筑波大学計算科学研究センター
CCS HPCサマーセミナー
「並列数値アルゴリズムI」

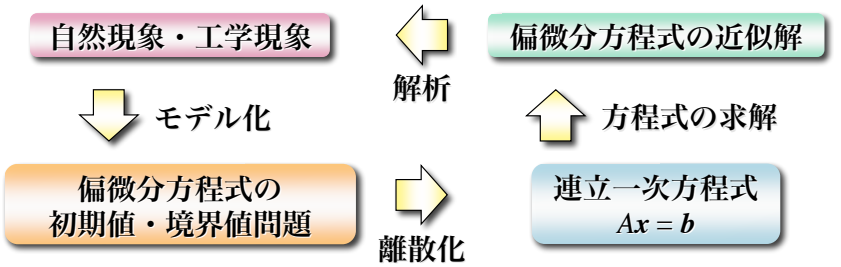
多田野寛人
tadano@cs.tsukuba.ac.jp
筑波大学大学院システム情報工学研究科
計算科学研究センター

講義内容

- 連立一次方程式の求解法
- Cell Broadband Engine における数値計算
- レポート課題について

連立一次方程式の求解法

自然現象・工学現象の解析



連立一次方程式は様々な分野における
数値シミュレーションで現れ，
計算時間の大部分が求解に費やされている

連立一次方程式

連立一次方程式：$Ax = b$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

連立一次方程式は様々な分野における
数値シミュレーションで現れ，
計算時間の大部分が求解に費やされている

直接解法と反復解法

直接解法

Gauss消去法，LU分解など

1) 係数行列 A を変形するため，非零要素数が増大
   ➡ 係数行列の疎性が利用出来ない
2) 有限回の演算で必ず解くことが出来る

反復解法

Krylov部分空間反復法

1) 必要な演算は係数行列 A とベクトルの積，内積など
   ➡ 係数行列の疎性がそのまま使える
2) 問題によっては多くの反復回数を要することがある

Krylov部分空間反復法

連立一次方程式 $Ax = b$ の近似解生成

近似解の生成条件　$\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{z}_k$，　$\boldsymbol{z}_k \in \mathcal{K}_k(A; \boldsymbol{r}_0)$

Krylov部分空間：$\mathcal{K}_k(A; \boldsymbol{r}_0) = \mathrm{span}(\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{k-1}\boldsymbol{r}_0)$

残差ベクトル　$\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k = \boldsymbol{r}_0 - A\boldsymbol{z}_k$

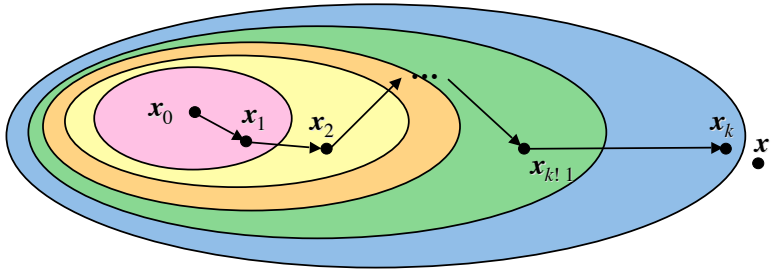


Fig. Krylov部分空間法の概念図.

Hermite行列に対する解法

1. 係数行列が Hermite行列（$A = A^{\mathrm{H}}$）の場合

- 共役勾配法（**C**onjugate **G**radient method: CG法）
- 共役残差法（**C**onjugate **R**esidual method: CR法）
- 最小残差法（**Min**inal **Res**idual Method: MINRES法）

係数行列の Hermite性を使うことで
短い漸化式（計算量が少ない）
アルゴリズムが導出できる

補足：Hermite行列

$A = A^{\mathrm{H}} = \bar{A}^{\mathrm{T}}$
$(a_{ij} = \bar{a}_{ji})$

Hermite行列に対する解法

x_0 is an initial guess,
Compute $r_0 = b - Ax_0$,
Set $p_0 = r_0$,
For $k = 0, 1, \ldots,$ **until** $\|r_k\|_2 \le \varepsilon_{\mathrm{TOL}}\|b\|_2$ **do** :

$$q_k = Ap_k,$$ ← 行列ベクトル積

$$\alpha_k = \frac{(r_k, r_k)}{(p_k, q_k)},$$ ← 内積

$$x_{k+1} = x_k + \alpha_k p_k,$$
$$r_{k+1} = r_k - \alpha_k q_k,$$ ← ベクトルの定数倍と加算

$$\beta_k = \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)},$$ ← 内積

$$p_{k+1} = r_{k+1} + \beta_k p_k,$$ ← ベクトルの定数倍と加算

End for

Fig. 共役勾配法（CG法）

非Hermite行列に対する解法

2. 係数行列が 非Hermite行列（$A \neq A^{\mathrm{H}}$）の場合

残差の双直交条件から導出される解法

- 双共役勾配法（**Bi**-Conjugate **G**radient: BiCG法）
- 自乗共役勾配法（Conjugate **R**esidual **S**quared: CGS法）
- 双共役勾配安定化法（**BiCG Stab**ilization: BiCGSTAB法）

⇒ 計算量は少ないが，残差は単調減少しない

残差の最小条件から導出される解法

- 一般化共役残差法（**G**eneralized **C**onjugate **R**esidual: GCR法）
- 一般化最小残差法（**G**eneralized **M**inimal **Res**idual: GMRES法）

⇒ 残差は単調減少するが，長い漸化式が必要

非Hermite行列に対する解法

x_0 is an initial guess,
Compute $r_0 = b - Ax_0$,
Choose r_0^* such that $(r_0^*, r_0) \neq 0$,
Set $p_0 = r_0$ and $p_0^* = r_0^*$,
For $k = 0, 1, \ldots,$ **until** $\|r_k\|_2 \le \varepsilon_{\mathrm{TOL}}\|b\|_2$ **do** :

$$q_k = Ap_k, \qquad q_k^* = A^{\mathrm{H}}p_k^*,$$
$$\alpha_k = \frac{(r_k^*, r_k)}{(p_k^*, q_k)},$$
$$x_{k+1} = x_k + \alpha_k p_k,$$
$$r_{k+1} = r_k - \alpha_k q_k, \qquad r_{k+1}^* = r_k^* - \bar{\alpha}_k q_k^*,$$
$$\beta_k = \frac{(r_{k+1}^*, r_{k+1})}{(r_k^*, r_k)},$$
$$p_{k+1} = r_{k+1} + \beta_k p_k, \qquad p_{k+1}^* = r_{k+1}^* + \bar{\beta}_k p_k^*,$$

End for

行列ベクトル積
内積
ベクトルの定数倍と加算

Fig. 双共役勾配法（BiCG法）

非Hermite行列に対する解法

x_0 is an initial guess,
Compute $r_0 = b - Ax_0$,
Set $p_0 = r_0$ and $q_0 = s_0 = Ar_0$,
For $k = 0, 1, \ldots,$ **until** $\|r_k\|_2 \le \varepsilon_{\mathrm{TOL}}\|b\|_2$ **do :**

$$\alpha_k = \frac{(q_k, r_k)}{(q_k, q_k)},$$
$$x_{k+1} = x_k + \alpha_k p_k,$$
$$r_{k+1} = r_k - \alpha_k q_k,$$
$$s_{k+1} = Ar_{k+1},$$
$$\beta_{k,i} = -\frac{(q_i, s_{k+1})}{(q_i, Aq_i)}, \quad (i = 0, 1, \ldots, k)$$
$$p_{k+1} = r_{k+1} + \sum_{i=0}^{k} \beta_{k,i} p_i,$$
$$q_{k+1} = s_{k+1} + \sum_{i=0}^{k} \beta_{k,i} q_i,$$

End for

- 行列ベクトル積は1反復あたり1回
- 長い漸化式を使うため多くのメモリが必要
- リスタートにより演算量とメモリ量を削減

Fig. 一般化共役残差法（GCR法）

反復法の収束特性

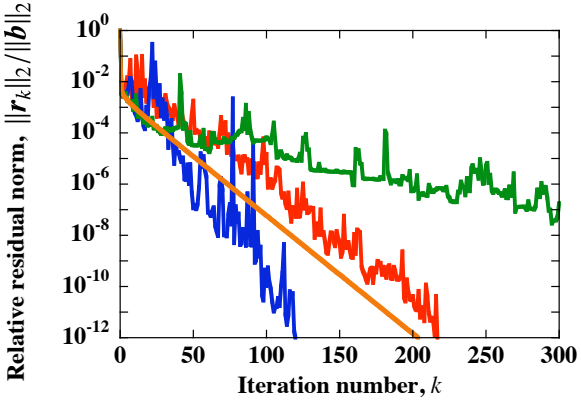


Fig. 反復法の相対残差履歴

—：BiCG法，—：CGS法，—：BiCGSTAB法，—：GCR法

複素対称行列に対する解法

3. 係数行列が複素対称行列（$A = A^{\mathrm{T}} \neq A^{\mathrm{H}}$）の場合

・共役直交共役勾配法
（**C**onjugate **O**rthogonal **C**onjugate **G**radient: COCG法）

係数行列が複素対称行列の場合は，
1反復あたり1回の行列ベクトル積で，
かつ短い漸化式で計算ができる

補足：複素対称行列
$A = A^{\mathrm{T}} \neq A^{\mathrm{H}}$
$(a_{ij} = a_{ji} \neq \bar{a}_{ji})$

複素対称行列に対する解法

x_0 is an initial guess,
Compute $r_0 = b - Ax_0$,
Set $p_0 = r_0$,
For $k = 0, 1, \ldots,$ **until** $\|r_k\|_2 \leq \varepsilon_{\mathrm{TOL}}\|b\|_2$ **do** :

$q_k = Ap_k,$ ← 行列ベクトル積
$\alpha_k = \dfrac{(\bar{r}_k, r_k)}{(\bar{p}_k, q_k)},$ ← 内積
$x_{k+1} = x_k + \alpha_k p_k,$
$r_{k+1} = r_k - \alpha_k q_k,$ ← ベクトルの定数倍と加算
$\beta_k = \dfrac{(\bar{r}_{k+1}, r_{k+1})}{(\bar{r}_k, r_k)},$ ← 内積
$p_{k+1} = r_{k+1} + \beta_k p_k,$ ← ベクトルの定数倍と加算
End for

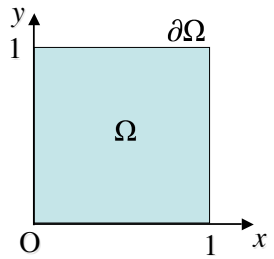
Fig. 共役直交共役勾配法（COCG法）

疎行列が現れる例

2次元 Poisson 問題

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = f, & \text{in } \Omega \\ u = \bar{u}, & \text{on } \partial\Omega \end{cases}$$

$f, \bar{u}$ は既知の関数.



をx, y 方向に $(M+1)$等分し，5点中心差分で離散化

$M!$ M行列をもつ連立一次方程式に帰着
行列の要素数：M^4 個
非零要素の数：$5M^2$ ！ $4M$ 個

疎行列の格納形式

Compressed Row Storage (CRS) 形式
非零要素を行方向に探索

$$A = \begin{bmatrix} 1 & 0 & 3 & 0 & 2 \\ 0 & 1 & 0 & 4 & 6 \\ 2 & 5 & 3 & 0 & 0 \\ 0 & 0 & 7 & 8 & 0 \\ 0 & 1 & 0 & 3 & 9 \end{bmatrix}$$

val: 非零要素を格納する配列

col_ind: 非零要素の列番号を格納

row_ptr: 各行の先頭の非零要素が格納されている場所を指し示す配列

val :	1	3	2	1	4	6	2	5	3	7	8	1	3	9
col_ind :	1	3	5	2	4	5	1	2	3	3	4	2	4	5
row_ptr :	1	4	7	10	12	15								

非零要素数+1 の値を格納

疎行列の格納形式

Compressed Column Storage (CCS) 形式
非零要素を列方向に探索

$$A = \begin{bmatrix} 1 & 0 & 3 & 0 & 2 \\ 0 & 1 & 0 & 4 & 6 \\ 2 & 5 & 3 & 0 & 0 \\ 0 & 0 & 7 & 8 & 0 \\ 0 & 1 & 0 & 3 & 9 \end{bmatrix}$$

val: 非零要素を格納する配列

row_ind: 非零要素の行番号を格納

col_ptr: 各列の先頭の非零要素が格納されている場所を指し示す配列

val :	1	2	1	5	1	3	3	7	4	8	3	2	6	9
row_ind :	1	3	2	3	5	1	3	4	2	4	5	1	2	5
col_ptr :	1	3	6	9	12	15								

非零要素数+1 の値を格納

CRS形式の行列ベクトル積

行列 A とベクトル x の積 $y = Ax$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Fortran Code

```
do i=1,n
  y(i) = 0.0D0
  do j=row_ptr(i),row_ptr(i+1)-1
    y(i) = y(i)+val(j)*x(col_ind(j))
  end do
end do
```


CCS形式の行列ベクトル積

行列 A とベクトル x の積 $y = Ax$

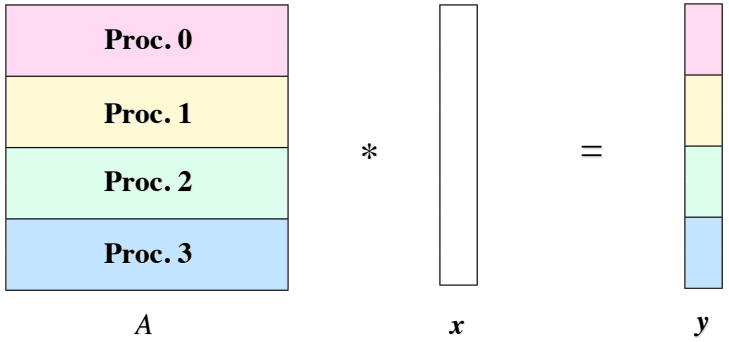
$$y = [a_1, a_2, \dots, a_n] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} a_i x_i$$

Fortran Code

```
do i=1,n
  y(i) = 0.0D0
end do
do j=1,n
  do i=col_ptr(j),col_ptr(j+1)-1
    y(row_ind(i)) = y(row_ind(i))+val(i)*x(j)
  end do
end do
```

行列ベクトル積の並列化

・CRS形式の場合の $y = Ax$ の計算



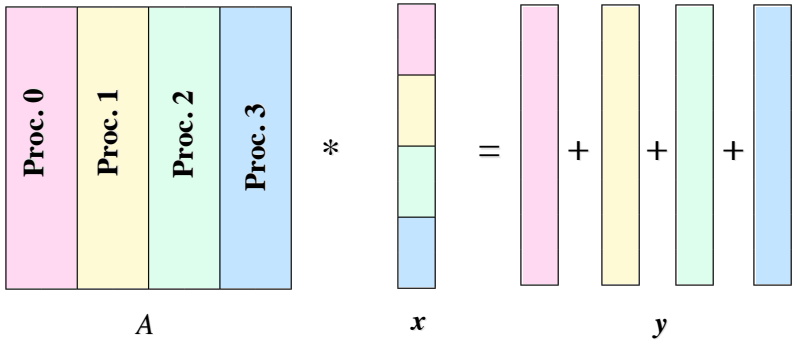
各プロセスで分散してデータを保持する

全てのプロセスで保持する

MPI_Gather で Proc. 0 に集める

行列ベクトル積の並列化

・CCS形式の場合の $y = Ax$ の計算



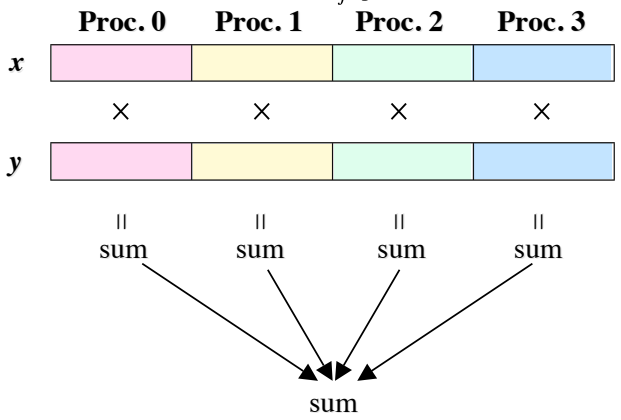
各プロセスで分散してデータを保持する

分散して保持する

MPI_Reduce でProc. 0に結果を足し合わせて送る

内積の並列化

$$(x, y) = \sum_{j=1}^{n} \bar{x}_j y_j$$



MPI_Reduce で Proc.0 に集める

MPIコード例

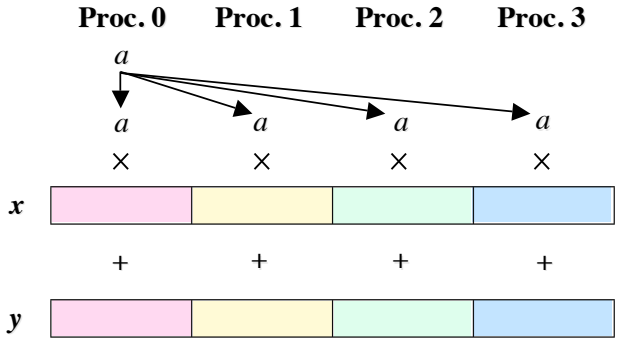
$$(x, y) = \sum_{j=1}^{n} \bar{x}_j y_j$$

```
program main
include 'mpif.h'
...
call mpi_init(ierr)
call mpi_comm_size(mpi_comm_world, nprocs, ierr)
call mpi_comm_rank(mpi_comm_world, myrank, ierr)
...
tmp_sum = (0.0D0, 0.0D0)
do i=istart(myrank+1), iend(myrank+1)
   tmp_sum = tmp_sum + conj(x(i)) * y(i)
end do

call mpi_reduce(tmp_sum, sum, 1, mpi_double_complex,
                mpi_sum, 0, mpi_comm_world, ierr)

call mpi_finalize(ierr)
```

ベクトルの定数倍と加算の並列化

$y = y + ax$

a を MPI_Bcast で全プロセスに送る



連立一次方程式の前処理法

Krylov部分空間反復法で，残差が収束しないことがある

Krylov 部分空間法の特徴

係数行列が単位行列に近い場合，少ない反復回数で残差が収束

連立一次方程式 $Ax = b$ → 前処理 → 前処理後の連立一次方程式 $A'x' = b'$

係数行列 A' が単位行列に近くなるように変形！

連立一次方程式の前処理法

係数行列 A を近似する前処理法

$A \approx K_1 K_2 \iff K_1^{-1} A K_2^{-1} \approx I$

これを用いて

$Ax = b \iff (K_1^{-1} A K_2^{-1})(K_2 x) = K_1^{-1} b$

逆行列 A^{-1} を近似する前処理法

$A \approx M^{-1} \iff AM \approx I,\ MA \approx I$

これを用いて

$Ax = b \iff MAx = Mb$

または $Ax = b \iff (AM)(M^{-1}x) = b$

近似逆行列前処理

逆行列 A^{-1} を近似する行列 M を生成する前処理

$F(M) = \|I - AM\|_F^2$ を最小にするように M を決定

$$F(M) = \|I - AM\|_F^2 = \sum_{j=1}^{n} \|e_j - Am_j\|_2^2$$

・行列 M の非零構造は，任意に選択できる
・M を密行列とすれば，$M = A^{-1}$ となる

互いに独立な n 個の最小自乗問題のため，並列処理が可能！

補足：フロベニウスノルム

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2}$$

多項式前処理

逆行列 A^{-1} の近似を A の多項式で生成する

行列のNeumann展開による多項式生成

$\|I - A\| < 1$ の場合

$$A^{-1} = [I - (I - A)]^{-1} = \sum_{j=0}^{\infty} (I - A)^j$$

より，m次までで打ち切った行列を M とすると

$$A^{-1} \approx M = \sum_{j=0}^{m} (I - A)^j$$

・実際に行列 M は生成せず，反復法内で行列ベクトル積で計算
・行列ベクトル積の並列化により，同前処理を並列化できる

前処理付き反復法の収束特性

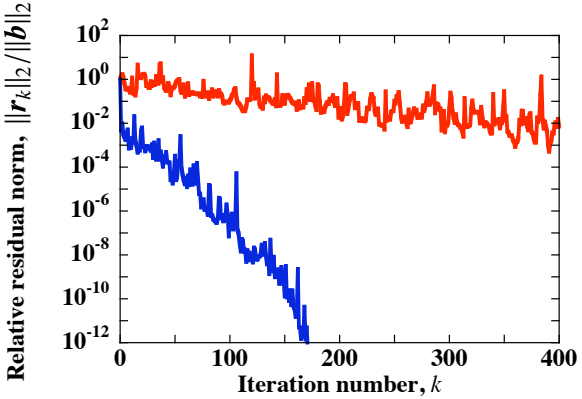
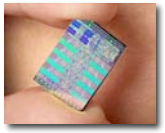


Fig. 反復法の相対残差履歴
－：BiCG法，－：近似逆行列前処理付きBiCG法

Cell Broadband Engineにおける数値計算

Cell Broadband Engine



・1つのPower Processor Element (PPE) と複数個の Synergistic Processing Element (SPE) から構成
・PlayStation 3 に搭載され，一般にも普及している．

PPE と SPE の特徴

PPE : 主に SPE の制御を担当．演算は苦手．
SPE : 単精度演算を得意とする．単精度の場合1SPEあたりの理論ピーク性能は 25.6Gflops．PS3では6個のSPEが使えるため，153.6Gflops.

SPEの性能を生かせるアルゴリズムが必要

Krylov部分空間法の前処理

Krylov部分空間反復法の特徴

係数行列 A が単位行列に近い場合，少ない反復回数で収束

連立一次方程式

$Ax = b$

左前処理

$MAx = Mb$

右前処理

$AMy = b,\ \ y = M^{-1}x$

$M:\ MA \approx I$ 及び $AM \approx I$ を満たす前処理行列.

前処理付きBiCGSTAB法

左前処理

x_0 is an initial guess,
compute $r_0 = p_0 = M(b - Ax_0)$,
choose r_0^* such that $(r_0^*, r_0) \neq 0$,
for $k = 0, 1, \ldots,$ **until** $\|r_k\|_2 \leq \varepsilon\|b\|_2$ **do**:
begin

$u_k = Ap_k,$
$v_k = Mu_k,$
$\alpha_k = (r_0^*, r_k)/(r_0^*, v_k),$
$t_k = r_k - \alpha_k v_k,$
$s_k = At_k,$
$q_k = Ms_k,$
$\zeta_k = (q_k, t_k)/(q_k, q_k),$
$x_{k+1} = x_k + \alpha_k p_k + \zeta_k t_k,$
$r_{k+1} = t_k - \zeta_k q_k,$
$\beta_k = (\alpha_k/\zeta_k) \cdot (r_0^*, r_{k+1})/(r_0^*, r_k),$
$p_{k+1} = r_{k+1} + \beta_k(p_k - \zeta_k v_k),$

end

前処理部分

右前処理

x_0 is an initial guess,
compute $r_0 = p_0 = b - Ax_0$,
choose r_0^* such that $(r_0^*, r_0) \neq 0$,
for $k = 0, 1, \ldots,$ **until** $\|r_k\|_2 \leq \varepsilon\|b\|_2$ **do**:
begin

$u_k = Mp_k,$
$v_k = Au_k,$
$\alpha_k = (r_0^*, r_k)/(r_0^*, v_k),$
$t_k = r_k - \alpha_k v_k,$
$s_k = Mt_k,$
$q_k = As_k,$
$\zeta_k = (q_k, t_k)/(q_k, q_k),$
$x_{k+1} = x_k + \alpha_k u_k + \zeta_k s_k,$
$r_{k+1} = t_k - \zeta_k q_k,$
$\beta_k = (\alpha_k/\zeta_k) \cdot (r_0^*, r_{k+1})/(r_0^*, r_k),$
$p_{k+1} = r_{k+1} + \beta_k(p_k - \zeta_k v_k),$

end

精度混合型アルゴリズム

x_0 is an initial guess,
compute $r_0 = p_0 = b - Ax_0$,
choose r_0^* such that $(r_0^*, r_0) \neq 0$,
for $k = 0, 1, \ldots,$ **until** $\|r_k\|_2 \leq \varepsilon\|b\|_2$ **do**:
begin

$u_k = Mp_k,$ 単精度計算
$v_k = Au_k,$
$\alpha_k = (r_0^*, r_k)/(r_0^*, v_k),$
$t_k = r_k - \alpha_k v_k,$
$s_k = Mt_k,$ 単精度計算
$q_k = As_k,$
$\zeta_k = (q_k, t_k)/(q_k, q_k),$
$x_{k+1} = x_k + \alpha_k u_k + \zeta_k s_k,$
$r_{k+1} = t_k - \zeta_k q_k,$
$\beta_k = (\alpha_k/\zeta_k) \cdot (r_0^*, r_{k+1})/(r_0^*, r_k),$
$p_{k+1} = r_{k+1} + \beta_k(p_k - \zeta_k v_k),$

end 倍精度計算

前処理計算部分

$u_k = Mp_k,\ \ s_k = Mt_k$

多くの演算量が必要！

精度混合アプローチ

- 前処理計算部分
  - 単精度計算で高速化
- その他の部分
  - 倍精度計算を行う

誤差の影響（左前処理）

第 k ステップの漸化式

$u_k = Ap_k,$
[1] $v_k = Mu_k,$
$\alpha_k = (r_0^*, r_k)/(r_0^*, v_k),$
[2] $t_k = r_k - \alpha_k v_k,$
$s_k = At_k,$
[3] $q_k = Ms_k,$
$\zeta_k = (q_k, t_k)/(q_k, q_k),$
[4] $x_{k+1} = x_k + \alpha_k p_k + \zeta_k t_k,$
$r_{k+1} = t_k - \zeta_k q_k,$
$\beta_k = (\alpha_k/\zeta_k) \cdot (r_0^*, r_{k+1})/(r_0^*, r_k),$
$p_{k+1} = r_{k+1} + \beta_k(p_k - \zeta_k v_k),$

[1] v_k には前処理部分で誤差が混入

$\tilde{v}_k = v_k + \delta v_k,\ \ \delta v_k$：$v_k$ の誤差

[2] α_k, t_k, s_k にも誤差が混入

$\alpha_k \Rightarrow \tilde{\alpha}_k,\ \ t_k \Rightarrow \tilde{t}_k,\ \ s_k \Rightarrow \tilde{s}_k$

[3] q_k には前処理部分で誤差が混入

$\tilde{q}_k = q_k + \delta q_k,\ \delta q_k$：$q_k$ の誤差

[4] $\zeta_k, x_{k+1}, r_{k+1}$ にも誤差が混入

$\zeta_k \Rightarrow \tilde{\zeta}_k,\ \ x_{k+1} \Rightarrow \tilde{x}_{k+1},\ \ r_{k+1} \Rightarrow \tilde{r}_{k+1}$

記号 ~ は誤差が混入した値を表す

誤差の影響（左前処理）

第 $(k+1)$ 番目の残差ベクトル

$$
\begin{aligned}
\tilde{\boldsymbol{r}}_{k+1} &= \tilde{\boldsymbol{t}}_k - \tilde{\zeta}_k \tilde{\boldsymbol{q}}_k \\
&= \boldsymbol{r}_k - \tilde{\alpha}_k \tilde{\boldsymbol{v}}_k - \tilde{\zeta}_k \tilde{\boldsymbol{q}}_k \\
&= \boldsymbol{r}_k - \tilde{\alpha}_k (\boldsymbol{v}_k + \delta \boldsymbol{v}_k) - \tilde{\zeta}_k (\boldsymbol{q}_k + \delta \boldsymbol{q}_k) \\
&= \boldsymbol{r}_k - \tilde{\alpha}_k (M A \boldsymbol{p}_k + \delta \boldsymbol{v}_k) - \tilde{\zeta}_k (M A \tilde{\boldsymbol{t}}_k + \delta \boldsymbol{q}_k) \\
&= M \left[\boldsymbol{b} - A(\boldsymbol{x}_k + \tilde{\alpha}_k + \tilde{\zeta}_k \tilde{\boldsymbol{t}}_k) \right] - \tilde{\alpha}_k \delta \boldsymbol{v}_k - \tilde{\zeta}_k \delta \boldsymbol{q}_k \\
&= M(\boldsymbol{b} - A \tilde{\boldsymbol{x}}_{k+1}) - \tilde{\alpha}_k \delta \boldsymbol{v}_k - \tilde{\zeta}_k \delta \boldsymbol{q}_k
\end{aligned}
$$

残差と近似解の関係式 $\tilde{\boldsymbol{r}}_{k+1} = M(\boldsymbol{b} - A\tilde{\boldsymbol{x}}_{k+1})$ が満たされない！

誤差の影響（右前処理）

第 k ステップの漸化式

$$
\begin{aligned}
[1]\quad & \boldsymbol{u}_k = M \boldsymbol{p}_k, \\
& \boldsymbol{v}_k = A \boldsymbol{u}_k, \\
[2]\quad & \alpha_k = (\boldsymbol{r}_0^*, \boldsymbol{r}_k)/(\boldsymbol{r}_0^*, \boldsymbol{v}_k), \\
& \boldsymbol{t}_k = \boldsymbol{r}_k - \alpha_k \boldsymbol{v}_k, \\
[3]\quad & \boldsymbol{s}_k = M \boldsymbol{t}_k, \\
& \boldsymbol{q}_k = A \boldsymbol{s}_k, \\
& \zeta_k = (\boldsymbol{q}_k, \boldsymbol{t}_k)/(\boldsymbol{q}_k, \boldsymbol{q}_k), \\
[4]\quad & \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{u}_k + \zeta_k \boldsymbol{s}_k, \\
& \boldsymbol{r}_{k+1} = \boldsymbol{t}_k - \zeta_k \boldsymbol{q}_k, \\
& \beta_k = (\alpha_k/\zeta_k) \cdot (\boldsymbol{r}_0^*, \boldsymbol{r}_{k+1})/(\boldsymbol{r}_0^*, \boldsymbol{r}_k), \\
& \boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k (\boldsymbol{p}_k - \zeta_k \boldsymbol{v}_k),
\end{aligned}
$$

[1] $\boldsymbol{u}_k$ には前処理部分で誤差が混入
$\tilde{\boldsymbol{u}}_k = \boldsymbol{u}_k + \delta \boldsymbol{u}_k$, $\delta \boldsymbol{u}_k$: $\boldsymbol{u}_k$ の誤差

[2] $\boldsymbol{v}_k, \alpha_k, \boldsymbol{t}_k$ にも誤差が混入
$\boldsymbol{v}_k \Rightarrow \tilde{\boldsymbol{v}}_k$, $\alpha_k \Rightarrow \tilde{\alpha}_k$, $\boldsymbol{t}_k \Rightarrow \tilde{\boldsymbol{t}}_k$

[3] $\boldsymbol{s}_k$ には前処理部分で誤差が混入
$\tilde{\boldsymbol{s}}_k = \boldsymbol{s}_k + \delta \boldsymbol{s}_k$, $\delta \boldsymbol{s}_k$: $\boldsymbol{s}_k$ の誤差

[4] $\boldsymbol{q}_k, \zeta_k, \boldsymbol{x}_{k+1}, \boldsymbol{r}_{k+1}$ にも誤差が混入
$\boldsymbol{q}_k \Rightarrow \tilde{\boldsymbol{q}}_k$, $\zeta_k \Rightarrow \tilde{\zeta}_k$,
$\boldsymbol{x}_{k+1} \Rightarrow \tilde{\boldsymbol{x}}_{k+1}$, $\boldsymbol{r}_{k+1} \Rightarrow \tilde{\boldsymbol{r}}_{k+1}$

誤差の影響（右前処理）

第 $(k+1)$ 番目の残差ベクトル

$$
\begin{aligned}
\tilde{\boldsymbol{r}}_{k+1} &= \tilde{\boldsymbol{t}}_k - \tilde{\zeta}_k \tilde{\boldsymbol{q}}_k \\
&= \boldsymbol{r}_k - \tilde{\alpha}_k \tilde{\boldsymbol{v}}_k - \tilde{\zeta}_k \tilde{\boldsymbol{q}}_k \\
&= \boldsymbol{r}_k - \tilde{\alpha}_k A \tilde{\boldsymbol{u}}_k - \tilde{\zeta}_k A \tilde{\boldsymbol{s}}_k \\
&= \boldsymbol{b} - A(\boldsymbol{x}_k + \tilde{\alpha}_k \tilde{\boldsymbol{u}}_k + \tilde{\zeta}_k \tilde{\boldsymbol{s}}_k) \\
&= \boldsymbol{b} - A \tilde{\boldsymbol{x}}_{k+1}
\end{aligned}
$$

残差と近似解の関係式 $\tilde{\boldsymbol{r}}_{k+1} = \boldsymbol{b} - A\tilde{\boldsymbol{x}}_{k+1}$ が満たされる

多項式前処理（再掲）

逆行列 A^{-1} の近似を A の多項式で生成する

行列のNeumann展開による多項式生成

$\|I - A\| < 1$ の場合

$$A^{-1} = [I - (I - A)]^{-1} = \sum_{j=0}^{\infty} (I - A)^j$$

より，m次までで打ち切った行列を M とすると

$$A^{-1} \approx M = \sum_{j=0}^{m} (I - A)^j$$

- 実際に行列 M は生成せず，反復法内で行列ベクトル積で計算
- 行列ベクトル積の並列化により，同前処理を並列化できる

Cell Broadband Engine

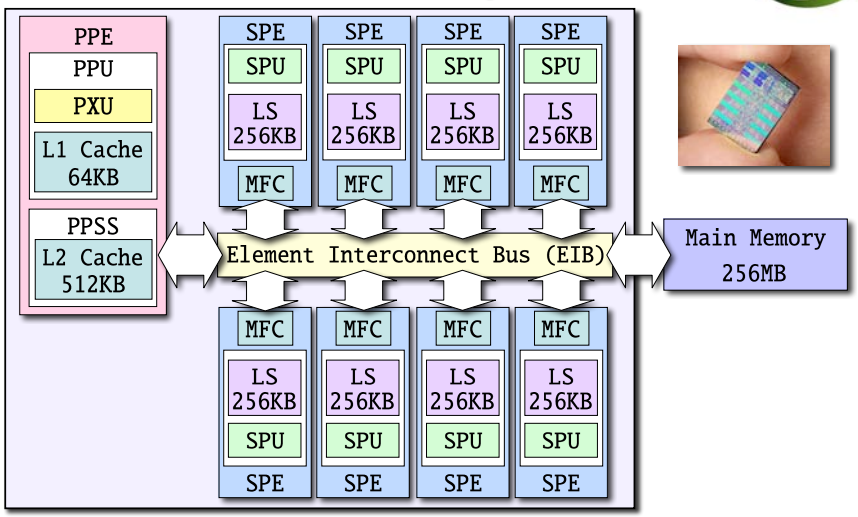


Fig. Cell Broadband Engine の構成図 (PlayStation 3)

CBEにおける疎行列ベクトル積

行列・ベクトル積の高速化

複数の SPE を用いて行列・ベクトル積を行う

⇩ 大きな問題点

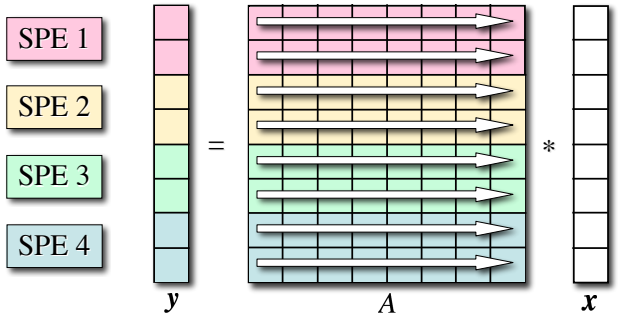
SPE の Local Store は 256KB しかない！

$$y = A * x$$

ベクトル1本すら Local Store に格納できない！

CBEにおける疎行列ベクトル積

SPEを4つ使った例　　行列のブロック化



1) 行列 A のブロックと対応する x のデータがSPE の Local Store に格納できるように分割数を決める.
2) ブロック行列のデータは，CRS形式で保持する.
3) ゼロブロック行列はスキップする.

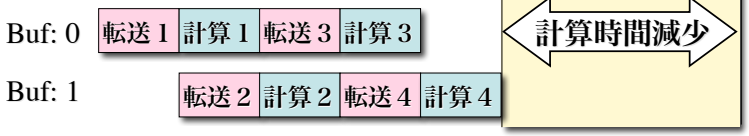
高速化のための手法

高速化のための手法：ダブルバッファリング処理

CBEでは，データの転送と計算を同時に実行できる

シングルバッファの場合

ダブルバッファの場合

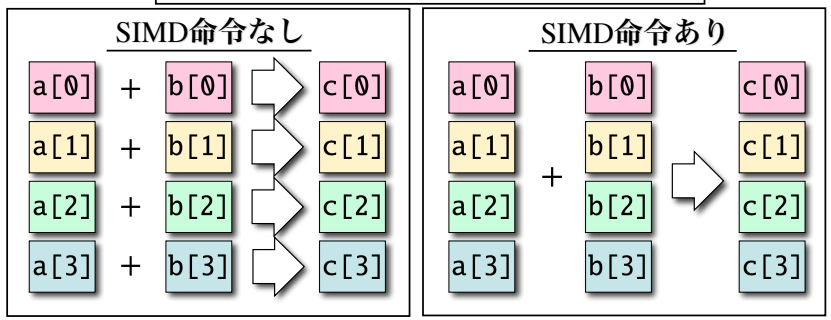


データ転送時間を隠蔽することで高速化が図れる

高速化のための手法

SIMD命令の利用

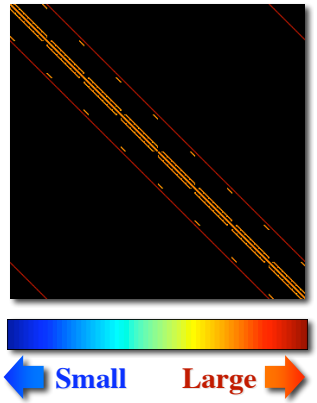
SIMD : **S**ingle **I**nstruction / **M**ultiple **D**ata
1回の命令で複数のデータを処理できる



1回の命令で複数データを処理することで高速化が図れる

数値実験

Quantum Chromodynamics (QCD) で現れる連立一次方程式



連立一次方程式 : $Ax = b$

係数行列 A

$$A = I_n - \kappa \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix}$$

I_n : n次単位行列,
κ : スカラーパラメータ.

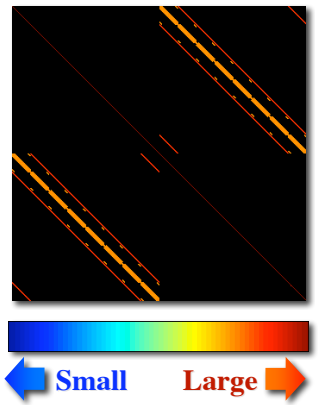
右辺ベクトル b

$$b = [5.4, 5.4, \ldots, 5.4]^{\mathrm{T}}$$

Fig. Aの非零構造図. (`conf5.4-00l8x8-1000`).
サイズ: 49, 152, 非零要素数 : 196, 6080, $\kappa = 0.182$.

数値実験

Red-Black オーダリング



係数行列 A

$$A = I_n - \kappa \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix}$$

RB reordering

並べ替え後の行列

$$A' = I_n - \kappa \begin{bmatrix} O & D'_{01} \\ D'_{10} & O \end{bmatrix}$$

Fig. 並べ替え後の係数行列の非零構造
サイズ : 49, 152, 非零要素数 : 196, 6080, $\kappa = 0.182$.

数値実験

SSOR 前処理の適用

$$A' = I_n - \kappa \begin{bmatrix} O & D'_{01} \\ D'_{10} & O \end{bmatrix}$$

SSOR 前処理

$$\hat{A} = (I_n + L)^{-1} A' (I_n + U)^{-1} = I_n - \kappa^2 \begin{bmatrix} O & O \\ O & D'_{10} D'_{01} \end{bmatrix}$$

前処理後の連立一次方程式

$$\begin{bmatrix} I_{n/2} & O \\ O & I_{n/2} - \kappa^2 D'_{10} D'_{01} \end{bmatrix} \begin{bmatrix} \hat{x}^{(0)} \\ \hat{x}^{(1)} \end{bmatrix} = \begin{bmatrix} \hat{b}^{(0)} \\ \hat{b}^{(1)} \end{bmatrix}$$

サイズが半分の連立一次方程式
$(I_{n/2} - \kappa^2 D'_{10} D'_{01}) \hat{x}^{(1)} = \hat{b}^{(1)}$
を解けばよい.

実験環境

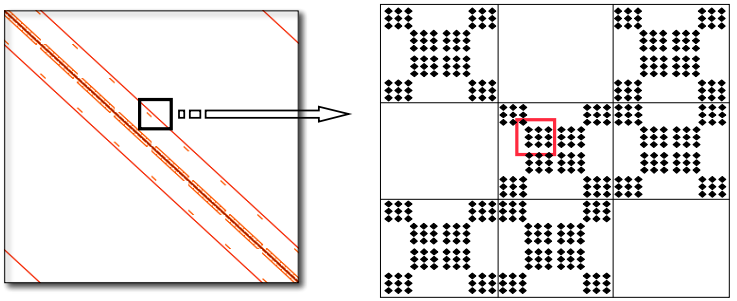
Sony PlayStation 3 (PS3)

CPU	PowerPC Processor Element (PPE) Synergistic Processor Element (SPE) on PS3 3.2GHz
Memory	256MBytes（XDR DRAM）
OS	Fedora Core 5
Compiler	ppu-gfortran, ppu-gcc

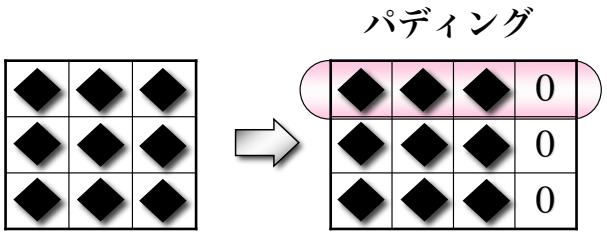
Experimental conditions

Solver	BiCGSTAB
Preconditioner	SSOR + Polynomial（次数5）

非零構造の特徴



SIMDを利用するための工夫



◆：非ゼロ要素

パディングによりSIMD命令が利用可能

数値実験

単精度多項式前処理付きの結果

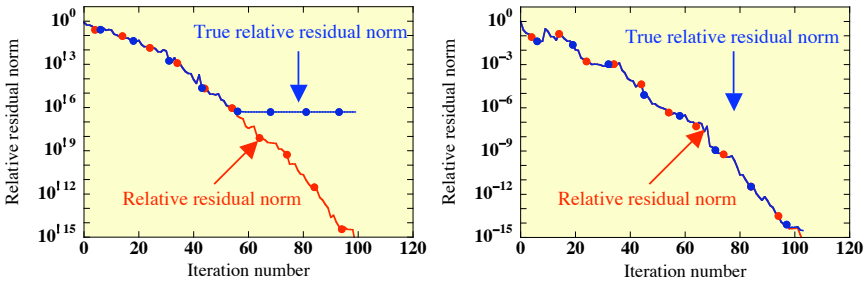
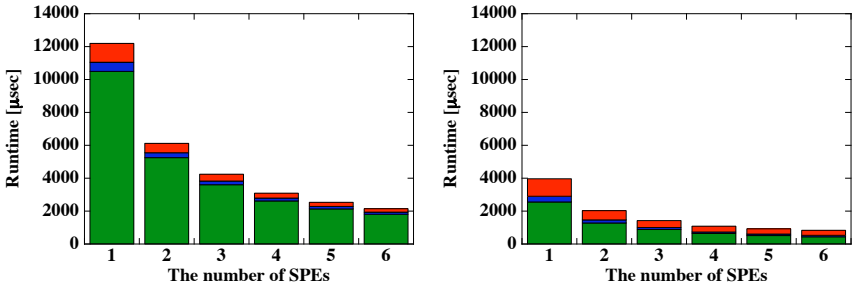


Fig. 単精度多項式前処理の相対残差履歴

●：相対残差ノルム（反復中に計算された残差のノルム）

●：真の相対残差ノルム（近似解から $\|b - Ax_k\|_2/\|b\|_2$ を計算）

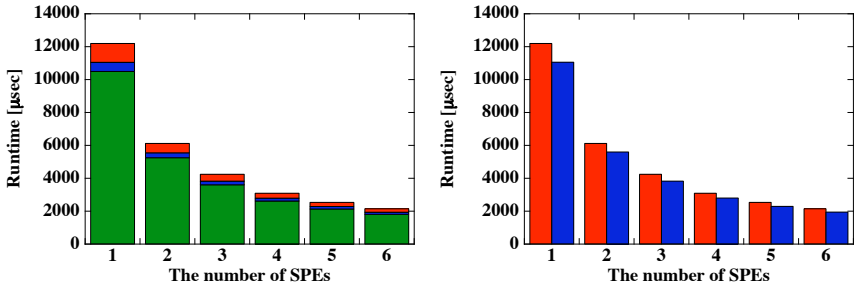
SIMD命令による高速化



(a) SIMD命令なし.　　(b) SIMD命令あり.

Fig. 行列ベクトル積1回の計算時間(シングルバッファリング)
■：DMA転送, ■：その他, ■：計算.

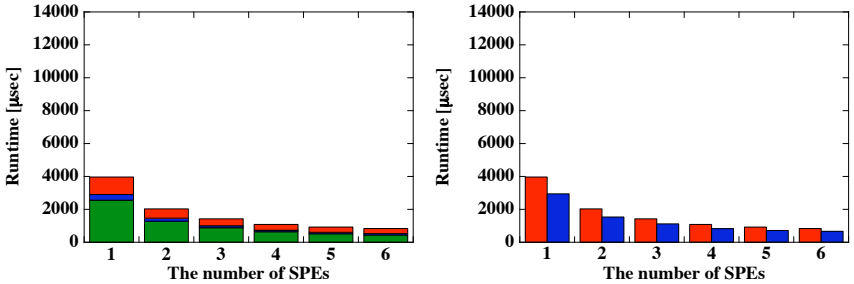
ダブルバッファリングの効果



シングルバッファリングの時間
■：DMA転送, ■：その他,
■：計算.

ダブルバッファリングの効果
■：シングルバッファリング,
■：ダブルバッファリング.

ダブルバッファリング+SIMD



シングルバッファリングの時間
■：DMA転送, ■：その他,
■：計算.

ダブルバッファリングの効果
■：シングルバッファリング,
■：ダブルバッファリング.

まとめ

- 連立一次方程式の解法である Krylov 部分空間反復法を取り上げた.
- 疎行列に対する行列・ベクトル積の実装方法とその並列化について述べた.
- Cell Broadband Engine における疎行列ベクトル積について述べ, その高速化手法を示した.

レポート課題

1. 行列

$$A = \begin{bmatrix} \gamma & -1 & & & \\ -1 & \gamma & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & \gamma & -1 \\ & & & -1 & \gamma \end{bmatrix}$$

をCRS形式で保持するプログラムを作れ．

2. CRS形式の行列ベクトル積を行うプログラムを作れ．
3. 共役勾配法のプログラムを作り，連立一次方程式
   $A\boldsymbol{x} = \boldsymbol{b}$ を解け．但し，$\boldsymbol{b} = A[1, 1, \ldots, 1]^{\mathrm{T}}$ とする．
   行列のパラメータは，$2 < \gamma \leq 4$ に設定せよ．
   $\|\boldsymbol{r}_k\|_2 / \|\boldsymbol{b}\|_2 \leq 1.0 \times 10^{-10}$ を満たしたら反復を停止すること．
4. 複数のパラメータ γ について実験し，反復過程に
   おける相対残差 $\|\boldsymbol{r}_k\|_2 / \|\boldsymbol{b}\|_2$ をグラフにプロットせよ．