電気303/電情303 数値解析 (5)

連立一次方程式の解法 (2)

アルゴリズムと その特徴

連立一次方程式の解法

- 連立一次方程式の解法は，大別すると，直接法, 反復法, 共役勾配法の 3 種類に分類される.
- 直接法は数値計算の誤差がない場合に有限回の演算で解を与える手法.
- 反復法は繰り返しによって近似解の系列を生成する方法.

- 直接法の代表格は, これから述べる Gauss の消去法
- Gauss の消去法の準備のために, LU 分解という概念を説明する.
- 以下では, 行列 $\boldsymbol{A}$ は n 行 n 列の正則行列とする.

LU 分解

- もっとも素朴なガウスの消去法は, 行列 $\boldsymbol{A}$ が正方かつ正則で, 行列 $\boldsymbol{A}$ が行の入れ換えなしに階段行列に変形できる場合に相当.
- この場合に相当する, 行列の LU 分解と呼ばれるものを, これから導出する.

- $\boldsymbol{A}^{(1)} = \boldsymbol{A}$ とし, 以下, 繰り返し計算によって, これを $\boldsymbol{A}^{(2)}, \boldsymbol{A}^{(3)}, \ldots$ のように変形してゆく.
- $\boldsymbol{A} = \boldsymbol{A}^{(1)}$ の各成分を以下のように書く.

$$\boldsymbol{A}^{(1)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & & & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}$$

- $a_{11}^{(1)} \neq 0$ と仮定し,

  $\boldsymbol{l}_1 = \begin{pmatrix} 1 \\ \frac{a_{21}^{(1)}}{a_{11}^{(1)}} \\ \vdots \\ \frac{a_{n1}^{(1)}}{a_{11}^{(1)}} \end{pmatrix}$, $\boldsymbol{u}_1 = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \end{pmatrix}$ と

  すると $\cdots$

- $\boldsymbol{A}^{(1)} = \boldsymbol{l}_1 \boldsymbol{u}_1^T + \boldsymbol{A}^{(2)}$,

$$\boldsymbol{A}^{(2)} = \left(\begin{array}{c|ccc} 0 & 0 & \cdots & 0 \\ \hline 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & & & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{array}\right)$$

という形になる (各成分の式は略).

- 次に, $a_{22}^{(2)} \neq 0$ と仮定し, 以下のようにおくと:

$$\boldsymbol{l}_2 = \begin{pmatrix} 0 \\ 1 \\ \frac{a_{32}^{(1)}}{a_{22}^{(2)}} \\ \vdots \\ \frac{a_{n2}^{(1)}}{a_{22}^{(2)}} \end{pmatrix}, \boldsymbol{u}_2 = \begin{pmatrix} 0 & a_{22}^{(2)} & a_{23}^{(2)} \cdots & a_{2n}^{(2)} \end{pmatrix}$$

- $\boldsymbol{A}^{(2)} = \boldsymbol{l}_2 \boldsymbol{u}_2^T + \boldsymbol{A}^{(3)}$,

$$\boldsymbol{A}^{(3)} = \left(\begin{array}{cc|cc} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \hline 0 & 0 & a_{22}^{(3)} & \cdots \\ \vdots & \vdots & & \end{array}\right)$$

という形になる (各成分の式は略).

- 同様にして一回計算するごとに行列の左および上側の零列と零行が 1 ずつ増えるから …
- $\boldsymbol{A}^{(n)} = \boldsymbol{l}_n \boldsymbol{u}_n^T + \boldsymbol{A}^{(n+1)}$ とすると…
- $\boldsymbol{A}^{(n+1)} = \boldsymbol{0}$(零行列) である.

- 以上をまとめると

$$\boldsymbol{A} = \boldsymbol{l}_1 \boldsymbol{u}_1^T + \cdots + \boldsymbol{l}_n \boldsymbol{u}_n^T$$

- $\boldsymbol{L}$ と $\boldsymbol{U}$ を以下のように定義する.

$$\boldsymbol{L} = \begin{pmatrix} \boldsymbol{l}_1 & \cdots & \boldsymbol{l}_n \end{pmatrix}, \quad \boldsymbol{U} = \begin{pmatrix} \boldsymbol{u}_1^T \\ \vdots \\ \boldsymbol{u}_n^T \end{pmatrix}$$

- 以上の定義のもとで,

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$$

となる. これを行列 $\boldsymbol{A}$ の **LU 分解**という.

- 以上の構成法により, $\boldsymbol{L}$ は対角成分が 1 の下三角行列となることが保証されている.
- 同様に, $\boldsymbol{U}$ は上三角行列であることが保証されている.

- 上三角行列とは以下の形の行列

$$\begin{pmatrix} * & \cdots & \cdots & * \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * \end{pmatrix}$$

ただし $*$ は任意の数 (零でもよい).

- 下三角行列とは以下の形の行列

$$\begin{pmatrix} * & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ * & \cdots & \cdots & * \end{pmatrix}$$

ただし $*$ は任意の数 (零でもよい).

- LU 分解を用いて連立一次方程式

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{L}\boldsymbol{U}\boldsymbol{x} = \boldsymbol{b}$$

を解くには,

$$\boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$$

$$\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$$

という 2 個の連立一次方程式を順に解けばよい.

- $\boldsymbol{L}$ は三角行列だから, 連立一次方程式

$$\boldsymbol{Ly} = \boldsymbol{b}$$

の解は, 以下の手順で構成できる

  - ▷ 最上段のスカラーに関する方程式を解く
  - ▷ その結果を 1 行下の式に代入して, スカラーに関する方程式を解く
  - ▷ 同じことを最も下の行まで繰り返す

- $\boldsymbol{U}$ は三角行列だから, 連立一次方程式

$$\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$$

の解は, 以下の手順で構成できる

  - ▷ 最下段のスカラーに関する方程式を解く
  - ▷ その結果を 1 行上の式に代入して, スカラーに関する方程式を解く
  - ▷ 同じことを最も上の行まで繰り返す

- $\boldsymbol{A} = \boldsymbol{LU}$ という LU 分解が得られているとき, さらに行列 $\boldsymbol{U}$ を対角行列 $\boldsymbol{D}$ と対角要素が 1 の上三角行列 $\boldsymbol{U}'$ の積であらわし,

$$\boldsymbol{A} = \boldsymbol{LDU}'$$

と書き直すことがある. これを LDU 分解という.

- LDU 分解は存在すれば一意的である．これを示す．

- $\boldsymbol{A} = \boldsymbol{L}_1\boldsymbol{D}_1\boldsymbol{U}_1 = \boldsymbol{L}_2\boldsymbol{D}_2\boldsymbol{U}_2$ であったと仮定すると, $\boldsymbol{L}_2^{-1}\boldsymbol{L}_1\boldsymbol{D}_1 = \boldsymbol{D}_2\boldsymbol{U}_2\boldsymbol{U}_1^{-1}$ であるが, $\boldsymbol{L}_2^{-1}\boldsymbol{L}_1$ が対角要素が1の下三角行列, $\boldsymbol{U}_2\boldsymbol{U}_1^{-1}$ が対角要素が1の上三角行列であることに注意すると, $\boldsymbol{D}_1 = \boldsymbol{D}_2$ が得らる.

- $\boldsymbol{L}_2^{-1}\boldsymbol{L}_1\boldsymbol{D}_1 = \boldsymbol{D}_2\boldsymbol{U}_2\boldsymbol{U}_1^{-1}$ の左辺は対角成分が 1 の下三角行列, 左辺は対角成分が 1 が上三角行列で, $\boldsymbol{D}_1 = \boldsymbol{D}_2$ であることは既に示されていたから, $\boldsymbol{L}_2^{-1}\boldsymbol{L}_1 = \boldsymbol{I}_n$, $\boldsymbol{U}_2\boldsymbol{U}_1^{-1} = \boldsymbol{I}_n$ となり, よって $\boldsymbol{L}_2 = \boldsymbol{L}_1$, $\boldsymbol{U}_2 = \boldsymbol{U}_1$ となる. よって, LDU 分解は, 存在すれば, 一意的である.

- $\boldsymbol{A}$ が対称行列で LDU 分解できるとき, $\boldsymbol{A} = \boldsymbol{LDU}$ とすると, $\boldsymbol{A} = \boldsymbol{A}^T$ と LDU 分解の一意性から, $\boldsymbol{L} = \boldsymbol{U}^T$, $\boldsymbol{U} = \boldsymbol{L}^T$ が導かれる. したがって, $\boldsymbol{A} = \boldsymbol{LDL}^T$ と書ける. これを対称行列 A の LDL^T 分解と呼ぶ.

- $\boldsymbol{A}$ が LDU 分解できる正定対称行列である場合には, $\boldsymbol{D}$ の各要素は正だから, $\boldsymbol{D}$ の対角要素の正の平方根を対角要素とする行列を $\boldsymbol{G}$ とすると, $\boldsymbol{D} = \boldsymbol{G}\boldsymbol{G}^T$ であり, したがって $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{G}\boldsymbol{G}^T\boldsymbol{L}^T$ と書ける. $\boldsymbol{C} = \boldsymbol{L}\boldsymbol{G}$ とおくと,

$$\boldsymbol{A} = \boldsymbol{C}\boldsymbol{C}^T$$

である. この表現を, $\boldsymbol{A}$ の Cholesky 分解と呼ぶ.

Gauss の消去法

- Gauss の消去法とは, 連立一次方程式を, 行列の基本変形により, 階段行列に関する連立一次方程式に変形する手順である.

- 階段行列に関する連立一次方程式は, 以下の手順によって解くことができる.
  - ▷ 最下段に関する方程式を解く
  - ▷ その結果を 1 行上の式に代入して, その方程式を解く
  - ▷ 同じことを最も上の行まで繰り返す

- したがって, 連立方程式を階段行列に関する連立一次方程式に変形できれば, 連立一次方程式は解けたと言える.

- 以下では, LU 分解から Gauss の消去法を導出する.
- ベクトル $\boldsymbol{l}_k$ の第 k 成分が 1 であったことを思い出し, 第 $k+1$ 成分以降をまとめた $n-k-1$ 次のベクトルを $\bar{\boldsymbol{l}}_k$ と書くことにする.

- $\boldsymbol{L}_1 = \begin{pmatrix} 1 & \boldsymbol{0} \\ -\bar{\boldsymbol{l}}_1 & \boldsymbol{I}_{n-1} \end{pmatrix}$ とおく ($\boldsymbol{I}_{n-1}$ は $n-1$ 次の単位行列, $\boldsymbol{0}$ はその部分が零であることを示す)
- $\boldsymbol{A}^{(1)} = \boldsymbol{l}_1 \boldsymbol{u}_1^T + \boldsymbol{A}^{(2)}$ の両辺に $\boldsymbol{L}_1$ を左から乗じ, $\boldsymbol{L}_1$ の構造を利用して整理すると (詳細は略), $\boldsymbol{L}_1 \boldsymbol{A}^{(1)} = \boldsymbol{e}_1 \boldsymbol{u}_1^T + \boldsymbol{A}^{(2)}$ となる (ただし $\boldsymbol{e}_1$ は第 1 番目の単位ベクトル).

- 次に, $\boldsymbol{A}^{(2)} = \boldsymbol{l}_2\boldsymbol{u}_2^T + \boldsymbol{A}^{(2)}$ を考える.
- $\boldsymbol{L}_2 = \left(\begin{array}{c|cc} 1 & 0 & \boldsymbol{0} \\ \hline 0 & 1 & \boldsymbol{0} \\ \boldsymbol{0} & -\bar{\boldsymbol{l}}_2 & \boldsymbol{I}_{n-2} \end{array}\right)$ とおく.
- この行列の構造に注意して計算すると, $\boldsymbol{L}_2\boldsymbol{A}^{(2)} = \boldsymbol{e}_2\boldsymbol{u}_2^T + \boldsymbol{A}^{(3)}$ となる.
- $\boldsymbol{L}_2\boldsymbol{e}_1 = \boldsymbol{e}_1$ であることに注意すると, $\boldsymbol{L}_2\boldsymbol{L}_1\boldsymbol{A} = \boldsymbol{e}_1\boldsymbol{u}_1^T + \boldsymbol{e}_2\boldsymbol{u}_2^T + \boldsymbol{A}^{(3)}$ となる.

- 以下同様にして,
  $\boldsymbol{L}_n \cdots \boldsymbol{L}_1 \boldsymbol{A} = \boldsymbol{e_1} \boldsymbol{u}_1^T + \cdots \boldsymbol{e}_n \boldsymbol{u}_n^T = \boldsymbol{U}$
  となる ($\boldsymbol{A}^{(n+1)} = \boldsymbol{0}$ に注意).

- 以上によって得られた式の $\boldsymbol{L}_n \cdots \boldsymbol{L}_1$ は基本行列の積であり, 右辺は階段行列になっている. 階段行列を求める手順が Gauss の消去法だったから, LU 分解から Gauss の消去法が導かれたことになる.
- 上述のように階段行列を求める手順を前進消去という.

- 行交換が必要ない場合には, Gauss の消去法も, LU 分解も, $\boldsymbol{l}_1, \ldots, \boldsymbol{l}_n$, $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ を求めることに相当するので, Gauss の消去法と LU 分解は本質的に同じ.

- 解の構成法についてもう少し詳しく見てゆく.
- $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ は, $\bar{\boldsymbol{b}} = \boldsymbol{L}_n \cdots \boldsymbol{L}_1 \boldsymbol{b}$ とおくと, $\boldsymbol{L}_n \cdots \boldsymbol{L}_1 \boldsymbol{A}\boldsymbol{x} = \boldsymbol{U}\boldsymbol{x} = \bar{\boldsymbol{b}}$ と変形されるから, $\boldsymbol{U}\boldsymbol{x} = \bar{\boldsymbol{b}}$ という連立一次方程式を解くことにより, 解 $\boldsymbol{x}$ が求められる.

- 具体的には, $\bar{\boldsymbol{b}}$ の各成分を $\bar{b}_1, \ldots, \bar{b}_n$, $\boldsymbol{U}$ の第 (i, j) 成分を u_{ij} とし, $\boldsymbol{U}$ が正則な上三角行列であったことに注意すると,
  - ▷ まず $x_n = \bar{b}_n / u_{nn}$ が得らる;
  - ▷ 次に $u_{n-1,n-1} x_{n-1} + u_{n-1,n} x_n = \bar{b}_{n-1}$ にこれを代入して x_{n-1} が得られる;
  - ▷ 以下同様に, 逐次的に解 $\boldsymbol{x}$ の全成分が得られる.

- 上述の操作を後退代入という.

ピボット選択

- 行列 $\boldsymbol{A}$ が正則であっても, $a_{11} = 0$ であるということはあり得る.
- $a_{11} \neq 0$ であっても, 絶対値が零に近い場合には, a_{11} を使って LU 分解あるいは Gauss の消去法によって連立一次方程式の解を求めると, 数値計算の誤差が大きくなる可能性がある.

- 以下では, 仮に, 零あるいは零に近い要素を「条件が悪い」と呼び, そうでない要素を「条件が良い」と呼ぶ.
- 計算不能あるいは数値的な条件悪化を防ぐには, 行を入れ換えて, 条件がよい $a_{k_1,1}$ が行列の一番上に来るようにすればよい.

- これは, 見方を変えると, a_{11} のかわりに $a_{k_1,1}$ に着目して, LU 分解あるいは Gauss の消去法を遂行していることになる.

- 第 k ステップについても同様に, 数値的な条件が良い $a_{j_k,k}$ に着目して LU 分解あるいは Gauss の消去法を遂行してゆく.

- LU 分解あるいは Gauss の消去法の各ステップで着目している条件が良い要素の番号 (j_k, k) のことをピボットあるいは枢軸と呼ぶ.
- 対応する $a^{(k)}_{j_k,k}$ のことをピボット要素あるいは枢軸要素と呼ぶ.

- 数値的な条件が良いように適切にピボットを選ぶ操作のことを, ピボット選択あるいは枢軸選択と呼ぶ.

- ピボット選択の選択法のひとつは, $\{a_{j,k}^{(k)} : j = k, k+1, \ldots, n\}$ の中で絶対値が最大の要素の添字を選ぶ方法 (行交換によるピボット操作).
- 列交換によるピボット操作と呼ばれる第 k 列以降の列について同様の操作をする手法や, 完全ピボット操作と呼ばれる第 (p, q) 要素 (ただし $p, q \geq k$) すべての中から絶対値最大のものを選ぶ手法もある.

- 列交換によるピボット操作や完全ピボット操作では, 変数の順番もこれに対応して入れ換える必要がある.
- 数値計算の誤差はベクトル $\boldsymbol{b}$ の成分の大きさにも依存するので, ピボット操作だけで数値計算の誤差の低減が保証されるとは限らない.

- 方程式 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ を解くためにピボット選択付きの Gauss の消去法を使うことは, $\boldsymbol{P}$ をそれに対応する置換行列としたとき, $\boldsymbol{P}\boldsymbol{A}$ を LU 分解することに相当する.

Scilab における LU 分解

- Scilab において LU 分解を求める関数は `lu` である.
- `[L,U,P]=lu(A)` とすることで, $\boldsymbol{PA} = \boldsymbol{LU}$ となる行列 $\boldsymbol{L}$, $\boldsymbol{U}$, $\boldsymbol{P}$ を求めることができる.
- 実行例は次ページの通り.

```
-->A=[1 2 3;4 5 6;7 8 9];
-->[L,U,P]=lu(A);
-->L
 L  =
    1.          0.    0.
    0.1428571   1.    0.
    0.5714286   0.5   1.
```

```
-->U
 U  =
    7.    8.            9.
    0.    0.8571429     1.7142857
    0.    0.            1.110D-16
```

```
-->P
 P  =
    0.    0.    1.
    1.    0.    0.
    0.    1.    0.
```

```
-->P*A
 ans  =
    7.    8.    9.
    1.    2.    3.
    4.    5.    6.
```

```
-->L*U
 ans  =
    7.    8.    9.
    1.    2.    3.
    4.    5.    6.
```

Gauss-Jordan 法

- Gauss の消去法の終了後に, さらに列基本変形を施して, 行列 $\boldsymbol{U}$ を単位行列に変換する方法もある. これを **Gauss-Jordan 法**という.
- 列基本変形による単位行列への変換の手法は, 線形代数で用いられる手順と同一なので, この講義では既知と見做して立ち入らない.

- 計算量という観点から言うと Gauss-Jordan 法にはあまりメリットはないが, 並列計算機には適しているという指摘もある.

疎行列

- 要素の大部分が零の行列を疎行列という.
- 応用であらわれる大規模行列は多くの場合疎行列である.
- 疎行列で零要素をメモリに格納することは無駄であるので, 必要な要素だけをメモリに記憶する方法が工夫されている.

- 疎行列に対する演算は, 行列の疎性を破壊しないことが望ましい.
- Scilab で疎行列を扱うための組み込み関数は `sparse`.
- 疎行列は大規模演算を行う場合に用いられるもので, 学部生向けの講義としてはマニアックすぎる話題なので, この講義ではこれ以上立ち入らない.

反復法

- 直接法は有限回の演算で連立一次方程式の解を求める手法で, その代表格がすでに述べたGaussの消去法であった.
- 直接法は一般に高速で, かつ有理数演算をサポートしている処理系では, 直接法は有理数に限定すれば, 数値計算の誤差なく連立一次方程式を解くことができるという利点がある.

- 一方で, 一般に直接法は大きな記憶域を必要とするという欠点もある.
- 大規模な問題では, 問題を解くための記憶域を確保でないことがある. このような場合には, 相対的に必要とされる記憶域が小さい反復法と呼ばれる解法が用いられる.

- 反復法は繰り返しによって解を連立一次方程式の真の解に漸近させる手法であり, 係数行列のうち零でない要素だけを記憶しておけばよいという利点を持つため, 大規模疎行列に関する連立一次方程式を解くときに威力を発揮することがある.

- 「大規模」「疎行列」といった概念は, 実用的な概念であって, 数学的な概念ではない.
- 大規模疎行列が応用上重要なのは, たとえば偏微分方程式を差分近似する際に疎行列があらわれるから. このため, 物理現象のシミュレ– ションなどで, 反復法が必要となることがある.

- 反復法は必要とされる記憶域が相対的に少ないという利点を持つ一方で, 特殊な条件を満たす連立一次方程式にしか適用できないという欠点を持つ. この点は, どんな問題でも解けた直接法と対照的.

- 反復法の基本はJacobi法とGauss-Seidel法であるが (後述), これら以外にも, 偏微分方程式への応用をベースにした種々の解法がある.
- いくつか名前を挙げると, SOR 法, Chebyshev 加速法, ADI 法, マルチグリッド法など.

Jacobi 法

- $\boldsymbol{A}$ を n 行 n 列の行列, $\boldsymbol{x}$ および $\boldsymbol{b}$ を n 次のベクトルとする.
- $\boldsymbol{Ax} = \boldsymbol{b}$ を解きたい. ただし, 行列 $\boldsymbol{A}$ の対角要素はすべて零でないと仮定する (この条件が満たされない場合には Jacobi 法は適用できない).

- $\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots \\ a_{21} & a_{22} & a_{23} & \cdots \\ a_{31} & a_{32} & a_{33} & \cdots \\ \vdots & \vdots & \vdots & \end{pmatrix}$ とする.
- $\boldsymbol{A}$ の対角要素のみを抽出して $\boldsymbol{D}$ を作る.

$$\boldsymbol{D} = \begin{pmatrix} a_{11} & 0 & & \\ 0 & a_{22} & 0 & \\ & 0 & a_{33} & \ddots \\ & & \ddots & \ddots \end{pmatrix}$$

- $\boldsymbol{A} - \boldsymbol{D}$ の左下の部分を $\boldsymbol{E}$ とすると:

$$\boldsymbol{E} = \begin{pmatrix} 0 & & & \\ a_{21} & 0 & & \\ a_{31} & a_{32} & 0 & \\ \vdots & & \ddots & \ddots \end{pmatrix}$$

- $\boldsymbol{A} - \boldsymbol{D}$ の右上の部分を $\boldsymbol{F}$ とすると:

$$\boldsymbol{F} = \begin{pmatrix} 0 & a_{12} & a_{13} & \cdots \\ & 0 & a_{23} & \\ & & 0 & \ddots \\ & & & \ddots \end{pmatrix}$$

- $\boldsymbol{Ax} = \boldsymbol{b}$ は,

$$(\boldsymbol{D} + \boldsymbol{E} + \boldsymbol{F})\,\boldsymbol{x} = \boldsymbol{b}$$

と書ける.

- 上記の左辺第 2・3 項を右辺に移項し, 両辺に $\boldsymbol{D}^{-1}$ を掛けると,

$$\boldsymbol{x} = -\boldsymbol{D}^{-1}\left(\boldsymbol{E} + \boldsymbol{F}\right)\boldsymbol{x} + \boldsymbol{D}^{-1}\boldsymbol{b}.$$

- これに基づき, 次の漸化式を考える.

$$\boldsymbol{x}(k+1) = -\boldsymbol{D}^{-1}\left(\boldsymbol{E}+\boldsymbol{F}\right)\boldsymbol{x}(k) + \boldsymbol{D}^{-1}\boldsymbol{b}$$

- 漸化式 $\boldsymbol{x}(k+1) = -\boldsymbol{D}^{-1}(\boldsymbol{E}+\boldsymbol{F})\boldsymbol{x}(k) + \boldsymbol{D}^{-1}\boldsymbol{b}$ の解が一定値 $\bar{\boldsymbol{x}}$ に収束するならば…
- この漸化式の両辺で $k \to \infty$ とすると…
- $\bar{\boldsymbol{x}} = -\boldsymbol{D}^{-1}(\boldsymbol{E}+\boldsymbol{F})\bar{\boldsymbol{x}} + \boldsymbol{D}^{-1}\boldsymbol{b}$ なので, $\bar{\boldsymbol{x}}$ は $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ の解である.

- 初期値 $\boldsymbol{x}(0)$ を定め (何でもよい), 漸化式 $\boldsymbol{x}(k+1) = -\boldsymbol{D}^{-1}(\boldsymbol{E}+\boldsymbol{F})\boldsymbol{x}(k)+\boldsymbol{D}^{-1}\boldsymbol{b}$ の解を $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ の近似解とする方法が Jacobi 法.

- Jacobi 法は行列 $\boldsymbol{A}$ の対角要素がすべて非零であればプログラムとしては動作させることができるが, $\boldsymbol{A}$ の要素の値次第で, 列 $(\boldsymbol{x}(k))$ が発散することもあり得る.

- 列 $(\boldsymbol{x}(k))_{k\in\mathbb{N}}$ が $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ の解に収束するための必要十分条件は, 差分方程式

$$\boldsymbol{x}(k+1)=-\boldsymbol{D}^{-1}\left(\boldsymbol{E}+\boldsymbol{F}\right)\boldsymbol{x}(k)+\boldsymbol{D}^{-1}\boldsymbol{b}$$

が漸近安定, すなわち行列 $\boldsymbol{D}^{-1}(\boldsymbol{E}+\boldsymbol{F})$ のすべての固有値の絶対値が 1 未満となること.

- 以上の説明では便宜上 $\boldsymbol{D}$ の逆行列を明示的に書き表したが, 実際には $\boldsymbol{D}$ の逆行列を使うわけではない.

- $\boldsymbol{D}^{-1} = \begin{pmatrix} \frac{1}{a_{11}} & 0 & \\ 0 & \frac{1}{a_{12}} & \ddots \\ & \ddots & \ddots \end{pmatrix}$ だから…

- $\boldsymbol{D}^{-1}(\boldsymbol{E}+\boldsymbol{F})$ は, 行列 $\boldsymbol{E}+\boldsymbol{F}$ の第 1 行から第 n 行までにそれぞれ $1/a_{11}, \ldots, 1/a_{nn}$ を掛けたものになる. なお零要素については計算不要.

- $\boldsymbol{D}^{-1}\boldsymbol{b}$は, ベクトル$\boldsymbol{b}$の第1行成分から第$n$成分までにそれぞれ$1/a_{11}, \ldots, 1/a_{nn}$を掛けたものになる. なお零要素については計算不要.

Gauss-Seidel 法

- Gauss-Seidel 法は, 先に得られた

$$(\boldsymbol{D} + \boldsymbol{E} + \boldsymbol{F})\,\boldsymbol{x} = \boldsymbol{b}$$

という式を, Jacobi 法とは違った形で整理し, そこから漸化式を導いて解く方法である.

- 具体的には, 初期値 $\boldsymbol{x}(0)$ を定め (何でもよい),

$$(\boldsymbol{D}+\boldsymbol{E})\,\boldsymbol{x}(k+1)=\boldsymbol{b}-\boldsymbol{F}\boldsymbol{x}(k)$$

という漸化式を解く.

- Gauss-Seidel 法を成分ごとに書くと

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(- \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} + b_i \right)$$

となる．ただし $x_i^{(k)}$ は第 k 回目の繰り返しにおけるベクトル $\boldsymbol{x}$ の第 i 成分．成分ごとの差分方程式を使った方がメモリ消費を減らせるが, 行列を使った場合と比べてどちらが速いかは処理系によって変わる．

- Gauss-Seidel 法によって得られる列 $(\boldsymbol{x}(k))_{k\in\mathbb{N}}$ が $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ の解に収束するための必要十分条件は, $(\boldsymbol{D} + \boldsymbol{E})^{-1}\boldsymbol{F}$ のすべての固有値の絶対値が 1 未満となること.

SOR 法

- SOR 法とは, Successive Over-Relaxation 法の略であり, 逐次過大緩和法と訳される.
- SOR 法は設計パラメータ w を含む (ただし $0 < w < 2$).
- SOR 法とは, 初期値 $\boldsymbol{x}(0)$ を定め (何でもよい), 次ページで与える漸化式を解く方法.

$$\frac{1}{w}\left(\boldsymbol{D}+w\boldsymbol{E}\right)\boldsymbol{x}(k+1)$$
$$=\frac{1}{w}\left(\left(1-w\right)\boldsymbol{D}-w\boldsymbol{F}\right)\boldsymbol{x}(k)+\boldsymbol{b}$$

上記を成分ごとに書くと:

$$y_i^{(k+1)} = \frac{1}{a_{ii}} \left(- \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} + b_i \right)$$

$$x_i^{(k+1)} = x_i^{(k)} + w \left(y_i^{(k+1)} - x_i^{(k)} \right)$$

- SOR 法で得られた列 $(\boldsymbol{x}(k))_{k\in\mathbb{N}}$ が $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ の解に収束するための必要十分条件は, 行列

$$(\boldsymbol{D}+w\boldsymbol{E})^{-1}\left((1-w)\boldsymbol{D}-w\boldsymbol{F}\right)$$

のすべての固有値の絶対値が 1 未満となること.

- SOR 法の収束性はパラメータ w に依存する.
- パラメータ w の値によって収束の速さが変わるが, 大きい方がよいとも小さい方がよいともいえない.
- 実用上は, w を試行錯誤によって定めるが, w を解析的に求められる問題もある.

共役勾配法

- 連立一次方程式の代表的な解法は大別すると直接法と反復法であるが…
- 直接法と反復法を組み合わせた解法があり，共役勾配法と呼ばれる．

- 共役勾配法は非線形最小化問題に適用される最急降下法という手法から派生した手法.
- この手法は 1952 年に提案されたが, 数値計算の誤差に弱いため不遇の時代が続いた. しかし, 前処理によって特性が改善されることが判明し, 見直されている.

- これ以上研究の余地がないと思われる直接法や間接法と異なり, 共役勾配法はいまだに研究が続いている方法で, 非線形最適化問題の解法としての拡張が可能.

- 線形計算の研究を志すのであれば共役勾配法は必須であるが, 一般的な工学系の選択科目としては専門的すぎる内容と思われるので, この講義では概要のみ紹介する.

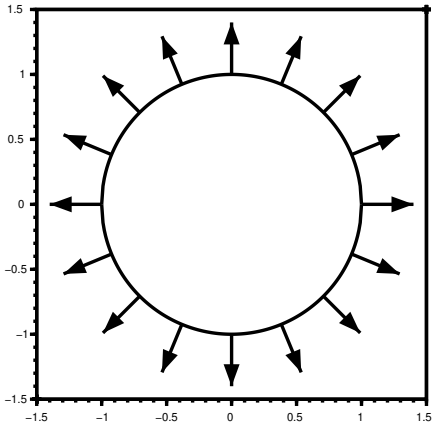
最急降下法

- 共役勾配法について述べるための準備として, まず最急降下法について述べる.
- 最急降下法は, 変数ベクトル $\boldsymbol{x}$ に関する実数値関数 $f(\boldsymbol{x})$ を最小化 (あるいは最大化) する手法のひとつ.

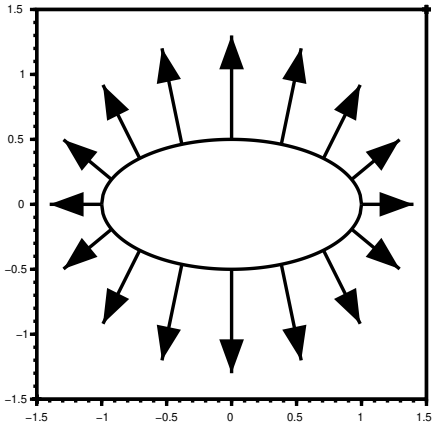
- 関数 f の勾配ベクトルを $\nabla f = (\frac{\partial f}{\partial \boldsymbol{x}})^T$ とする.
- ∇f は関数 f の等高線の外向き法線ベクトルを与える.

- ∇f は関数 f の等高線の外向き法線ベクトルだから, 関数 f が一定の条件を満たすときには, 解を $-\nabla f$ の方向に少しずつ動かせば, 解は f の最小値を与える点 $\boldsymbol{x}_*$ に収束する. この方法を最急降下法という.

- 最急降下法は, $f(x, y) = x^2 + y^2$ のように, 内向き法線ベクトルと関数が最小となる点の方向が近い場合には最急降下法はそれなりに高効率だが…



- 関数の等高線が細長い楕円になっているような場合には, 等高線の外向き法線ベクトルと関数が最小値を取る点への方向とのずれが大きく, 効率的でない.

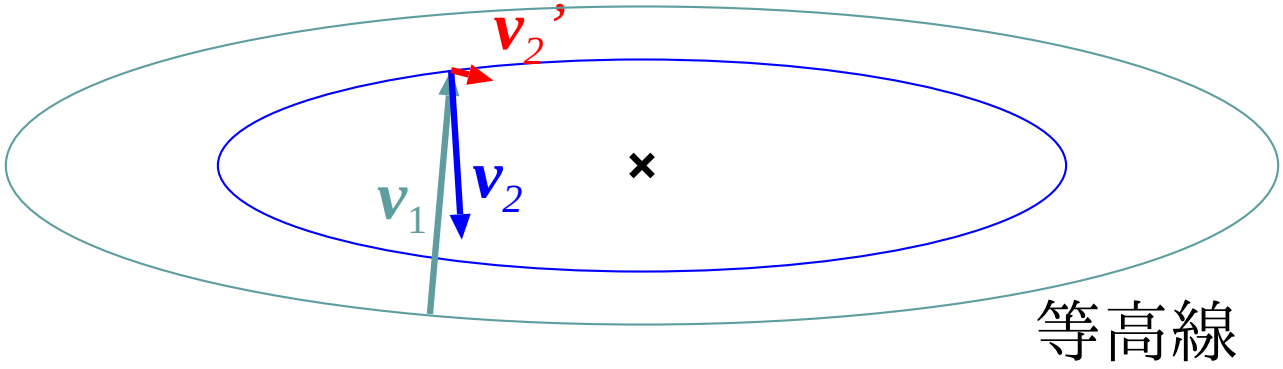


共役方向法と共役勾配法

- 上記の最急降下法の欠点を解消した1手法に共役方向法と呼ばれるものがあり, 共役勾配法はその一種.
- 以下, 解を動かす方向を探索ベクトルと呼ぶ.

- 共役方向法は最急降下法の改良版で, 過去の勾配ベクトルの系列を直交化して探索ベクトルを作ることが特徴.
- 探索ベクトルを作るには, 勾配ベクトルから過去の探索ベクトルと線形独立な成分を抽出する (射影を用いる).
- これがなぜ効率的かは, 関数 $f(\boldsymbol{x})$ の等高線が楕円の場合を考えればわかる.

- 等高線が楕円の場合に, 内向き法線ベクトル ($(-1)\times$ 勾配ベクトル) を直交化すると, 次のようになる.



- 共役方向法は「探索ベクトルの直交化」ということしか主張していない.
- 共役勾配法は, 共役方向法の枠内で, より具体的に探索ベクトルの構成法を与える.
- 以下では $(\boldsymbol{x}, \boldsymbol{y})$ により $\boldsymbol{x}$ と $\boldsymbol{y}$ の内積を表す.
- もっとも単純な共役勾配法は, 行列 $\boldsymbol{A}$ が正定対称行列である場合を対象とする.

- 解くべき問題は, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ の解 $\boldsymbol{x}$ を求めることである.
- この場合の共役勾配法のアルゴリズムは次ページに示す通り (典拠は杉原・室田, p. 150).

(初期化) $k = 0$ とし, 初期値 $\boldsymbol{x}_0$ を定め, $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$ とする. 終了条件に相当するパラメータ $\varepsilon > 0$ を定める.

(ループ) $\|\boldsymbol{r}_k\| < \varepsilon\|\boldsymbol{b}\|$ であれば終了. そうでなければ, 以下のように設定し, $k = k + 1$ としてループ冒頭に戻る.

$$\alpha_k = (\boldsymbol{r}_k, \boldsymbol{p}_k)/(\boldsymbol{p}_k, \boldsymbol{A}\boldsymbol{p}_k),$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k,$$

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{p}_k,$$

$$\beta_k = -(\boldsymbol{r}_{k+1}, \boldsymbol{A}\boldsymbol{p}_k)/(\boldsymbol{p}_k, \boldsymbol{A}\boldsymbol{p}_k),$$

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$$

- 共役勾配法は数値計算の誤差の影響を受けやすいので, 実用上は, $\boldsymbol{C}$ をある正則行列とし, 連立一次方程式 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ を,

$$(\boldsymbol{C}^{-1}\boldsymbol{A}\boldsymbol{C}^{-T})(\boldsymbol{C}^{T}\boldsymbol{x}) = \boldsymbol{C}^{-1}\boldsymbol{b}$$

というふうに変形してから共役勾配法を適用する. 行列 $\boldsymbol{C}$ を使って問題を変形する操作を前処理という.

- 前処理のしかたは色々あるが, 決定版と言うべき方法はない.

- 行列 $\boldsymbol{A}$ が正定対称行列でない場合の共役勾配法は, たとえば目的関数 $(\boldsymbol{Ax}-\boldsymbol{b}, \boldsymbol{Ax}-\boldsymbol{b})$ に関する最小化問題を解く, といったような形で定式化される.
- 上記の方法を一般化共役残差法 (Generalized Conjugate Residual 法; GCR 法) とよぶ.
- これ以外にも色々な方法がある. 良く知られたものを列挙すると…

GCR(m) 法, Orthomin(m) 法, 一般化最小残差法 (Generalized Minimal RESidual 法; GMRES 法), 双共役勾配法 (BiConjugate-Gradient 法; BCG 法), 擬似最小残差法 (Quasi-Minimal Residual 法; QMR 法), 安定化双共役勾配法 (BiConjugate Gradient STABilized 法; BiCGSTAB 法), Conjugate Gradient Squared 法 (CGS 法)

実際に使ってみると…

- 数値解析の分野では, コンピュータを使って実際に問題を問いてみると, 文献に書いてある通りにならないということが, しばしばある.

- 今回の講義で取り扱った連立一次方程式の解法ではどうかというと…

- テストに用いたパソコン
  - ▷ 11th Gen Intel(R) Core(TM) i5-11400 2.60GHz
  - ▷ 32.0GB Memory
  - ▷ Windows 11 22H2 22621.755

- ソフトウェア
  - MATLAB R2022a
  - Scilab-6.1.1
  - GNU Octave 7.1.0

- テスト用の連立 1 次方程式

$$\mathbf{A} = \begin{pmatrix} 100 & 1 & & & \\ 1 & 100 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 100 \end{pmatrix}$$

とする.

- 前ページの $\boldsymbol{A}$ と全要素が 1 のベクトル $\mathbf{b}$ に対し,

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

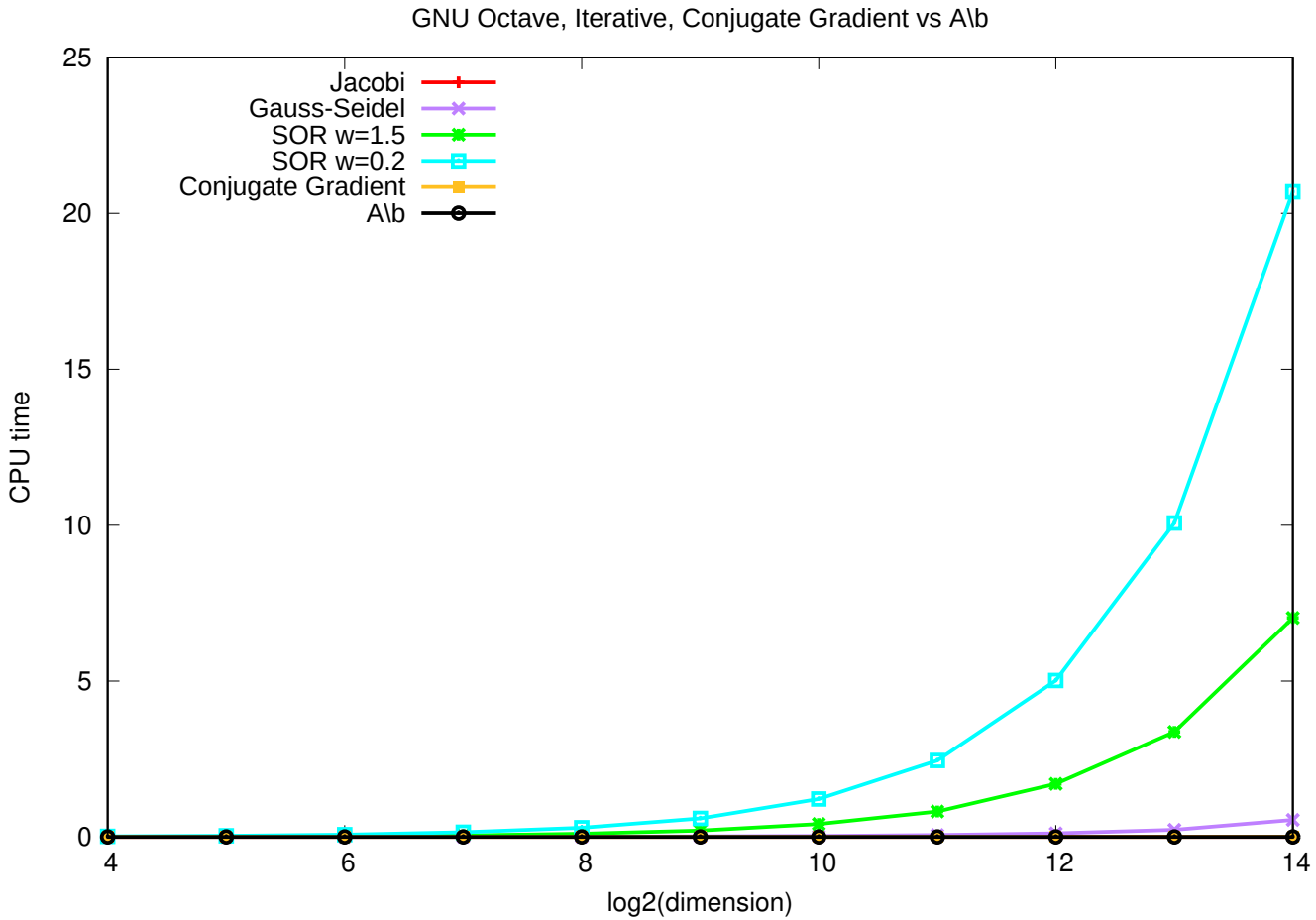
を解く.

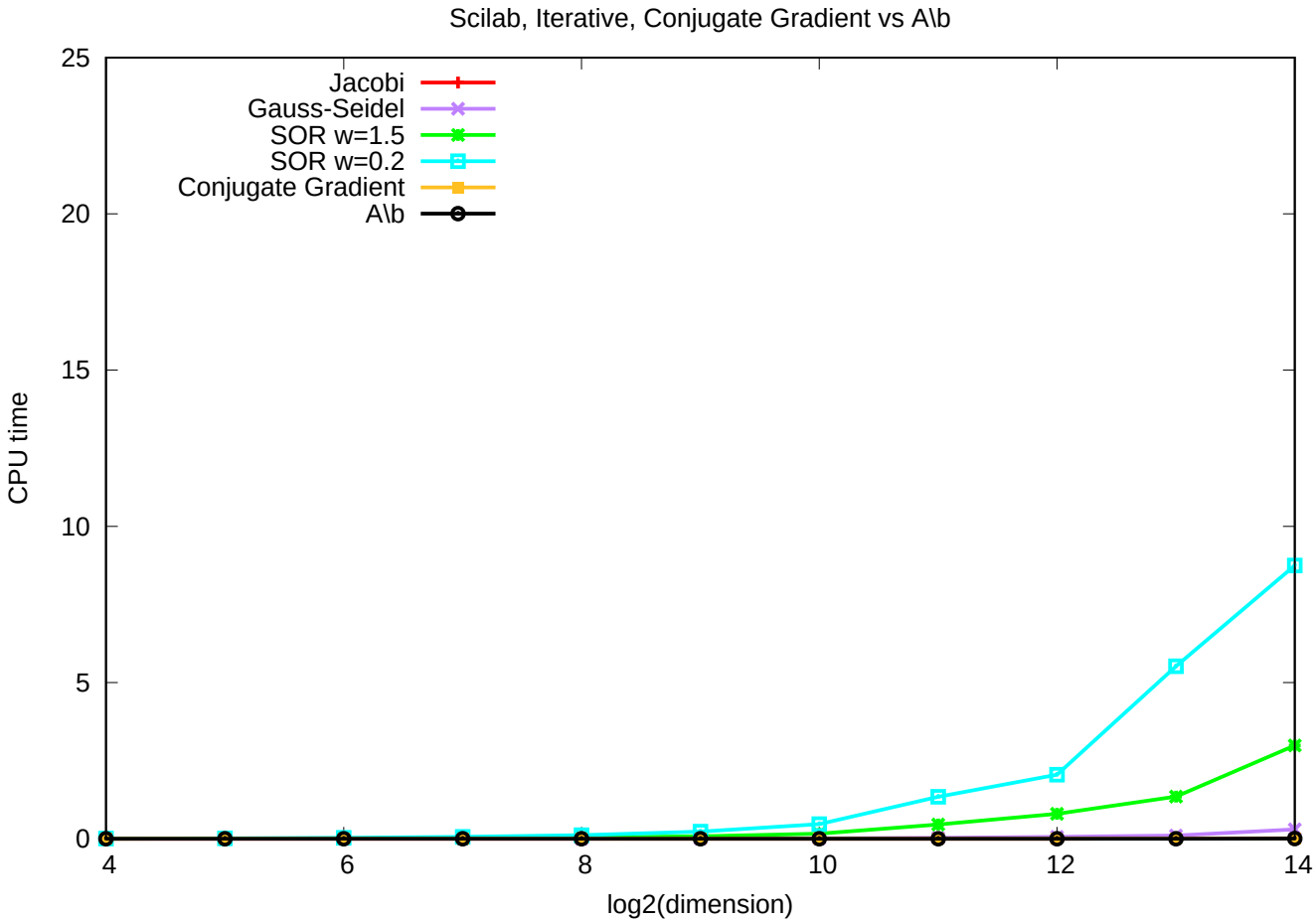
- この問題は, 反復法が得意とするタイプの問題で, $\boldsymbol{A} \simeq 100\boldsymbol{I}$ なので, $\boldsymbol{x} \simeq \dfrac{1}{100}\mathbf{1}$ ($\mathbf{1}$ は全成分が 1 のベクトル) であることがわかる ($\simeq$ は「ほぼ等しい」の意味).

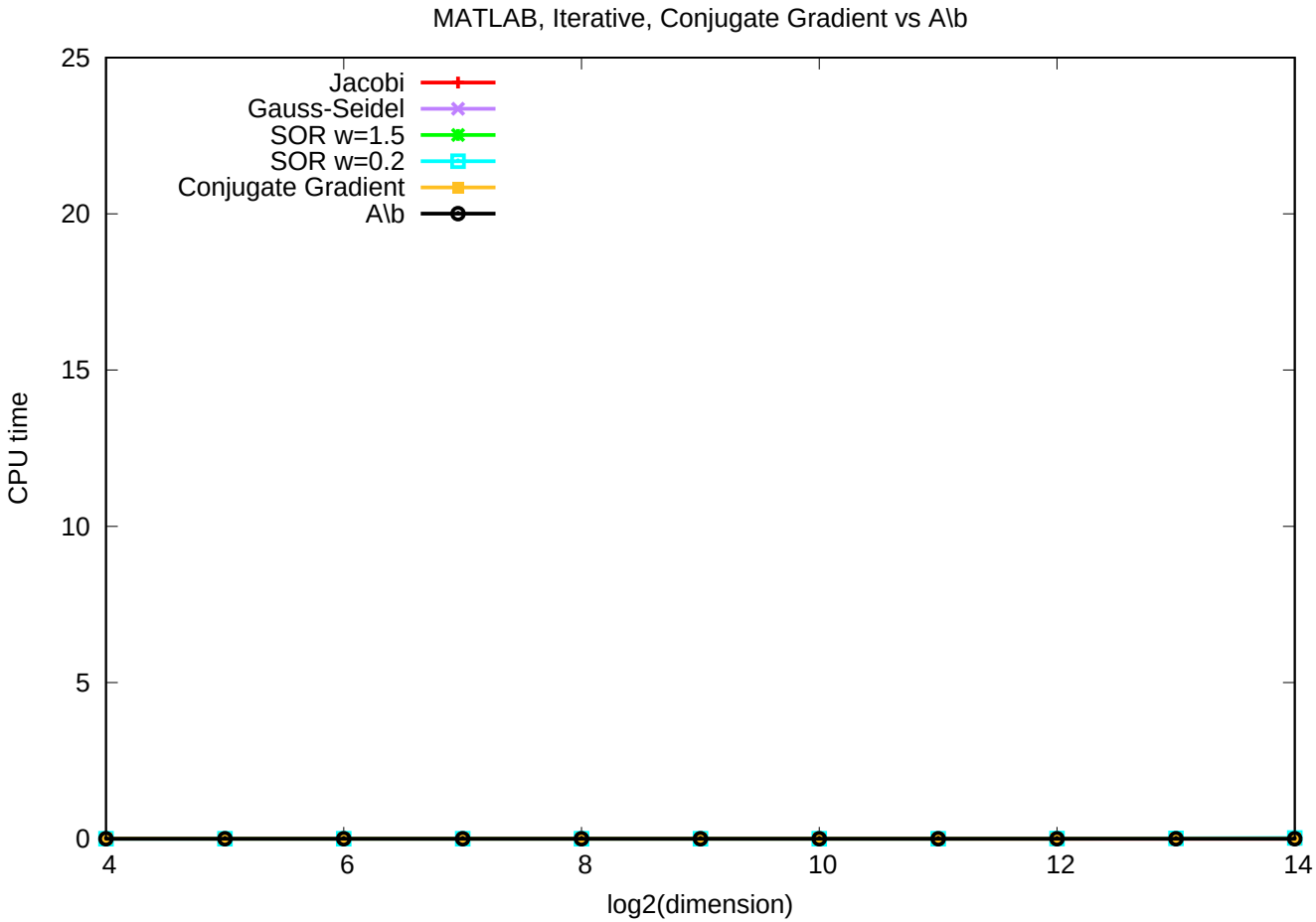
- GNU Octave, Scilab および MATLAB で, この連立方程式を Jacobi 法, Gauss-Seidel 法, SOR (1.5) 法, SOR (0.2) 法, 前処理付き共役勾配法 (PCG 法), $\boldsymbol{A}\backslash\boldsymbol{b}$ (標準ソルバ) で 10 回解き, 平均計算時間を比較する.
- $n = 2^i$ とし, i を 4 から 14 まで変化させる ($\boldsymbol{x}$ の次元に直すと 16 から 16384 まで).

- これから, GNU Octave, Scilab, MATLAB の順で, 各アルゴリズムの平均実行時間のグラフを示す.

- 縦軸を最も遅かった GNU Octave に合わせているため, グラフが非常に見にくいことに注意. 通常はこのようなグラフを作成すべきではないが, MATLAB と比較して, GNU Octave と Scilab の実行が極めて遅いことを強調するため, 敢えてこのようにしている.
- 縦軸に実行時間の対数を取った, より見やすいグラフも, 後で示す.


GNU Octave, Iterative, Conjugate Gradient vs A\b
Jacobi
Gauss-Seidel
SOR w=1.5
SOR w=0.2
Conjugate Gradient
A\b
CPU time
log2(dimension)
0
5
10
15
20
25
4
6
8
10
12
14

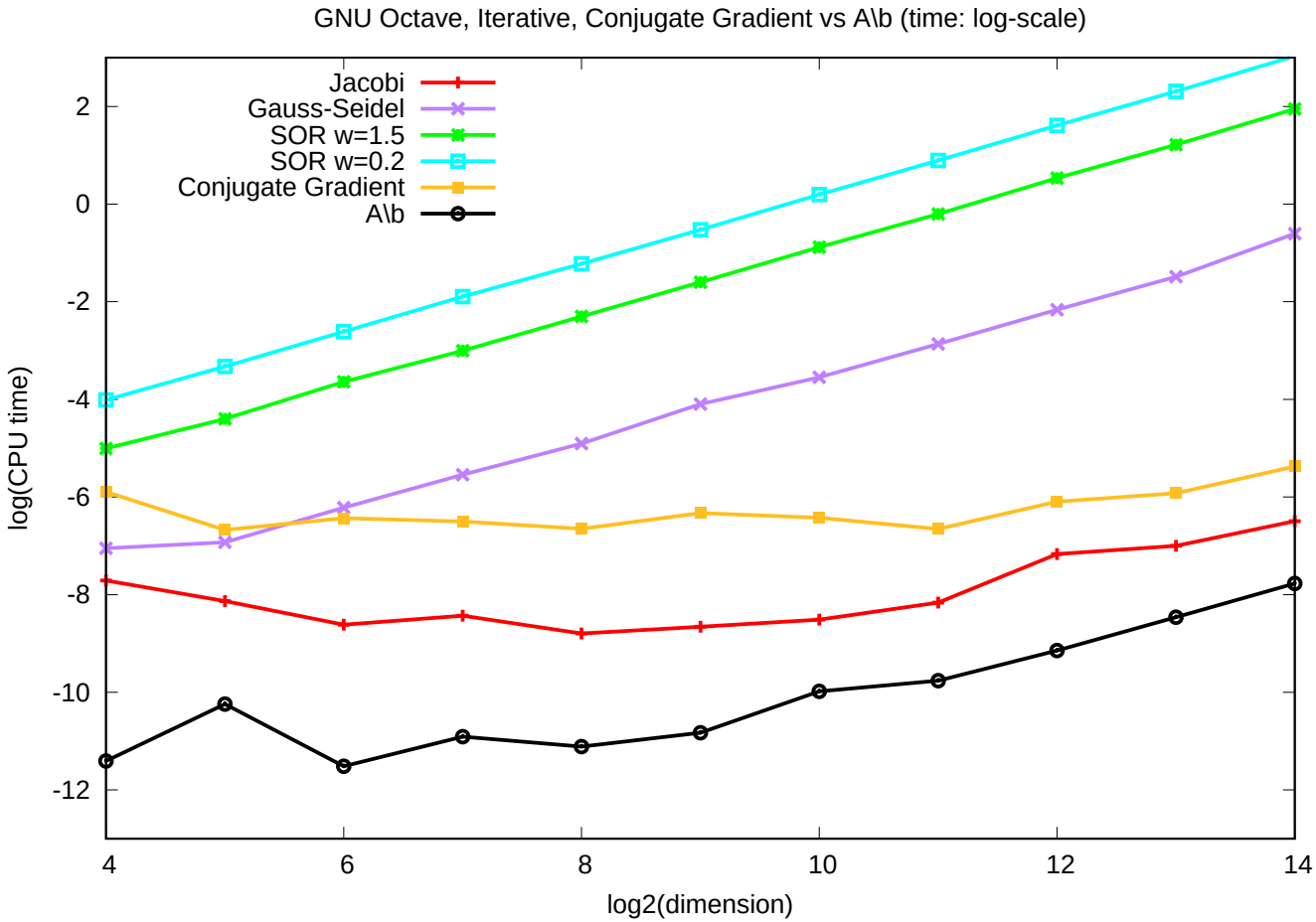

Scilab, Iterative, Conjugate Gradient vs A\b
Jacobi
Gauss-Seidel
SOR w=1.5
SOR w=0.2
Conjugate Gradient
A\b
CPU time
log2(dimension)
0
5
10
15
20
25
4
6
8
10
12
14

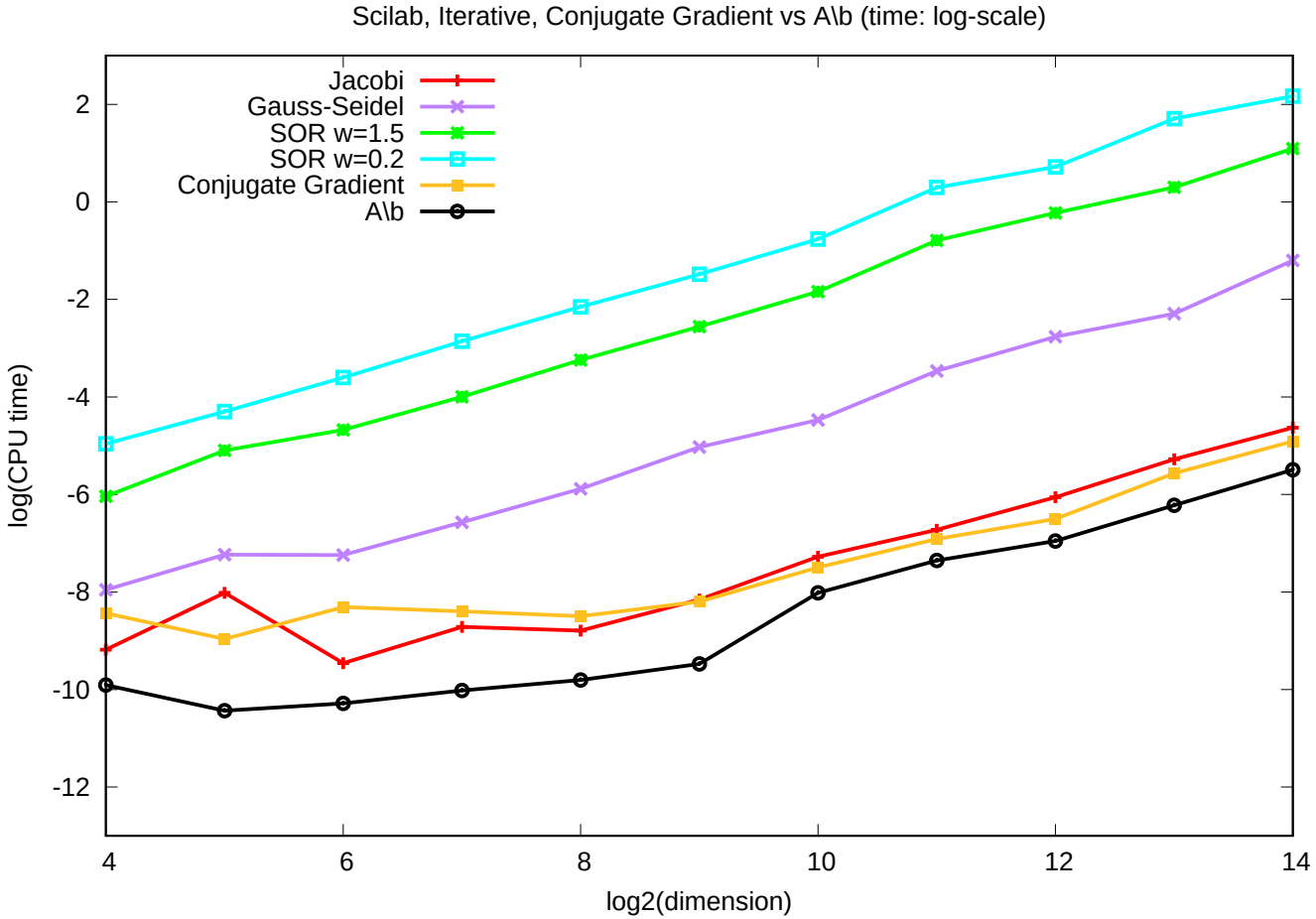

MATLAB, Iterative, Conjugate Gradient vs A\b
Jacobi
Gauss-Seidel
SOR w=1.5
SOR w=0.2
Conjugate Gradient
A\b
CPU time
log2(dimension)
0
5
10
15
20
25
4
6
8
10
12
14

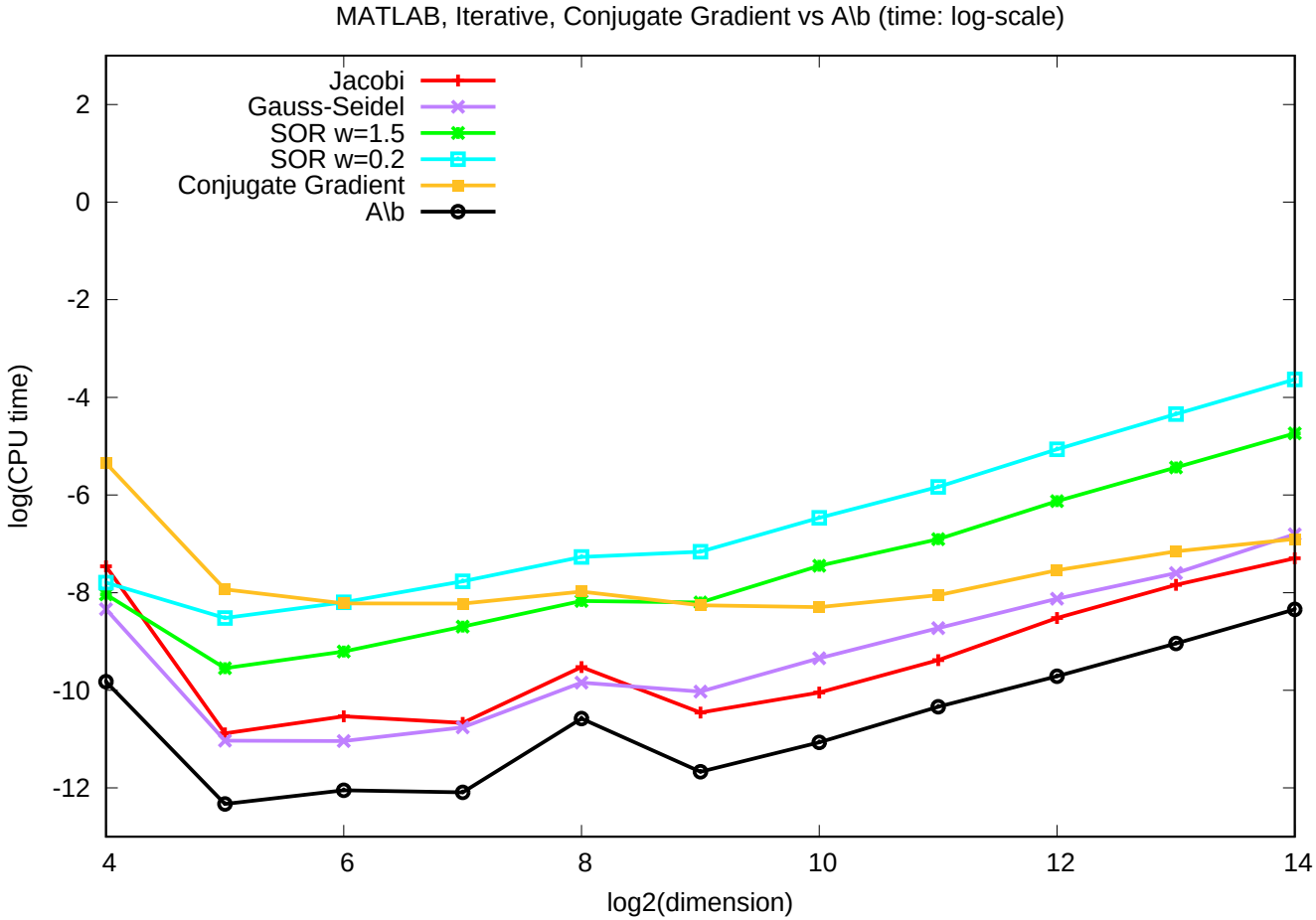
- 文献の記述から期待される結果と異なり, GNU Octave と Scilab において, SOR 法が, $w = 1.5$ と $w = 0.2$ の双方において, 他と比較して劇的に遅いことが確認できる.

- GNU Octave の実行時間は, Scilab の倍程度になっている.
- MATLAB はこれらと比較して圧倒的に速いため, グラフが潰れてしまっている.

- 続いて, 縦軸を実行時間の対数にしたグラフを示す.


GNU Octave, Iterative, Conjugate Gradient vs A\b (time: log-scale)
Jacobi
Gauss-Seidel
SOR w=1.5
SOR w=0.2
Conjugate Gradient
A\b
log(CPU time)
log2(dimension)
2
0
-2
-4
-6
-8
-10
-12
4
6
8
10
12
14


Scilab, Iterative, Conjugate Gradient vs A\b (time: log-scale)
Jacobi
Gauss-Seidel
SOR w=1.5
SOR w=0.2
Conjugate Gradient
A\b
log(CPU time)
log2(dimension)
2
0
-2
-4
-6
-8
-10
-12
4
6
8
10
12
14


MATLAB, Iterative, Conjugate Gradient vs A\b (time: log-scale)
Jacobi
Gauss-Seidel
SOR w=1.5
SOR w=0.2
Conjugate Gradient
A\b
log(CPU time)
log2(dimension)
2
0
-2
-4
-6
-8
-10
-12
4
6
8
10
12
14

- 縦軸を対数軸に取ることで, 各ソフトウェアおよびアルゴリズムの比較が容易になる (本来はこのようなグラフを描くべきであるが, この講義では, 強調のため, 敢えて見にくいグラフを作成している).

- 縦軸を対数軸に取ったグラフをよく見ると, GNU Octave および Scilab では, SOR 法ほどではないが, Gauss-Siedel 法の実行もかなり遅いことがわかる.

- MATLAB ではこのような現象は発生していないため, この現象は, アルゴリズムの特徴ではなく, GNU Octave および Scilab の特徴 (欠点) であることがわかる.

- 実はこれはプログラムの書き方に依存した問題で, GNU Octave や Scilab では, 大規模な問題を解くためのプログラムで for 文などの繰り返し構造を用いると, 実行が極端に遅くなる. Gauss-Siedel 法や SOR 法のアルゴリズムを書き下す際には for 文が不可避なので, このような結果になった. なお, Jacobi 法は for 文を使わずに書き下せる.

- 上記は, ユーザが書いた繰り返しの構造を含むスクリプトをそのソフトウェアが効率良く実行できるか否かという問題. MATLAB はこれが得意だが, GNU Octave と Scilab は不得手. アルゴリズムの良し悪しとは関係がない.

- 数値計算を専門とする研究者は, 自分で新しい数値計算のアルゴリズムを作り, その性能をコンピュータを使った数値実験でテストするのであるが, 性能評価に使ったソフトウェアによっては, 上述のような現象の影響を強く受けることには注意を要する.

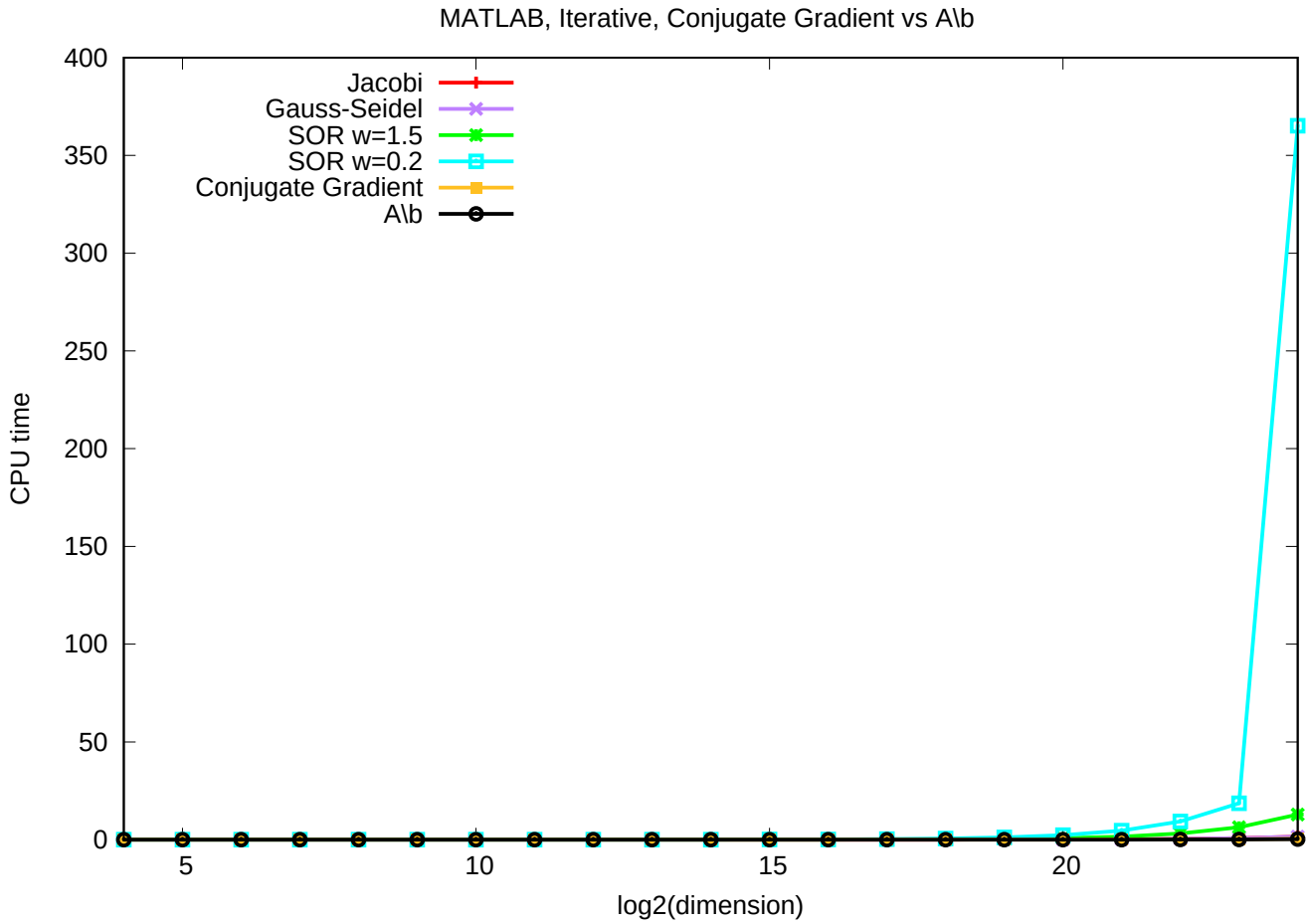
- 次に, より大規模な問題が解けるかどうかについて述べる.

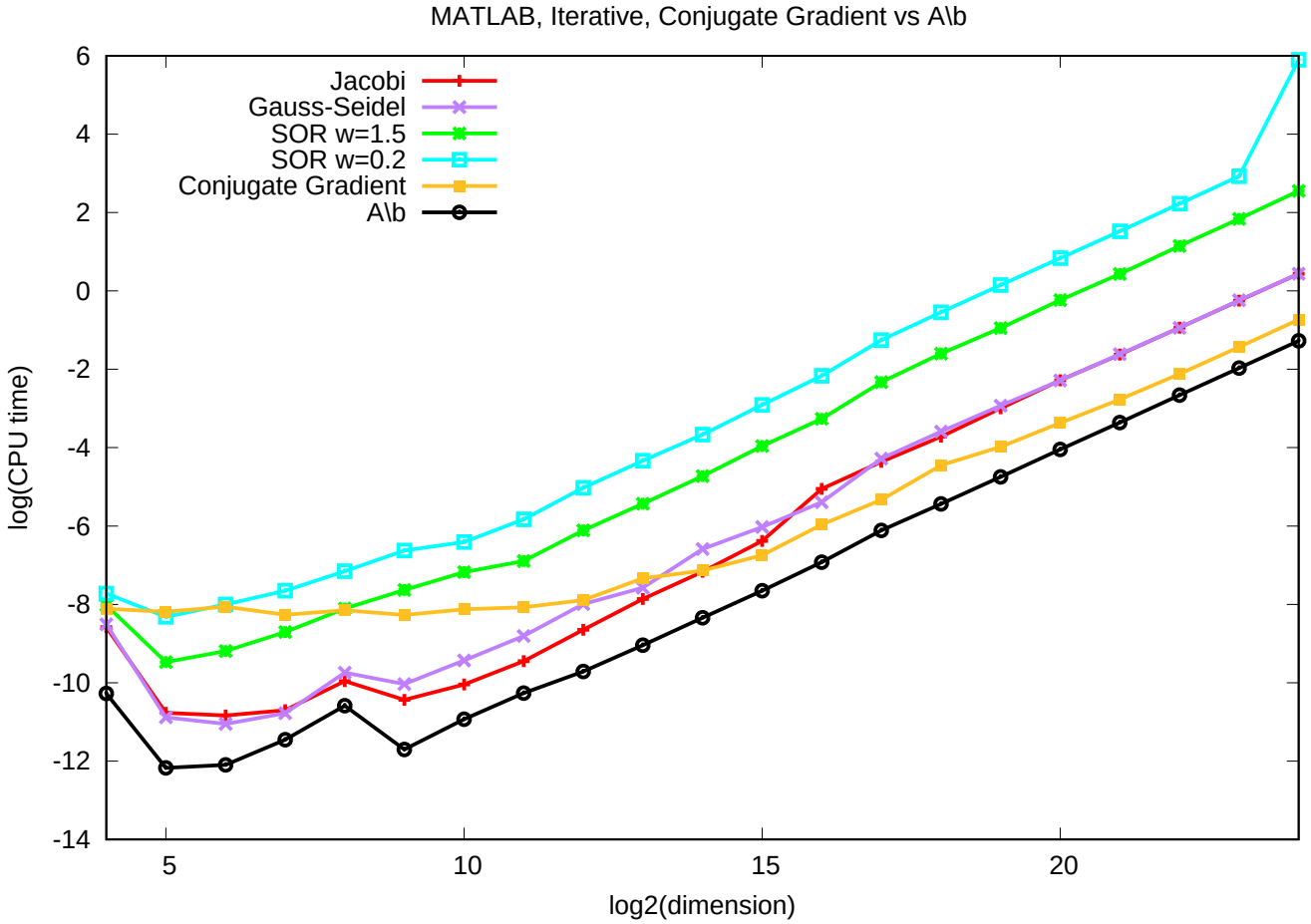
- Scilab は, 取り扱うことができる疎行列の大きさに制限があり, 上述のパソコンでは, 次元が 2^{15} までの問題しか正しく取り扱うことができなかった.
- GNU Octave および MATLAB の標準ソルバ ($\boldsymbol{A}\backslash\boldsymbol{b}$) は, 次元が 2^{28} (約 2.7 億) の問題まで解くことができた. 2^{29} ではメモリ不足になった.

- 今日のふつうのパソコンは, 適切なソフトウェアを選択すれば, この程度のことはできる.

- 今度は, 問題の次元を 2^4 から 2^{24}(約 168 万) までとし, MATLAB で, 上述の連立方程式を Jacobi 法, Gauss-Seidel 法, SOR (1.5) 法, SOR (0.2) 法, 前処理付き共役勾配法 (PCG 法), $\boldsymbol{A}\backslash\boldsymbol{b}$ (標準ソルバ) で 10 回解き, 平均計算時間を比較する.

- 先と同様に, 実行時間の相異を強調するため, まず縦軸を対数軸にしていないグラフを示してから, 続いて縦軸を対数軸にしたグラフを示す.

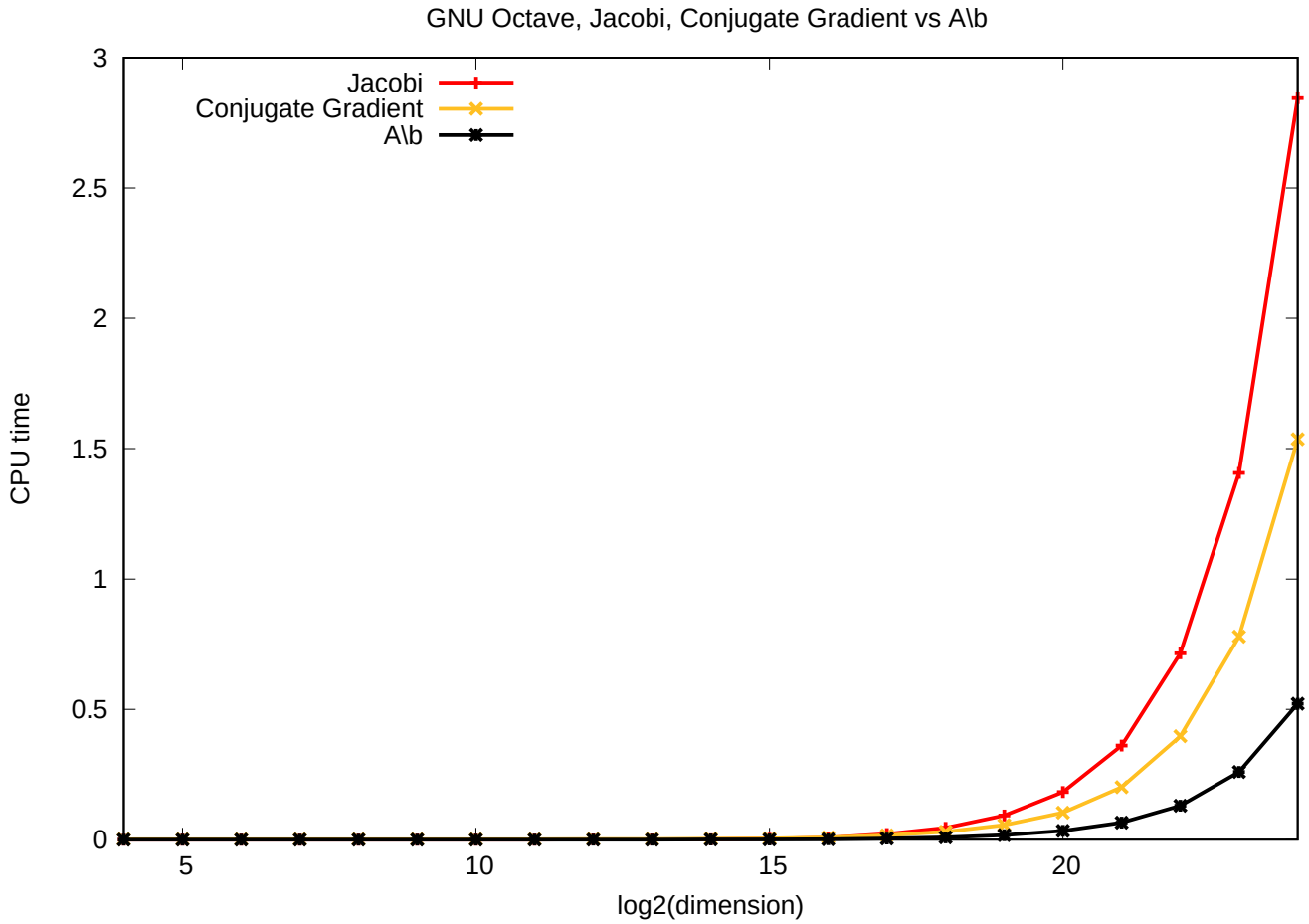

MATLAB, Iterative, Conjugate Gradient vs A\b
CPU time
log2(dimension)
Jacobi
Gauss-Seidel
SOR w=1.5
SOR w=0.2
Conjugate Gradient
A\b
400
350
300
250
200
150
100
50
0
5
10
15
20

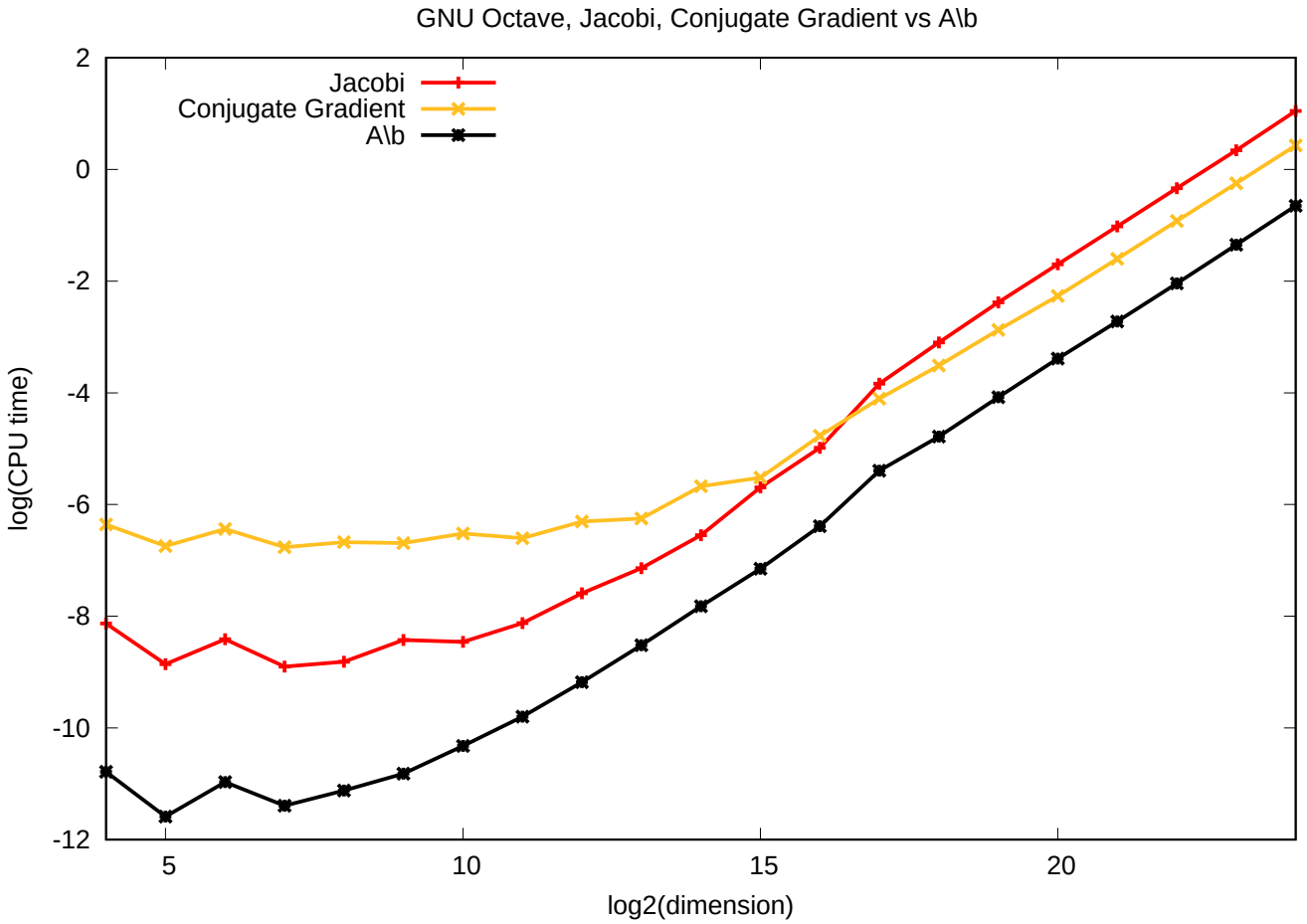

MATLAB, Iterative, Conjugate Gradient vs A\b
Jacobi
Gauss-Seidel
SOR w=1.5
SOR w=0.2
Conjugate Gradient
A\b
log(CPU time)
log2(dimension)
6
4
2
0
-2
-4
-6
-8
-10
-12
-14
5
10
15
20

- 結局のところ, MATLAB の標準ソルバ $\boldsymbol{A}\backslash\boldsymbol{b}$ は優秀であり, 大規模問題に適するとされる Jacobi 法も, 理論的には重要な共役勾配法も, 標準ソルバと競合できていない.

- スーパーコンピュータで専用のプログラムを書く場合には, これとは違った結果になるものと思われが…
- 普通のパソコンのメモリに格納できる程度の問題であれば, MATLAB の標準ソルバを使うのが効率が良い.

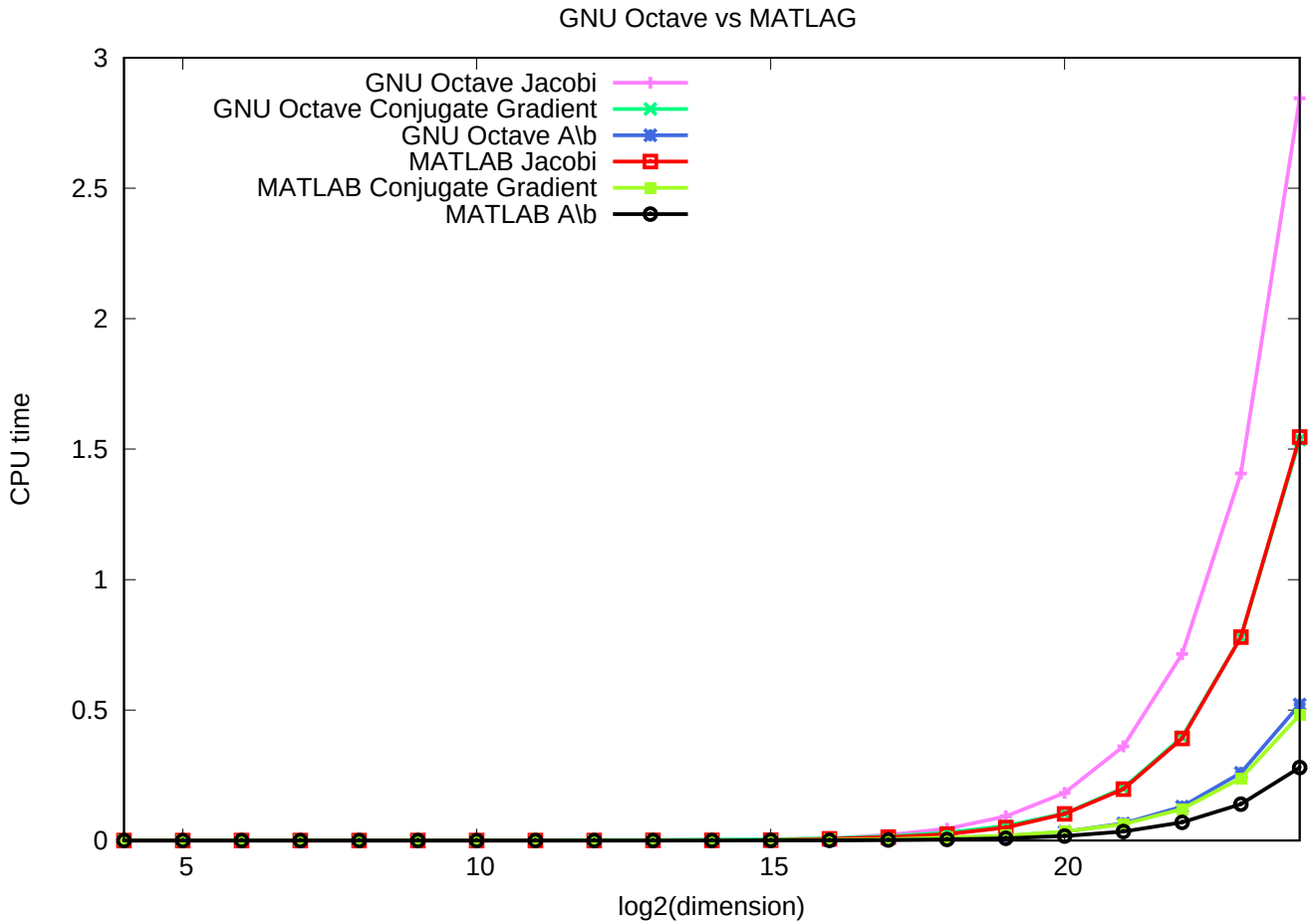
- GNU Octave では, Jacobi 法, 共役勾配法および標準ソルバについては, MATLAB と比較して, 多少劣る結果になる.
- これから, GNU Octave による実行結果を示す.

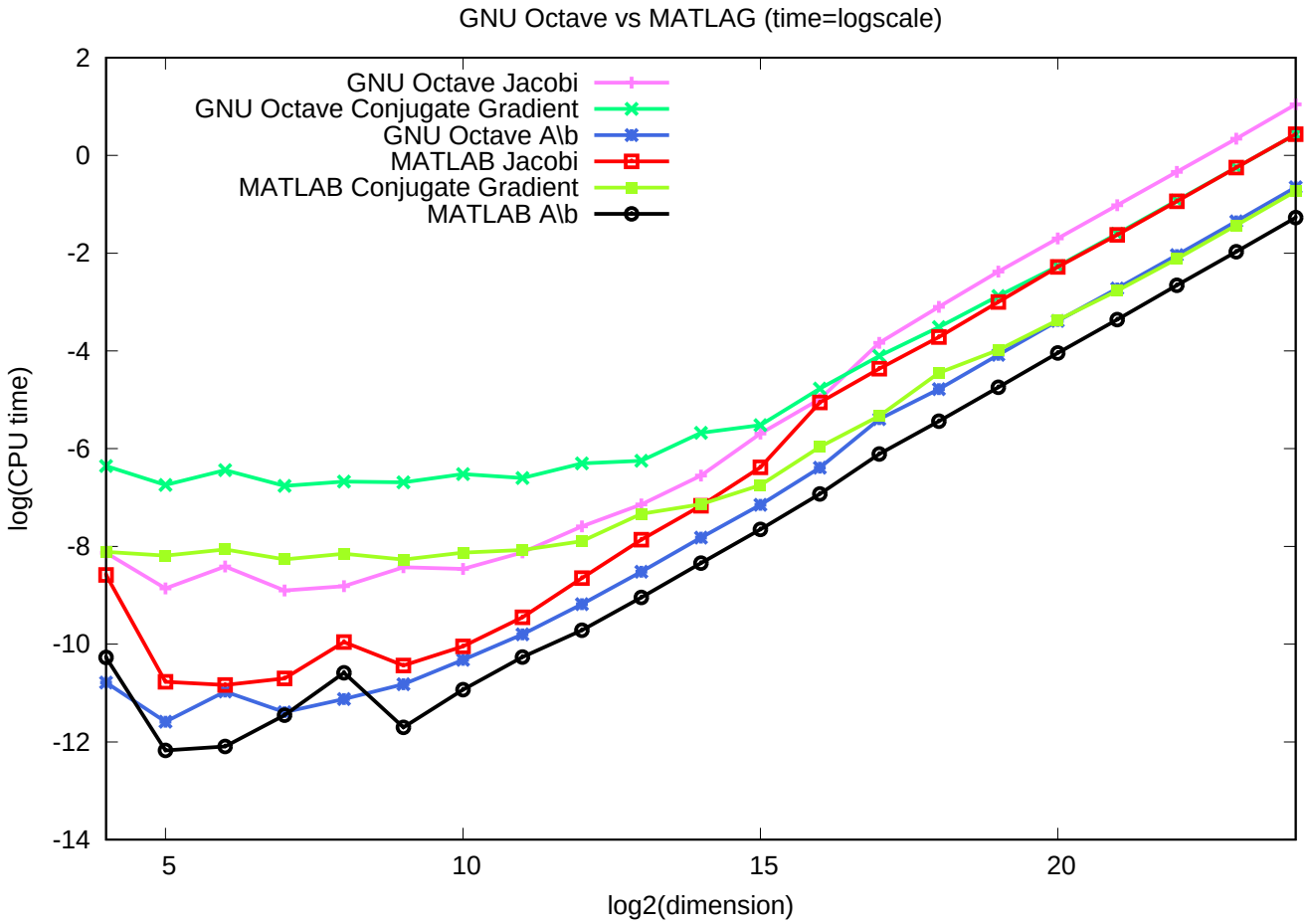

GNU Octave, Jacobi, Conjugate Gradient vs A\b
Jacobi
Conjugate Gradient
A\b
CPU time
log2(dimension)
0
0.5
1
1.5
2
2.5
3
5
10
15
20


GNU Octave, Jacobi, Conjugate Gradient vs A\b
Jacobi
Conjugate Gradient
A\b
log(CPU time)
log2(dimension)
2
0
-2
-4
-6
-8
-10
-12
5
10
15
20

- GNU Octave の標準ソルバ $\boldsymbol{A}\backslash\boldsymbol{b}$ は優秀であり，大規模問題に適するとされる Jacobi 法も，理論的には重要な共役勾配法も，標準ソルバと競合できていない．

- 最後に, GNU Octave と MATLAB の, Jacobi 法, 前処理付き共役勾配法, 標準ソルバ $\boldsymbol{A}\backslash\boldsymbol{b}$ の実行測度を比較する.
- 先の結果から該当箇所を抜き出して重ね書きしたものが, 以下のグラフになる.


GNU Octave vs MATLAG
GNU Octave Jacobi
GNU Octave Conjugate Gradient
GNU Octave A\b
MATLAB Jacobi
MATLAB Conjugate Gradient
MATLAB A\b
CPU time
log2(dimension)
0
0.5
1
1.5
2
2.5
3
5
10
15
20


GNU Octave vs MATLAG (time=logscale)
GNU Octave Jacobi
GNU Octave Conjugate Gradient
GNU Octave A\b
MATLAB Jacobi
MATLAB Conjugate Gradient
MATLAB A\b
log(CPU time)
log2(dimension)
2
0
-2
-4
-6
-8
-10
-12
-14
5
10
15
20

- 全体的に見て, GNU Octaveの処理測度はMATLABの半分程度ではあるが, スクリプト中でfor文等の繰り返し構造を避けるようにしておけば, 十分実用になるレベルではある.
- 先に述べたように, Scilabは大規模問題は不向き.