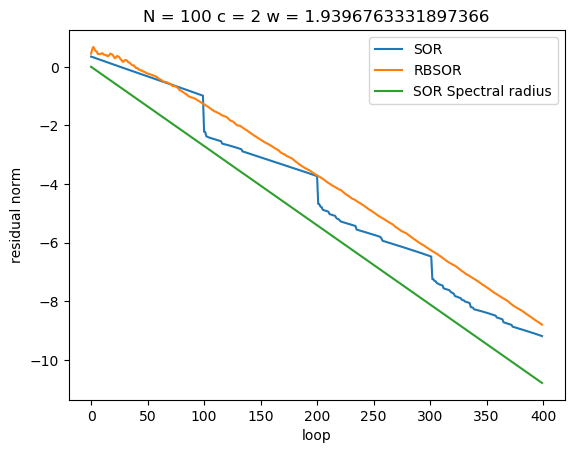
考察

CG法について。

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。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法はで入れたノルムが、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。

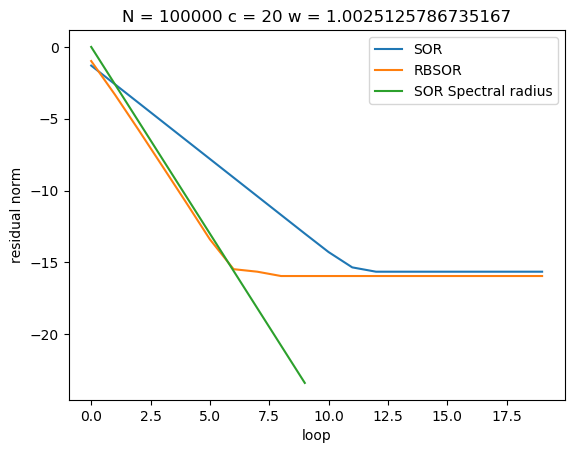
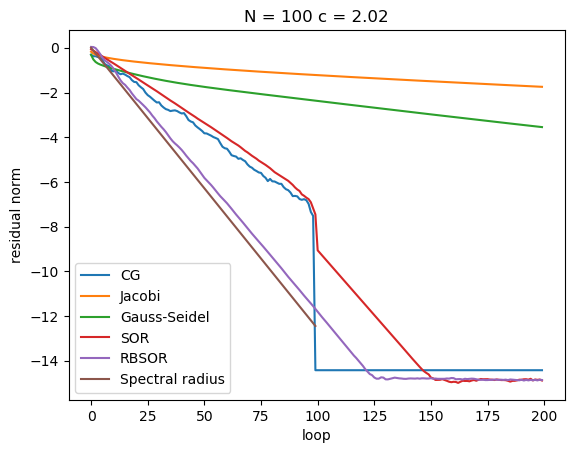
Jacobi法、Gauss-Seidel法について

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法はのとき100ループごとに急に残差が減少し、のときはRed-Black法に比較して収束が遅い。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。

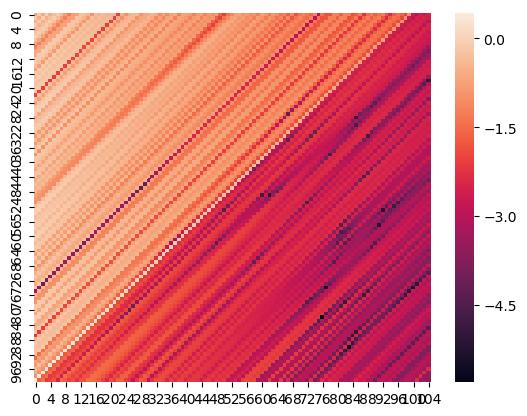
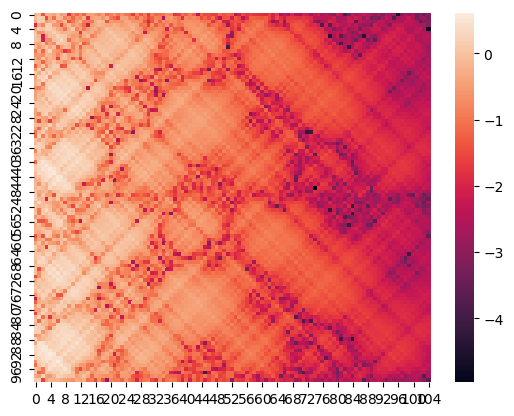
事実

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。

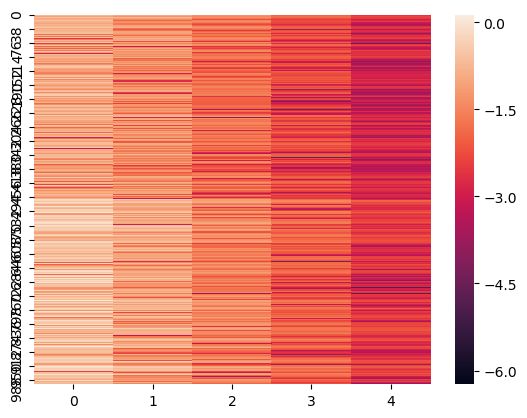
反復法は適切なノルムを入れるとスペクトル半径の速度で残差ノルムを減少させられることが知られており、反復行列とinfノルムの相性が問題である。ここで残差ベクトルの成分ごとの大きさをヒートマップで見てみる。

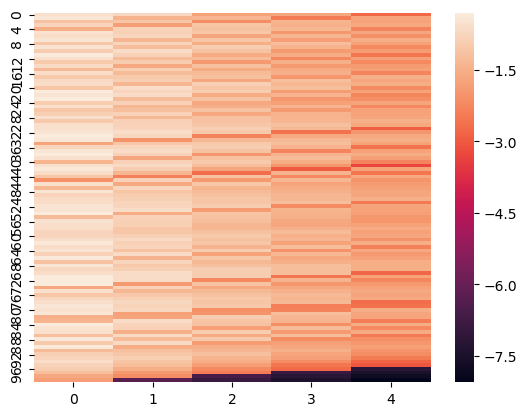


この図はのときの、残差ベクトルの減少の様子を成分ごとに見たものである。色で大きさを表す。一番左の列が初期残差であり、右に行くにつれて残差がだんだん小さくなる。

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。

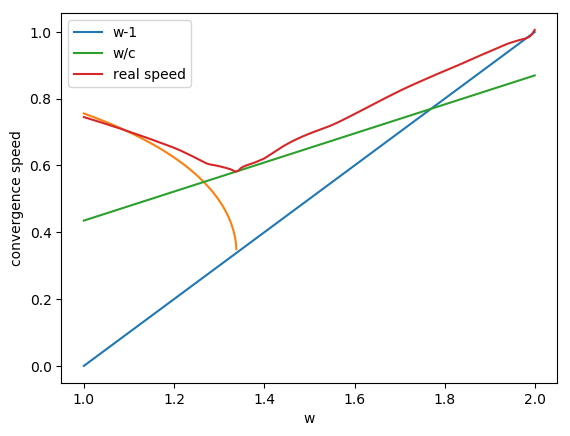
のときに５回ループを回した時のものも比較してみる。左がSOR法, 右がRed-Blach法である。



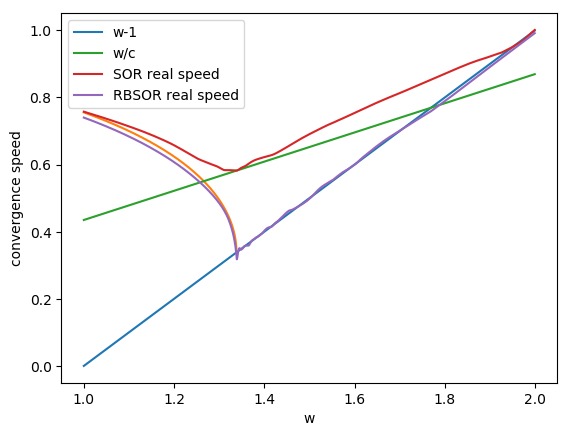
SOR法では一部の成分の残差が急激に減少しているが、Red-Black法では均一に誤差が減少している。適切なノルムを入れれば両者は同じ収束速度であっても、2ノルムやinfノルムを考えると、Red-Black法の方が収束性が良いといえるであろう。

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（添え字で書けばと書ける成分）を見ればよい。

SOR法の反復行列は

とかけることから、が対角成分がの行列であることに注意すれば、対角成分の隣にはが並んでいることが分かる。よってSOR法において残差はされて隣に移る。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 |