Solving linear systems

P. Berger, M. Daydé, R. Guivarch, E. Simon, D. Ruiz (INP-ENSEEIHT- IRIT),
A. Buttari (CNRS, INP-ENSEEIHT-IRIT),
P. Amestoy, J.-Y. L'Excellent (Mumps Technologies),
A. Guermouche (Univ. Bordeaux-LaBRI),
F.-H. Rouet (Ansys, USA),
Bora Uçar (INRIA-CNRS/LIP-ENS Lyon) and
L. Giraud (INRIA)

2025 - ModIA

Ronan Guivarch

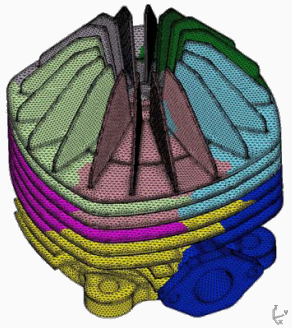
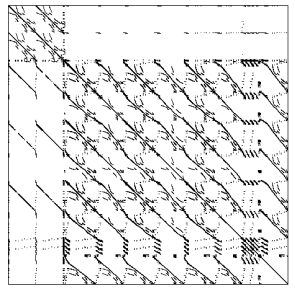
Ronan.Guivarch@toulouse-inp.fr

- Enseignant au département Sciences du Numériques de l'INP-ENSEEIHT : Calcul Scientifique, Probabilités, Méthodes Itératives et Algèbre Linéaire Creuse, Programmation des cartes FPGA, Calcul Parallèle
- Chercheur au laboratoire IRIT, équipe APO (Algorithmes Parallèles et Optimisation) - Mathématiques Numériques / Calcul Parallèle

- 3 CTD autour des solveurs de Krylov (moi)
- 2 CTD d'introduction à l'Algèbre Linéaire Creuse (ALC) (moi)
- 4 CTD autour des méthodes Multigrilles (Carola Kruse (CERFACS))
- 3 CTD autour des méthodes de Décomposition de Domaine (Alena Kopanicakova (INP-ENSEEIHT-IRIT))
- 6 TP sur les 4 sujets précédents

- 1 exam + rendus de TP

Context


Linear System $Ax = b$

At the foundations of many **scientific computing applications** (discretization of PDEs, step of an optimization method, ...).

Large-scale computations...

Up to few **billions (10^9) of unknowns**, applications asking TeraBytes (10^{12}) of memory and Exaflops (10^{18}) of computation.

...require large-scale computers.

Increasingly **large numbers of cores** available, high **heterogeneity in the computation** (CPU, GPU, FPGA, TPU, etc), and high **heterogeneity in data motions** (RAM to cache, out-of-core, node to node transfer, etc).

The sparse case

Matrix sparsity

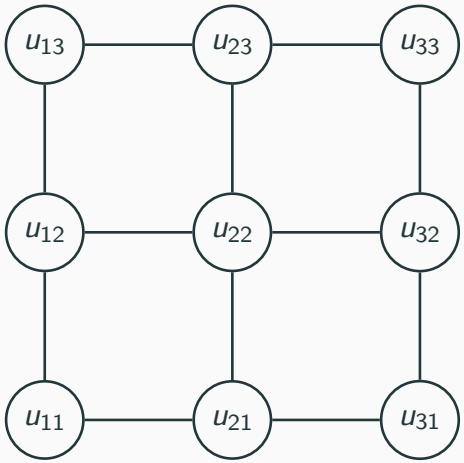
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} \\ \\ \\ \end{bmatrix} \quad A \quad \begin{bmatrix} \\ \\ \\ \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}$$

The sparse case

Matrix sparsity

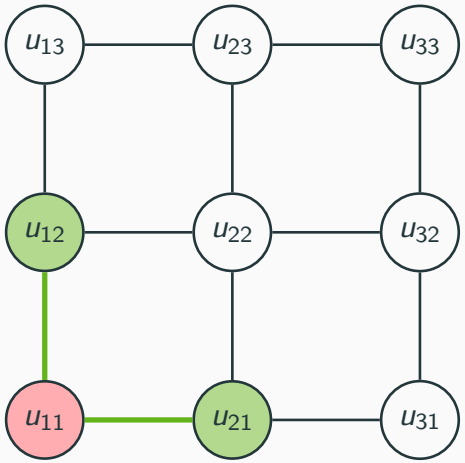
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & & -1 & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{bmatrix} \begin{bmatrix} u_{11} \\ \\ \\ \\ \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} h^2 f_{11} \\ \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

The sparse case

Matrix sparsity

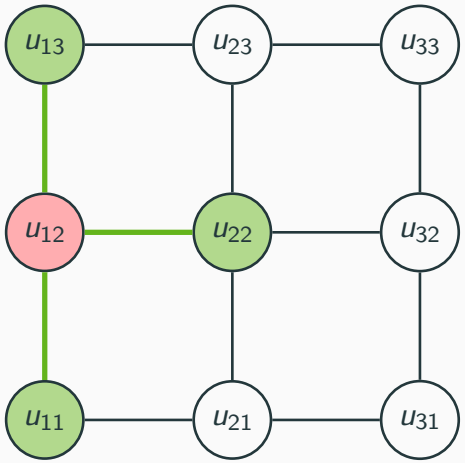
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & & -1 & \\ -1 & 4 & -1 & & -1 \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ \\ \\ \end{bmatrix} = \begin{bmatrix} h^2 f_{11} \\ h^2 f_{12} \\ \\ \\ \end{bmatrix}$$

The sparse case

Matrix sparsity

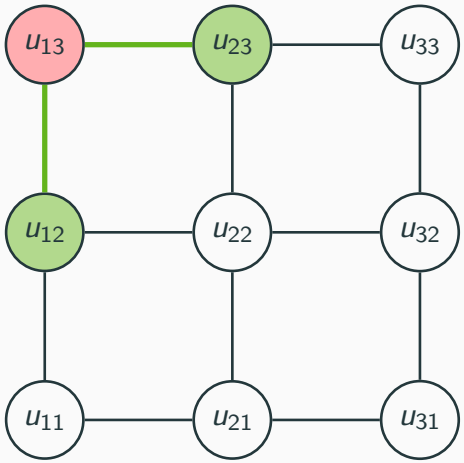
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & & -1 & & \\ -1 & 4 & -1 & & -1 & \\ & \text{-1} & \text{4} & & & \text{-1} \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ \mathbf{u_{13}} \\ \\ \\ \end{bmatrix} = \begin{bmatrix} h^2 f_{11} \\ h^2 f_{12} \\ h^2 f_{13} \\ \\ \\ \end{bmatrix}$$

The sparse case

Matrix sparsity

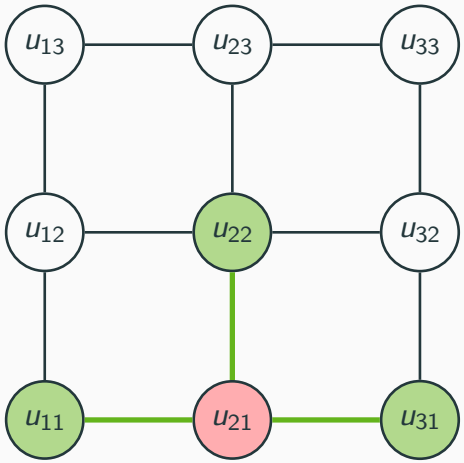
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ \text{-1} & & & 4 & \text{-1} & & \text{-1} & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ \mathbf{u_{21}} \\ \\ \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} h^2 f_{11} \\ h^2 f_{12} \\ h^2 f_{13} \\ h^2 f_{21} \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

The sparse case

Matrix sparsity

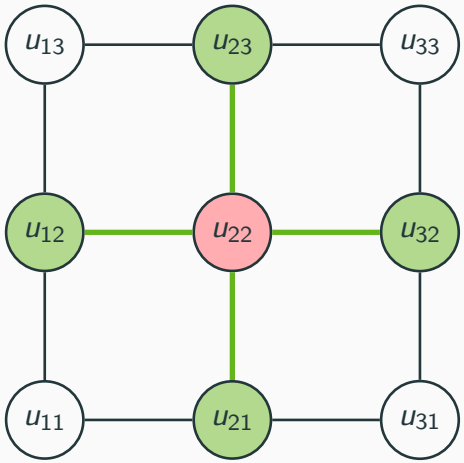
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ u_{22} \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} h^2 f_{11} \\ h^2 f_{12} \\ h^2 f_{13} \\ h^2 f_{21} \\ h^2 f_{22} \\ \\ \\ \\ \end{bmatrix}$$

The sparse case

Matrix sparsity

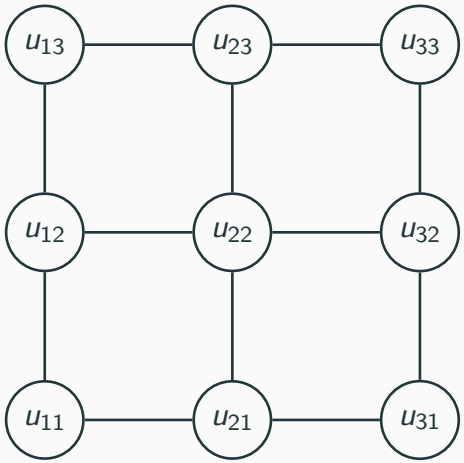
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ u_{22} \\ u_{23} \\ u_{31} \\ u_{32} \\ u_{33} \end{bmatrix} = \begin{bmatrix} h^2 f_{11} \\ h^2 f_{12} \\ h^2 f_{13} \\ h^2 f_{21} \\ h^2 f_{22} \\ h^2 f_{23} \\ h^2 f_{31} \\ h^2 f_{32} \\ h^2 f_{33} \end{bmatrix}$$

Solvers

What are the ways to solve a $Ax = b \in \mathbb{R}^n$ on computers ?

Iterative solvers

Compute a sequence of x_k converging towards x.

Examples: Gauss-Seidel, SOR, Steepest Descent, Conjugate Gradient, Krylov subspace methods, etc.

- Low computational cost and memory consumption if the convergence is quick (about $\mathcal{O}(n^2)$ (dense case)) operations per iteration), ...
- BUT convergence depends on the matrix properties.

Direct solvers

Compute a factorization of A followed by forward and backward substitutions.

Examples: LDL^T, LU, QR, etc.

- High computational cost and memory consumption...
- BUT they are robust and easy to use.

Solvers

What are the ways to solve a sparse $Ax = b \in \mathbb{R}^n$ on computers ?

Iterative solvers

Compute a sequence of x_k converging towards x.

Examples: Gauss-Seidel, Steepest Descent, Conjugate Gradient, SOR, Krylov subspace methods, etc.

- Low computational cost and memory consumption if the convergence is quick (about $\mathcal{O}(nnz(A))$ (sparse case)) operations per iteration), ...
- BUT convergence depends on the matrix properties.

Direct solvers: dense/sparse matrix $\Rightarrow$ dense/sparse solver

Compute a factorization of A followed by forward and backward substitutions.

Examples: LDL^T, LU, QR, etc.

- High computational cost and memory consumption...
- BUT they are robust and easy to use.

Outline

Quality of a solution: conditioning and errors

Iterative Methods: Krylov Methods

Iterative Methods: Symmetric Krylov Solvers

Iterative Method: Conjugate Gradient

Iterative Methods: Algebraic preconditioning techniques

Quality of a solution: conditioning and errors

Quality of a solution: conditioning and errors

Conditionnement

Conditionnement - Sensibilité du problème

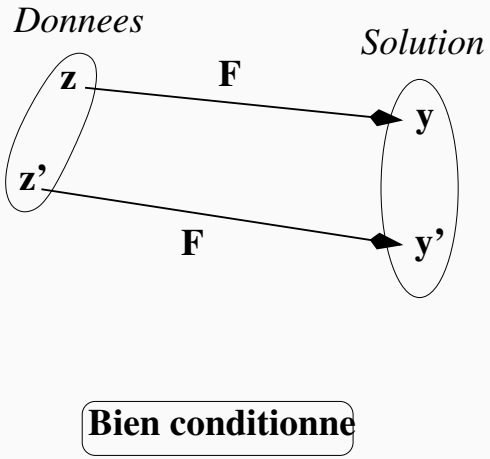
- Soit A :
$$\begin{bmatrix} .780 & .563 \\ .913 & .659 \end{bmatrix}$$ matrice presque singulière
- Soit A' :
$$\begin{bmatrix} .780 & .563001095 \\ .913 & .659 \end{bmatrix}$$ matrice singulière

  $\rightarrow$ une perturbation des entrées de la matrice d'ordre $O(10^{-6})$ rend le problème insoluble.
- Autre situation si A est presque singulière : une petite perturbation sur A et/ou b (mesure, numérique, ...) $\rightarrow$ grandes perturbations sur la solution

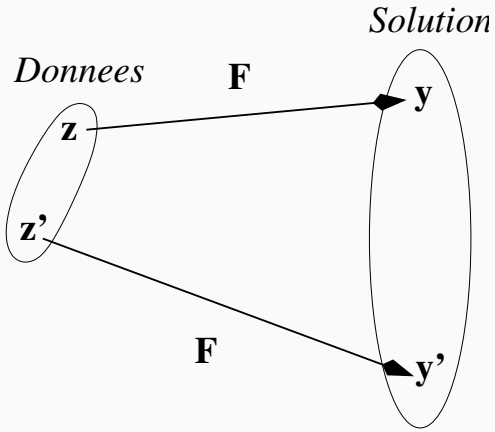
Ceci est propre au problème,
et est indépendant de l'algorithme de résolution utilisé

Conditionnement d'un problème

Le conditionnement peut être mesuré par le changement relatif solution / changement relatif données $\frac{\| \frac{F(z')-F(z)}{F(z)} \|}{\| \frac{(z'-z)}{z} \|}$



Bien conditionne



Mal conditionne

Définition (Problème bien conditionné)

Un problème est bien conditionné si une faible variation des données entraine une faible variation de la solution (*c.à.d.* $\|z - z'\|$ *petit* $\Rightarrow \|F(z) - F(z')\|$ *petit*)

Conditionnement

$$Ax = b$$

Les coefficients de A et b sont des variables calculées, numériquement incorrectes.

$$\Downarrow$$

Influence de ces erreurs sur le résultat du système :

$$(A + \Delta A) \cdot (x + \Delta x) = b + \Delta b$$

$$\Downarrow$$

Recherche d'un majorant de $\|(x + \Delta x) - x\| / \|x\| = \|\Delta x\| / \|x\|$

Conditionnement

Avec une norme induite, résultat généralement sous la forme :
$K(A) = \|A\| \cdot \left\|A^{-1}\right\|$ nombre de conditionnement de A

$$\|\Delta x\|/\|x\| \leqslant K(A)\,(\|\Delta A\|/\|A\| + \|\Delta b\|/\|b\|)$$

- Plus $K(A)$ est faible, meilleur est le conditionnement du système
- Valeur minimale de $K(A)$? $K(A) = \|A\| \left\|A^{-1}\right\| \geqslant \left\|A \cdot A^{-1}\right\| = 1$
- Selon la norme, $K(A)$ prend des valeurs différentes
- Si toutes les valeurs propres de A sont réelles (par ex., A symétrique) :

$$|\lambda_1| \geqslant \ldots \geqslant |\lambda_n| \qquad K(A) = \|A\|_2 \|A^{-1}\|_2 = |\lambda_1|/|\lambda_n|$$

Conditionnement

- Si A est orthogonale, $K(A) = 1$
- $K(A)$ élevé ne prouve que l'existence d'un **risque** de mauvais comportement lors de la résolution
- **Comment** évaluer $K(A)$? Ne jamais évaluer A^{-1} et utiliser de préférence des propriétés liées aux valeurs propres
- **Quand** calculer $K(A)$? Si l'on envisage d'utiliser une méthode de résolution connue pour être sensible au conditionnement

Et si $K(A) \gg 1$?? Changer de méthode ou préconditionner :

$$A \cdot x = b \rightarrow C \cdot A \cdot x = C \cdot b \text{ avec } K(C \cdot A) < K(A)$$

- Choix idéal de C : A^{-1} ????
- Choix réaliste de C : une approximation de A^{-1}

Quality of a solution: conditioning and errors

Forward and Backward Errors

Notion de bonne solution

- Considérons le système linéaire :

$$\begin{bmatrix} .780 & .563 \\ .913 & .659 \end{bmatrix} x = \begin{bmatrix} .217 \\ .254 \end{bmatrix}$$

- Supposons que deux algorithmes différents donnent les deux solutions suivantes :

$$x_1 = \begin{bmatrix} -20.568 \\ 28.881 \end{bmatrix} \text{ and } x_2 = \begin{bmatrix} 0.999 \\ -1.00 \end{bmatrix}$$

De x_1 et x_2, quelle est la meilleure solution ?

Notion de bonne solution

- Résidus :

$$b - Ax_1 = \begin{bmatrix} -37 \\ -5 \end{bmatrix} \times 10^{-5} \text{ and } b - Ax_2 = \begin{bmatrix} -78 \\ -91 \end{bmatrix} \times 10^{-4}$$

  - x_1 est la meilleure solution car son résidu est le plus petit
- Solution exacte :

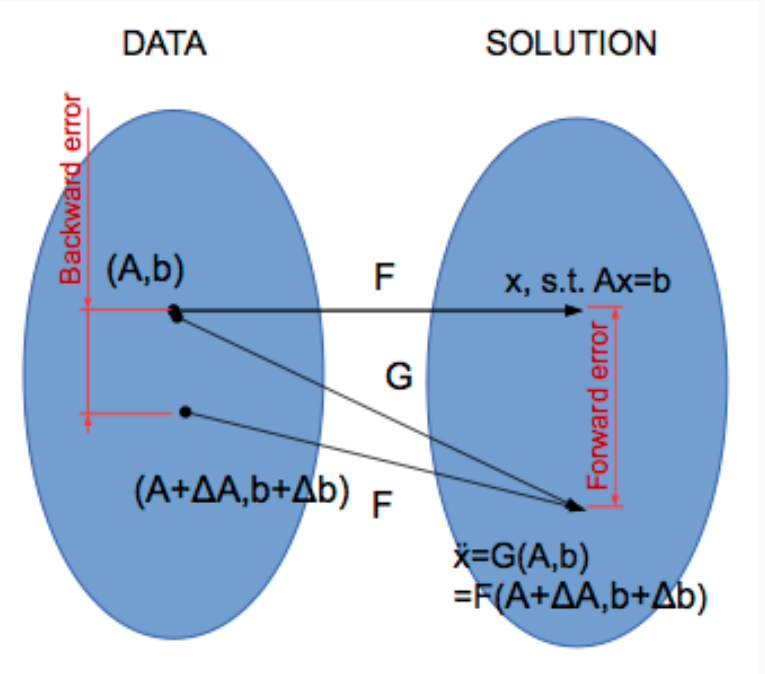
$$x^* = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

  - en fait, x_2 est plus proche de la solution exacte

La notion de bonne solution est un concept ambigu

Erreurs directe et inverse

- F fait correspondre (précision exacte) les données (A, b) à la solution x de $Ax = b$
- G est un algorithme implémentant F et travaillant en précision finie
- $\tilde{x}$ est telle que $\tilde{x} = G(A, b) = F(A + \Delta A, b + \Delta b)$



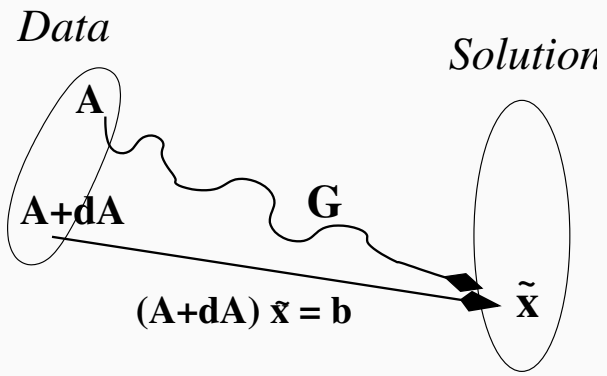
solution calculée (algorithme G) = solution exacte d'un problème perturbé

- Amplitude des perturbations (erreur inverse) associée à l'erreur dans la solution (erreur directe)
- La solution calculée sera satisfaisante si elle est la solution exacte d'un problème perturbé avec des perturbations acceptables pour l'utilisateur (ordre des erreurs d'arrondi, incertitude sur les données, ...)

Erreur inverse d'un algorithme résolvant $Ax = b$

soit $\tilde{x}$ la solution approchée calculée par l'algorithme G.

Objectif : trouver les plus petites perturbations ΔA et Δb telles que $(A + \Delta A)\tilde{x} = b + \Delta b$



Théorème (Erreur Inverse (Rigal and Gaches, 1967))

Si nous définissons l'erreur inverse (backward error)

$$\eta^N_{A,b}(\tilde{x}) = \min\{\varepsilon > 0 \text{ tel que } \|\Delta A\|_2 \leq \varepsilon\|A\|_2,\ \|\Delta b\|_2 \leq \varepsilon\|b\|_2, \text{ et } (A + \Delta A)\tilde{x} = b + \Delta b\}$$

Alors on peut prouver que :

$$\eta^N_{A,b}(\tilde{x}) = \frac{\|b - A\tilde{x}\|_2}{\|A\|_2\|\tilde{x}\|_2 + \|b\|_2}$$

Autres Définitions de l'erreur inverse

Erreur Inverse ne faisant intervenir que b

$$\begin{aligned}
\eta_b^N(\tilde{x}) &= \min\{\varepsilon > 0 \text{ tel que } \|\Delta b\|_2 \leq \varepsilon \|b\|_2, \\
&\qquad \text{et } A\tilde{x} = b + \Delta b\} \\
&= \frac{\|b - A\tilde{x}\|_2}{\|b\|_2} \\
&= \frac{\|r\|_2}{\|b\|_2} \quad (r \text{ est le résidu.})
\end{aligned}$$

- $\eta_{A,b}^N$ et η_b^N sont des erreurs inverses "***Normwise***" et sont utilisées dans le critère d'arrêt des méthodes itératives

Autres Définitions de l'erreur inverse

Erreur Inverse "*Componentwise*"

en considérant les perturbations au niveau des composantes, nous définissons :

$$\begin{aligned} \eta^{C}_{A,b}(\tilde{x}) &= \min\{\varepsilon > 0 \text{ tel que } |\Delta A| \leq \varepsilon |A|,\ |\Delta b| \leq \varepsilon |b|, \\ &\qquad \text{et } (A + \Delta A)\tilde{x} = b + \Delta b\} \\ &= \max_{i=1,\ldots,n} \frac{|b - A\tilde{x}|_i}{(|A||\tilde{x}| + |b|)_i} \end{aligned}$$

avec la convention $\frac{0}{0} = 0$.

Cette erreur inverse convient bien aux problèmes présentant des différences de magnitude entre les composantes.

Quality of a solution: conditioning and errors

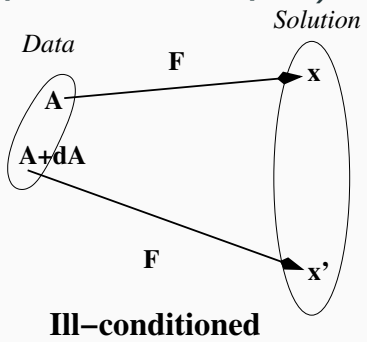
Synthèse (nombre de conditionnement et erreur inverse)

Synthèse

- Conditionnement (de manière générale Δb doit être pris en compte) :

Approx. premier ordre $\quad \frac{\|\Delta x\|}{\|x\|} \leq \kappa(A) \frac{\|\Delta A\|}{\|A\|}$

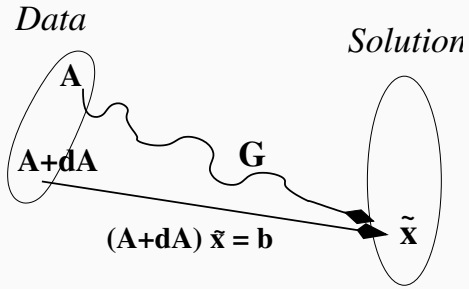
avec $\quad \kappa(A) = \|A^{-1}\|\|A\|$



- Erreur inverse :

$$\eta^N_{A,b}(\tilde{x}) = \frac{\|A\tilde{x} - b\|}{\|A\|\|\tilde{x}\| + \|b\|}$$

(mesure la qualité d'un algorithme)



$\rightarrow$ à comparer à l'ε-machine ou l'incertitude sur les données en entrée

- Prédiction de l'erreur directe :
  Erreur Directe $\leq$ Nombre de Cond. $\times$ Erreur Inverse

Iterative Methods: Krylov Methods

Some references

- Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2nd edition
- C.T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM
- Luc Giraud and Serge Gratton, "Introduction to Krylov subspace methods for the solution of linear systems" (pdf on Moodle)

Iterative Methods: Krylov Methods

Some background

Krylov methods: some background

Aleksei Nikolaevich Krylov

1863-1945: Russia, Maritime Engineer

His research spans a wide range of topics, including shipbuilding, magnetism, artillery, mathematics, astronomy, and geodesy. In 1904 he built the first machine in Russia for integrating ODEs.

In 1931 he published a paper on what is now called the "Krylov subspace".

(Most slides of this part are from Luc Giraud / INRIA)

Krylov methods: some background

Definition

Let $A \in \mathbb{R}^{n \times n}$ and $r \in \mathbb{R}^n$; the space denoted by $\mathcal{K}_m(A, r)$ (with $m \leq n$) and defined by

$$\mathcal{K}_m(A, r) = Span\{r, Ar, ..., A^{m-1}r\}$$

is referred to as the Krylov space of dimension m associated with A and r.

Assumption

Let us assume $x_0 = 0$

No loss of generality,

because the situation $x_0 \neq 0$ can be transformed with a simple shift to the system $Ay = b - Ax_0 = \bar{b}$,

for which obviously $y_0 = 0$.

Why using this search space?

The minimal polynomial $q(t)$ of A is the unique monic polynomial of minimal degree such that $q(A) = 0$.

It is constructed from the eigenvalues of A as follows:

If the distinct eigenvalues of A are $\lambda_1, ..., \lambda_\ell$ and if λ_j has index m_j (the size of the largest Jordan block associated with λ_j), then the sum of all indices is

$$m = \sum_{j=1}^{\ell} m_j, \text{ and } q(t) = \prod_{j=1}^{\ell} (t - \lambda_j)^{m_j}. \tag{1}$$

When A is diagonalizable, m is the number of distinct eigenvalues of A. When A is a Jordan block of size n, then $m = n$.

Why using this search space?

If we write

$$q(t) = \prod_{j=1}^{\ell} (t - \lambda_j)^{m_j} = \sum_{j=0}^{m} \alpha_j t^j,$$

then the constant term is $\alpha_0 = \prod_{j=1}^{\ell} (-\lambda_j)^{m_j}$.

Therefore $\alpha_0 \neq 0$ iff A is nonsingular.

Furthermore, from

$$0 = q(A) = \alpha_0 I + \alpha_1 A + ... + \alpha_m A^m, \tag{2}$$

it follows that

$$A^{-1} = -\frac{1}{\alpha_0} \sum_{j=0}^{m-1} \alpha_{j+1} A^j.$$

This description of A^{-1} portrays $x = A^{-1}b$ immediately as a member of the Krylov space of dimension m associated with A and b denoted by $\mathcal{K}_m(A, b) = Span\{b, Ab, ..., A^{m-1}b\}$.

Taxonomy of the Krylov subspace approaches

The Krylov methods for identifying $x_m \in \mathcal{K}_m(A, b)$ can be distinguished in four classes:

- The Galerkin approach (or Ritz-Galerkin approach) (FOM, CG,...):
  construct x_m such that the residual is orthogonal to the current subspace: $b - Ax_m \perp \mathcal{K}_m(A, b)$.
- The minimum norm residual approach (GMRES,...):
  construct $x_m \in \mathcal{K}_m(A, b)$ such that $||b - Ax_m||_2$ is minimal that means that the residual is orthogonal to the subspace $A\mathcal{K}_m(A, b)$ (ie $b - Ax_m \perp A\mathcal{K}_m(A, b)$).
- The Petrov-Galerkin approach:
  construct x_m such that $b - Ax_m$ is orthogonal to some other m-dimensional subspace.
- The minimum norm error approach:
  construct $x_m \in A^T\mathcal{K}_m(A, b)$ such that $||b - Ax_m||_2$ is minimal.

Constructing a basis of $\mathcal{K}_m(A, b)$

- Obvious choice b, Ab, ..., $A^{m-1}b$
  - not very attractive from the numerical point of view because vectors $A^j b$ become more and more colinear to the eigenvector associated to the largest eigenvalue.
  - In finite arithmetic, leads to a loss of rank: suppose A is diagonalizable $A = VDV^{-1}$, then $A^k b = VD^k(V^{-1}b)$.
  - A better choice is the Arnoldi procedure.

Iterative Methods: Krylov Methods

Arnoldi procedure

Arnoldi

Walter Edwin Arnoldi
1917-1995: USA.
His main research subjects covered vibration of propellers, engines and aircraft, high speed digital computers, aerodynamics and acoustics of aircraft propellers, lift support in space vehicles and structural materials.

"The principle of minimized iterations in the solution of the eigenvalue problem" in Quart. of Appl. Math., Vol.9 in 1951.

The Arnoldi procedure

$=>$ builds an orthonormal basis of $\mathcal{K}_m(A, b)$

1 $v_1 = b/\|b\|$
2 **for** $j = 1, 2, \ldots, m$ **do**
3 | **for** $i = 1, \ldots, j$ **do**
4 | | Compute $h_{i,j} = v_i^T A v_j$
5 | **end**
6 | $w_j = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
7 | Compute $h_{j+1,j} = \|w_j\|$
8 | exit if $(h_{j+1,j} = 0)$
9 | Compute $v_{j+1} = w_j / h_{j+1,j}$
10 **end**

Algorithm 1: Arnoldi procedure

Steps $\{3,4,5\}$ and 6 : Classical Gram-Schmidt (CGS)

The Arnoldi procedure properties

Proposition

If the Arnoldi procedure does not stop before the m^{th} step, the vectors $v_1, ..., v_m$ form an orthonormal basis of the Krylov subspace $\mathcal{K}_m(A, b)$.

Proof.

The vectors are orthogonal by construction and have a norm equal to 1. They span $\mathcal{K}_{m-1}(A, b)$ follows from the fact that each vector v_j is of the form $q_{j-1}(A)v_1$, where q_{j-1} is a polynomial of degree $j-1$.

This can be shown by induction.

For $j = 1$ it is true as $v_1 = q_0(A)v_1$ with $q_0 = 1$.

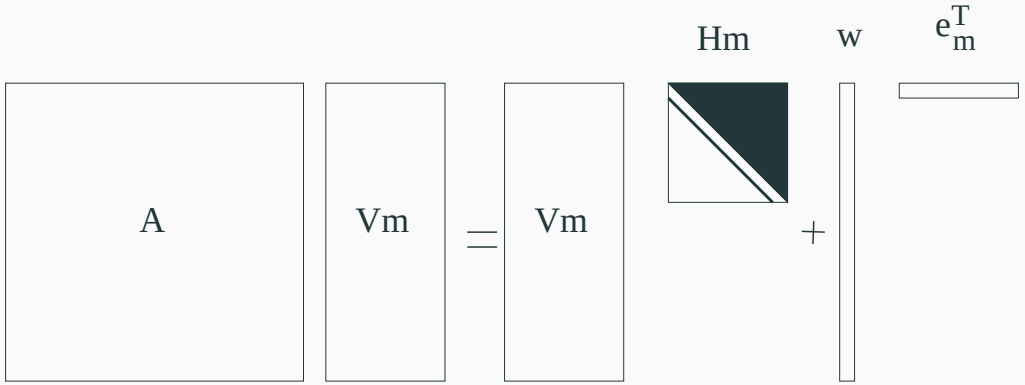
Assume that it is true for all j and consider v_{j+1}.

We have:

$$h_{j+1,j}v_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j}v_i = Aq_{j-1}(A)v_1 + \sum_{i=1}^{j} h_{i,j}q_{i-1}(A)v_1.$$

So v_{j+1} can be expressed as $q_j(A)v_1$ where q_j is of degree j. □

The Arnoldi procedure properties



Proposition

Denote

- V_m *the* $n \times m$ *matrix with column vector* $v_1, ..., v_m$
- $\overline{H}_m$ *the* $(m+1) \times m$ *Hessenberg matrix whose nonzero entries are* h_{ij}
- H_m *the square matrix obtained from* $\overline{H}_m$ *by deleting its last row.*

Then the following relations hold:

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T \quad (3)$$

$$AV_m = V_{m+1} \overline{H}_m \quad (4)$$

$$V_m^T A V_m = H_m \quad (5)$$

Proof.

Equality (3) can be shown for each column. Using Matlab notations:

For $j \neq m$

$$\begin{aligned} & AV_m(:,j) && = V_m H(:,j) + w_m \times 0 \\ \Leftrightarrow \quad & Av_j && = \sum_{i=1}^{j+1} v_i h_{i,j} \\ \Leftrightarrow \quad & Av_j && = \sum_{i=1}^{j} h_{i,j} v_i + h_{j+1,j} v_{j+1} \end{aligned}$$

which corresponds to steps 3, 4 and 5 of the algorithm.

For $j = m$

$$\begin{aligned} & AV_m(:,m) && = V_m H(:,m) + w_m \times 1 \\ \Leftrightarrow \quad & Av_m && = \sum_{i=1}^{m} v_i h_{i,m} + w_m \\ \Leftrightarrow \quad & w_m && = Av_m - \sum_{i=1}^{m} h_{i,m} v_i \end{aligned}$$

which corresponds to step 4 of the algorithm. □

Proof.

To show equality (4), we first notice the structure of $\overline{H}_m$

$$\begin{pmatrix} H_m \\ h_{m+1,m} e_m^T \end{pmatrix}$$

Equality (3) writes

$$\begin{aligned} AV_m &= V_m H_m + h_{m+1,m} v_{m+1} e_m^T \\ &= V_m H_m + v_{m+1}(h_{m+1,m} e_m^T) \\ &= \begin{pmatrix} V_m v_{m+1} \end{pmatrix} \begin{pmatrix} H_m \\ h_{m+1,m} e_m^T \end{pmatrix} \\ &= V_{m+1} \overline{H}_m \end{aligned}$$

□

Proof.

Equality (5):

$$V_m^T A V_m = H_m$$

V_m is an orthonormal matrix: $V_m^T V_m = I_m$ (identity in $\mathbb{R}^m$).

By construction v_{m+1} is orthogonal to all the previous vectors: $V_m^T v_{m+1} = 0$.

Multiplying Equation (3) by V_m^T on the left gives

$$V_m^T A V_m = \underbrace{V_m^T V_m}_{I_m} H_m + h_{m+1,m} \underbrace{V_m^T v_{m+1}}_{O_m} e_m^T$$

□

When $x_0 \neq 0$

- we assumed $x_0 = 0$ and we showed that we seek x_m in the affine subspace $\mathcal{K}_m(A, b)$.
- if $x_0 \neq 0$, then we can show that we seek x_m in the affine subspace $x_0 + \mathcal{K}_m(A, r_0)$ where $r_0 = b - Ax_0$.

Using Arnoldi procedure to solve a linear system

From an initial guess x_0, we search $x_m \in x_0 + \mathcal{K}_m = x_0 + \mathcal{K}_m(A, r_0)$.

How to choose x_m?

- Galerkin approach: $b - Ax_m \perp \mathcal{K}_m$

  $=>$ FOM Solver (Full Orthogonalization Method)

- Minimum Norm Residual approach: $\|b - Ax_m\|_2$ minimal

  $=>$ GMRES Solver (Generalised Minimal RESisual)

Iterative Methods: Krylov Methods

Galerkin Approach – FOM

Galerkin Approach

- $x_m \in x_0 + \mathcal{K}_m$ means $\exists y_m \in \mathbb{R}^m$ such that $x_m = x_0 + V_m y_m$

Exercise

How y_m can be computed to verify $b - Ax_m \perp \mathcal{K}_m$?

FOM Algorithm

1 Set the initial guess x_0
2 $r_0 = b - Ax_0$; $\beta = \|r_0\|$; $v_1 = r_0/\beta$;
3 **for** $j = 1, 2, \ldots, m$ **do**
4 $\quad w_j = Av_j$;
5 $\quad$ **for** $i = 1, \ldots, j$ **do**
6 $\quad\quad h_{i,j} = v_i^T w_j$;
7 $\quad\quad w_j = w_j - h_{i,j} v_i$;
8 $\quad$ **end**
9 $\quad h_{j+1,j} = \|w_j\|$;
10 $\quad$ **if** $h_{j+1,j} = 0$ **then**
11 $\quad\quad$ m = j;
12 $\quad\quad$ goto 16
13 $\quad$ **end**
14 $\quad v_{j+1} = w_j / h_{j+1,j}$
15 **end**
16 $y_m = H_m^{-1}(\beta e_1)$
17 $x_m = x_0 + V_m y_m$

Algorithm 2: FOM - MGS variant - Y.Saad - Algo 6.4

steps 4 to 8 : Modified Gram-Schmidt (MGS)

Some remarks on this algorithm

- with this algorithm presented this way, we compute $x_m \in x_0 + \mathcal{K}_m$ for a given m.
- there is no garanty that x_m is close to the exact solution ($\eta^N_{A,b}(\tilde{x_m}) < \varepsilon$? or $\eta^N_b(\tilde{x_m}) < \varepsilon$?).
- we have to re-formulate the algorithm in order to stop when the wanted precision is achieved (or when we have reach a maximum number of iterations).

Exercise

re-write the algorithm

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration?
(computational cost: 2 matrix$\times$vector products)

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration? (computational cost: 2 matrix×vector products)

- a priori, yes: $\|r_j\|$ is present in the stopping criteria

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration? (computational cost: 2 matrix×vector products)

- a priori, yes: $\|r_j\|$ is present in the stopping criteria
- we show that

$$b - Ax_j = -h_{j+1,j} e_j^T y_j v_{j+1}$$

and

$$\|b - Ax_j\| = h_{j+1,j} |e_j^T y_j|$$

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration? (computational cost: 2 matrix×vector products)

- a priori, yes: $\|r_j\|$ is present in the stopping criteria
- we show that

$$b - Ax_j = -h_{j+1,j} e_j^T y_j v_{j+1}$$

and

$$\|b - Ax_j\| = h_{j+1,j} |e_j^T y_j|$$

Exercise

Prove these two relations

Improvements of FOM algorithm

Question: is it necessary to compute x_j and r_j at each iteration? (computational cost: 2 matrix×vector products)

- a priori, yes: $\|r_j\|$ is present in the stopping criteria
- we show that

$$b - Ax_j = -h_{j+1,j} e_j^T y_j v_{j+1}$$

and

$$\|b - Ax_j\| = h_{j+1,j} |e_j^T y_j|$$

Exercise

Prove these two relations

$\Rightarrow$

we don't have to compute x_j and r_j at each iteration. We will compute these values once we have converged (or we estimate that we have converged).

Iterative Methods: Krylov Methods

Minimum Norm Residual approach – GMRES

Minimum Norm Residual approach

- $x_m \in x_0 + \mathcal{K}_m$ means $\exists y_m \in \mathbb{R}^m$ such that $x_m = x_0 + V_m y_m$

- **Question**:
  How y_m can be computed such that $\|b - Ax_m\|_2$ is minimal?

Central Equality

Exercise

Prove that $\|b - Ax_m\|_2 = \|\beta e_1 - \overline{H}_m y_m\|_2$

GMRES method

The GMRES iterate is the vector of $x_0 + \mathcal{K}_m$ such that

$$\begin{aligned} x_m &= x_0 + V_m y_m \\ y_m &= \arg\min_{y \in \mathbb{R}^m} \|\beta e_1 - \bar{H}_m y\| \end{aligned}$$

GMRES algorithm

1 Set the initial guess x_0
2 $r_0 = b - Ax_0$; $\beta = \|r_0\|$; $v_1 = r_0/\beta$;
3 **for** $j = 1, 2, \ldots, m$ **do**
4 $\quad w_j = Av_j$;
5 $\quad$ **for** $i = 1, \ldots, j$ **do**
6 $\quad\quad h_{i,j} = v_i^T w_j$;
7 $\quad\quad w_j = w_j - h_{i,j} v_i$;
8 $\quad$ **end**
9 $\quad h_{j+1,j} = \|w_j\|$;
10 $\quad$ **if** $h_{j+1,j} = 0$ **then**
11 $\quad\quad$ m = j;
12 $\quad\quad$ goto 16
13 $\quad$ **end**
14 $\quad v_{j+1} = w_j / h_{j+1,j}$
15 **end**
16 Solve the least-squares problem $y_m = \arg\min \|\beta e_1 - \bar{H}_m y\|$
17 $x_m = x_0 + V_m y_m$

Algorithm 3: GMRES - MGS variant - Y.Saad - Algo 6.9

Solution of the linear least-square problem

A stable solution technique to solve $\|\beta e_1 - \overline{H}_m y\|_2$ is to use Givens rotations to transform the Hessenberg matrix into upper triangular form.

$$\mathcal{Q}_i = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & c_i & s_i & & & \\ & & & -s_i & c_i & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \leftarrow \text{row i} \\ \leftarrow \text{row i} + 1 \\ \\ \\ \\ \end{matrix}$$

with $c_i^2 + s_i^2 = 1$.
Computing $B = \mathcal{Q}_i A$ copies all the rows but the i^{th} and $(i+1)^{th}$ from A in B and replaces those two rows by linear combinations of them such that $B_{i+1,i} = 0$ if $s_i = \frac{a_{i+1,i}}{\sqrt{a_{i,i}^2 + a_{i+1,i}^2}}$ and $c_i = \frac{a_{i,i}}{\sqrt{a_{i,i}^2 + a_{i+1,i}^2}}$.

Application of the first Givens rotation to $\overline{H}_5$

$$\text{Consider } \overline{\mathrm{H}}_5 = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \\ & & & & h_{65} \end{pmatrix} \text{ and } \beta \mathrm{e}_1 = \begin{pmatrix} \beta \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Multiplying by the left by

$$\mathcal{Q}_1 = \begin{pmatrix} c_1 & s_1 & & & & \\ -s_1 & c_1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix} \text{ with } \begin{cases} s_1 = \frac{h_{2,1}}{\sqrt{h_{1,1}^2 + h_{2,1}^2}} \\ c_1 = \frac{h_{1,1}}{\sqrt{h_{1,1}^2 + h_{2,1}^2}} \end{cases}.$$

Clearly $\mathcal{Q}_1^T \mathcal{Q}_1 = I$.

Result after the first rotation

$$\overline{H}_5^{(1)} = \begin{pmatrix} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & h_{14}^{(1)} & h_{15}^{(1)} \\ 0 & h_{22}^{(1)} & h_{23}^{(1)} & h_{24}^{(1)} & h_{25}^{(1)} \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \\ & & & & h_{65} \end{pmatrix} \text{ and } \bar{\mathrm{g}}_1^{(1)} = \begin{pmatrix} c_1\beta \\ -s_1\beta \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Result after 5 rotations

This elimination process is continued until the 5^{th} rotation is applied which leads to

$$\overline{H}_5^{(5)} = \begin{pmatrix} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & h_{14}^{(1)} & h_{15}^{(1)} \\ 0 & h_{22}^{(2)} & h_{23}^{(2)} & h_{24}^{(2)} & h_{25}^{(2)} \\ & 0 & h_{33}^{(3)} & h_{34}^{(3)} & h_{35}^{(3)} \\ & & 0 & h_{44}^{(4)} & h_{45}^{(4)} \\ & & & 0 & h_{55}^{(5)} \\ & & & & 0 \end{pmatrix} \text{ and } \overline{\mathrm{g}}^{(5)} = \begin{pmatrix} \gamma_1^{(1)} \\ \gamma_2^{(2)} \\ \gamma_3^{(3)} \\ \gamma_4^{(4)} \\ \gamma_5^{(5)} \\ \gamma_6^{(5)} \end{pmatrix}$$

QR Factorisation of $\bar{H}_m$

Generally, the rotation $\mathcal{Q}_i$ is defined by

$$s_i = \frac{h_{i+1,i}}{\sqrt{\left(h_{i,i}^{(i-1)}\right)^2 + h_{i+1,1}^2}} \text{ and } \mathrm{c_i} = \frac{\mathrm{h_{i,i}^{(i-1)}}}{\sqrt{\left(\mathrm{h_{i,i}^{(i-1)}}\right)^2 + \mathrm{h_{i+1,1}^2}}}. \tag{6}$$

Define, the unitary matrix

$$Q_m = \mathcal{Q}_m \dots \mathcal{Q}_1 \in \mathbb{R}^{(m+1)\times(m+1)}$$

and let

$$\begin{aligned} \bar{R}_m &= \bar{H}_m^{(m)} = Q_m \bar{H}_m, \\ \bar{g}_m &= \beta Q_m e_1 = (\gamma_1^{(1)} \dots \gamma_m^{(m)} \gamma_{m+1}^{(m)})^T. \end{aligned}$$

Because Q_m is unitary we have

$$\min \|\beta e_1 - \bar{H}_m y\| = \min \| Q_m \left(\beta e_1 - \bar{H}_m y\right) \| = \min \|\bar{g}_m - \bar{R}_m y\|$$

QR Factorisation of $\overline{H}_m$

- The solution of the least-squares problem is obtained by solving the triangular system

$$R_m y = g_m,$$

  where $R_m \in \mathbb{R}^{m \times m}$ is obtained from $\overline{R}_m$ by deleting its last row and similarly $g_m \in \mathbb{R}^m$ is obtained from $\bar{g}_m$ by discarding its last component.
  The residual of this least-squares problem is the last component of $\bar{g}_m$.
- QR factorisation in the general case
- particular case of $\overline{H}_m$

Some properties of GMRES

Proposition

1. *The rank of AV_m is equal to the rank of R_m.*
   In particular if $r_{mm} = 0$, then A must be singular.
2. $y_m = \underset{y \in \mathbb{R}^m}{\operatorname{argmin}} \|\beta e_1 - \bar{H}_m y\|$ *is given by*

$$y_m = R_m^{-1} g_m.$$

3. *The residual vector is*

$$\begin{aligned} b - Ax_m = V_{m+1}(\beta e_1 - \bar{H}_m y_m) &= V_{m+1} Q_m^T (Q_m \beta e_1 - Q_m \bar{H}_m y_m) \\ &= V_{m+1} Q_m^T (\gamma_{m+1}^{(m)} e_{m+1}) \end{aligned}$$

then

$$\|b - Ax_m\| = |\gamma_{m+1}^{(m)}| \tag{7}$$

Incremental Factorisation of $\overline{H}_m$

The QR factorization of $\overline{H}_{m+1}$ can be cheaply computed from the QR factorization of $\overline{H}_m$.

$$\overline{H}_{m+1} = \left(\begin{array}{c|c} & h_{1,m+1} \\ \overline{H}_m & \vdots \\ & h_{m+1,m+1} \\ \hline 0 & h_{m+2,m+1} \end{array}\right)$$

It is enough to apply the m Givens rotations computed to factor $\overline{H}_m$ to the last column of $\overline{H}_{m+1}$ and build and apply a $(m+1)^{th}$ rotation to zero the $h_{m+2,m+1}$ entry.

Incremental Factorisation of $\overline{H}_m$

$$\overline{H}_6^{(5)} = \left(\begin{array}{ccccc|c} h_{11}^{(1)} & h_{12}^{(1)} & h_{13}^{(1)} & h_{14}^{(1)} & h_{15}^{(1)} & h_{16}^{(1)} \\ 0 & h_{22}^{(2)} & h_{23}^{(2)} & h_{24}^{(2)} & h_{25}^{(2)} & h_{26}^{(2)} \\ & 0 & h_{33}^{(3)} & h_{34}^{(3)} & h_{35}^{(3)} & h_{36}^{(3)} \\ & & 0 & h_{44}^{(4)} & h_{45}^{(4)} & h_{46}^{(4)} \\ & & & 0 & h_{55}^{(5)} & h_{56}^{(5)} \\ & & & & 0 & h_{66} \\ & & & & 0 & h_{76} \end{array}\right) \text{ and } \overline{\mathrm{g}}_6^{(5)} = \begin{pmatrix} \gamma_1^{(1)} \\ \gamma_2^{(2)} \\ \gamma_3^{(3)} \\ \gamma_4^{(4)} \\ \gamma_5^{(5)} \\ \gamma_6^{(5)} \\ 0 \end{pmatrix}$$

We then apply both to $\overline{H}_6^{(5)}$ and $\overline{g}_5^{(6)}$ a 6^{th} Givens rotation defined by

$$s_6 = \frac{h_{76}}{\sqrt{\left(h_{66}^{(5)}\right)^2 + h_{76}^2}} \text{ and } \mathrm{c}_6 = \frac{\mathrm{h}_{66}^{(5)}}{\sqrt{\left(\mathrm{h}_{66}^{(5)}\right)^2 + \mathrm{h}_{76}^2}}$$

to get $\overline{R}_6$ and $\overline{g}_6$.

Incremental Factorisation of $\bar{H}_m$

In particular we have $\bar{g}_6 = \begin{pmatrix} \gamma_1^{(1)} \\ \gamma_2^{(2)} \\ \gamma_3^{(3)} \\ \gamma_4^{(4)} \\ \gamma_5^{(5)} \\ c_6\gamma_6^{(5)} \\ -s_6\gamma_6^{(5)} = \gamma_7^{(6)} \end{pmatrix}$ with

$$\|b - Ax_6\| = |\gamma_7^{(6)}| = |-s_6\gamma_6^{(5)}| = |s_6||\gamma_6^{(5)}| = |s_6|\|b - Ax_5\|.$$

By induction it can be shown that

$$\|b - Ax_{m+1}\| = |s_m|\|b - Ax_m\|.$$

Remark:

1. The norm of the residual is monoticaly decreasing,
2. If $s_m = 0$, the solution is found at step m.

"Happy Breakdown" of GMRES

It exists one possible breakdown in the GMRES algorithm if $h_{k+1,k} = 0$ which prevents to increase the dimension of the search space.

Proposition

Let A be a nonsingular matrix. Then the GMRES algorithm breaks down at step k (i.e. $h_{k+1,k} = 0$) iff the iterate x_k is the exact solution of $Ax = b$.

Iterative Methods: Krylov Methods

Unsymmetric Krylov solver based on Arnoldi procedure – the costs

Computational cost and memory storage of FOM and GMRES

- n, dimension of the problem,
- $nnz(A)$, number of non-zeros of A,
- m, size of the Krylov subspace where we found a solution x_m that suits us.

FOM Algorithm

1 Set the initial guess x_0
2 $r_0 = b - Ax_0$; $\beta = \|r_0\|$; $v_1 = r_0/\beta$;
3 **for** $j = 1, 2, \ldots, m$ **do**
4 $\quad w_j = Av_j$;
5 $\quad$ **for** $i = 1, \ldots, j$ **do**
6 $\quad\quad h_{i,j} = v_i^T w_j$;
7 $\quad\quad w_j = w_j - h_{i,j} v_i$;
8 $\quad$ **end**
9 $\quad h_{j+1,j} = \|w_j\|$;
10 $\quad$ **if** $h_{j+1,j} = 0$ **then**
11 $\quad\quad$ m = j;
12 $\quad\quad$ goto 16
13 $\quad$ **end**
14 $\quad v_{j+1} = w_j / h_{j+1,j}$
15 **end**
16 $y_m = H_m^{-1}(\beta e_1)$
17 $x_m = x_0 + V_m y_m$

Algorithm 4: FOM - MGS variant - Y.Saad - Algo 6.4

Computational cost and memory storage of FOM and GMRES

FOM:

FLoating Point OPerations (Flops)

- m matrix $\times$ vector products: $\simeq 2m \times nnz(A)$ operations
- iteration j : $\simeq 4 \times j \times n$ operations (GS : `ddot` : $\simeq 2n$, `daxpy` : $\simeq 2n$)
  $=> \simeq 2m^2 n$ for m iterations
- Total : $\simeq m(2 \times nnz(A) + 2mn)$
  $+$ solution $H_m y = \beta e_1$ and matrix $\times$ vector product $V_m . y_m$

Computational cost and memory storage of FOM and GMRES

FOM:

FLoating Point OPerations (Flops)

- m matrix $\times$ vector products: $\simeq 2m \times nnz(A)$ operations
- iteration j : $\simeq 4 \times j \times n$ operations (GS : `ddot` : $\simeq 2n$, `daxpy` : $\simeq 2n$)
  $=> \simeq 2m^2 n$ for m iterations
- Total : $\simeq m(2 \times nnz(A) + 2mn)$
  $+$ solution $H_m y = \beta e_1$ and matrix $\times$ vector product $V_m.y_m$

Memory storage

- base V_m, vectors x_0, x_m, w_j, matrix $\overline{H}_m$
- Total : $\simeq (m+3)n + \frac{m^2}{2}$

Computational cost and memory storage very similar for **GMRES**.

Alternative Versions

Goal: Reduce the computational cost and/or the memory storage

- restarted versions: `FOM(m)` and `GMRES(m)`
- truncated versions: `DIOM(m)` and `DQGMRES(m)`

Restarted versions

- the idea of these versions is to fix a maximum size for the Krylov subspace m,
- if the solution x_m in this subspace does not suit us, we start again and build a new subspace taking the solution x_m as the initial guess x_0,
- and this as many times it is necessary to obtain a solution that verifies the convergence criteria.
- GMRES(m) algorithm

1 Set the initial guess x_0
2 Call GMRES with stopping criteria ($\eta_b^N(x_k) \leq \varepsilon$ or $nbit = m$)
3 **if** $\eta_b^N(x_k) > \varepsilon$ **then**
4 | $x_0 = x_m$
5 | goto 2
6 **end**

- m, maximum size is a parameter of this algorithm and depends of constraints of the target computer.

Truncated versions

- the idea of these versions is to maintain orthogonality only among the last m "Arnoldi's" vectors

- that leads to
  - computational gains during orthogonalisation procedure,
  - sparse structure de $\overline{H}_m$ that can be exploited
    - incremental factorisation **LU** for `DIOM(m)` ,
    - simplified incremental factorisation **QR** for `DQGMRES(m)`,
  - memory storage gain of the basis V_m (we can exhibit a short recurrence between x_{m+1} and x_m) [see the idea when describing symmetric solvers].

Iterative Methods: Symmetric Krylov Solvers

Arnoldi for symmetric problems: the Lanczos method

Proposition

Assume that the Arnoldi's method is applied to a real symmetric matrix A. The matrix H_m is tridiagonal and symmetric.

Exercise

Prove the Proposition

The standard notation used to described the Lanczos algorithm is to used T_m rather than H_m and $\alpha_j \equiv h_{j,j}$ and $\beta_j \equiv h_{j-1,j} = h_{j,j-1}$.

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{pmatrix}.$$

Mr Lanczos

Cornelius Lanczos

1893-1974: Hungary.

Lanczos worked on relativity and mathematical physics and invented what is now called the Fast Fourier Transform. He worked with Einstein in Berlin, for the Boeing Aircraft Company, at Purdue University and the Department at the Dublin Institute for Advance Study in Ireland.

Lanczos algorithm

1: Choose an initial vector v_1 of norm equal to one.
2: $\beta_1 = 0$ and $v_0 = 0$
3: **for** $j = 1, 2, \ldots m$ **do**
4: $\quad w_j = Av_j - \beta_j v_{j-1}$
5: $\quad \alpha_j = w_j^T v_j$
6: $\quad w_j = w_j - \alpha_j v_j$
7: $\quad \beta_{j+1} = \|w_j\|$
8: $\quad$ If β_{j+1}=0 then Stop.
9: $\quad v_{j+1} = w_j/\beta_{j+1}$
10: **end for**

The following "Arnoldi" relations read:

$$AV_m = V_m T_m + \beta_{m+1} v_{m+1} e_m^T, \quad (8)$$

$$V_m^T A V_m = T_m \quad (9)$$

Arnoldi relations for a symmetric problem

Arnoldi relations:

$$\begin{aligned} AV_m &= V_m H_m + h_{m+1,m} v_{m+1} e_m^T \\ AV_m &= V_{m+1} \bar{H}_m \\ V_m^T A V_m &= H_m \end{aligned}$$

Arnoldi relations when the matrix is symmetric:

$$\begin{aligned} AV_m &= V_m T_m + \beta_{m+1} v_{m+1} e_m^T, \\ V_m^T A V_m &= T_m \end{aligned}$$

From Arnoldi procedure to Lanczos algorithm

1: $v_1 = b/\|b\|$;
2: **for** j = 1, 2, ..., m **do**
3: $w_j = Av_j$;
4: **for** $i = 1, \ldots, j$ **do**
5: $h_{i,j} = v_i^T w_j$;
6: $w_j = w_j - h_{i,j} v_i$;
7: **end for**
8: Compute $h_{j+1,j} = \|w_j\|$;
9: exit if $(h_{j+1,j} = 0)$;
10: Compute $v_{j+1} = w_j / h_{j+1,j}$
11: **end for**

Algorithm 5: Arnoldi (MGS)

1: Choose an initial vector v_1 of norm equal to one.
2: $\beta_1 = 0$ and $v_0 = 0$
3: **for** $j = 1, 2, \ldots m$ **do**
4: $w_j = Av_j - \beta_j v_{j-1}$
5: $\alpha_j = w_j^T v_j$
6: $w_j = w_j - \alpha_j v_j$
7: $\beta_{j+1} = \|w_j\|$
8: If β_{j+1}=0 then Stop.
9: $v_{j+1} = w_j / \beta_{j+1}$
10: **end for**

Algorithm 6: Lanczos

Iterative Methods: Symmetric Krylov Solvers

Lanczos algorithm for symmetric linear systems

Lanczos algorithm for symmetric linear systems

- Galerkin approach:
  search $x_m \in x_0 + \mathcal{K}_m$ such that $b - Ax_m \perp \mathcal{K}_m$

Exercise

how express x_m?

Lanczos algorithm for symmetric linear systems

1: $r_0 = b - Ax_0$; $\beta = \|r_0\|$; $v_1 = r_0/\beta$
2: **for** $j = 1, 2, \ldots m$ **do**
3: $\quad w_j = Av_j - \beta_j v_{j-1}$ (If $j = 1$ set $\beta_1 v_0 = 0$)
4: $\quad \alpha_j = w_j^T v_j$
5: $\quad w_j = w_j - \alpha_j v_j$
6: $\quad \beta_{j+1} = \|w_j\|$
7: $\quad$ **if** β_{j+1}=0 **then**
8: $\quad\quad m = j$; goto 12
9: $\quad$ **end if**
10: $\quad v_{j+1} = w_j/\beta_{j+1}$
11: **end for**
12: Set $T_m = tridiag(\beta_i, \alpha_i, \beta_{i+1})$ and $V_m = (v_1, \cdots, v_m)$.
13: Compute $y_m = T_m^{-1}(\beta e_1)$ and $x_m = x_0 + V_m y_m$.

Residual property

Exercise

Show that the residual associated with x_m is colinear to v_{m+1} ?

We have (third time?)

$$\begin{aligned} b - Ax_m &= b - A(x_0 + V_m y_m) \\ &= r_0 - AV_m y_m \\ &= V_m(\beta e_1) - (V_m T_m + \beta_{m+1} v_{m+1} e_m^T) y_m \text{ (from (8))} \\ &= V_m \underbrace{(\beta e_1 - T_m y_m)}_{0} - \beta_{m+1} v_{m+1} e_m^T y_m. \end{aligned}$$

Storage considerations

1. T_m, 2 vectors
2. To compute v_{j+1}, we need v_{j-1} and v_j, 3 vectors?
   $=>$ NO, we still need the base V_m to compute x_m

Iterative Methods: Symmetric Krylov Solvers

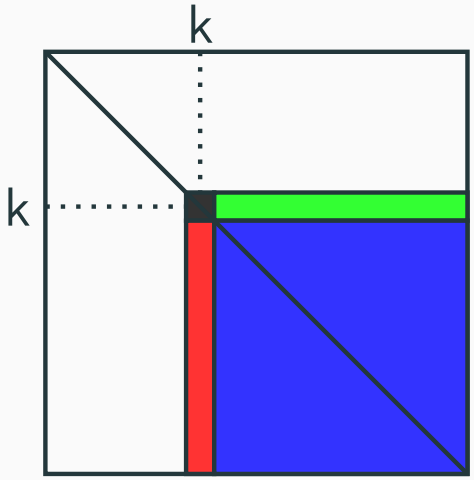
The D-Lanczos variant

The D-Lanczos variant

As T_m is tridiagonal, we can compute an incremental factorisation **L.U** of T_m and we will show that:

- factors can be updated at each Lanczos iteration,
- we do not have to store all the basis V_m to compute x_m.

Rappel: factorisation $L.U$



Factorisation $L.U$:

For k from 1 to $n-1$:

1. Compute column k of L:
   $A(k+1:n,k) = A(k+1:n,k)/A(k,k)$
2. Update the sub-block:
   $$\begin{aligned} & A(k+1:n,k+1:n) \\ =& A(k+1:n,k+1:n) \\ -& A(k+1:n,k) \times A(k,k+1:n) \end{aligned}$$

Factorisation $L.U$ of $T_m = L_m \times U_m$

Exemple : $T_5 = L_5 \times U_5$

$$T_5 = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \beta_4 & \\ & & \beta_4 & \alpha_4 & \beta_5 \\ & & & \beta_5 & \alpha_5 \end{pmatrix}$$

Factorisation $L.U$ of $T_m = L_m \times U_m$

We show that

- L and U are bi-diagonal matrices
- the coefficients of the upper diagonal of U_m are the coefficients β_i of T_m

$$T_5 = \begin{pmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \lambda_3 & 1 & & \\ & & \lambda_4 & 1 & \\ & & & \lambda_5 & 1 \end{pmatrix} \times \begin{pmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \eta_3 & \beta_4 & \\ & & & \eta_4 & \beta_5 \\ & & & & \eta_5 \end{pmatrix}$$

Factorisation LU of $T_m = L_m \times U_m$

Another way to represent the factors L and U in the same memory space A

$$memory(A) \quad := \quad \begin{pmatrix} \eta_1 & \beta_2 & & & \\ \lambda_2 & \eta_2 & \beta_3 & & \\ & \lambda_3 & \eta_3 & \beta_4 & \\ & & \lambda_4 & \eta_4 & \beta_5 \\ & & & \lambda_5 & \eta_5 \end{pmatrix}$$

(remember factorization L.U)

How compute coefficients of L_m and U_m

- Using a recurrence, we show that

$$\lambda_m = \frac{\beta_m}{\eta_{m-1}},\ m \in \{2, \ldots\}$$

$$\eta_m = \alpha_m - \lambda_m \beta_m,\ m \in \{1, \ldots\}, \beta_1 = 0$$

Expression of x_m

The approximate solution is given by

$$x_m = x_0 + \underbrace{V_m U_m^{-1}}_{P_m} \underbrace{L_m^{-1}(\beta e_1)}_{z_m},$$

then

$$x_m = x_0 + P_m z_m.$$

Because of the structure of U_m, we can easily update P_m and compute its last column using the previous p_i's and v_m.

Equating the last columns of $P_m U_m = V_m$ gives

$$\begin{aligned} & \eta_m p_m + \beta_m p_{m-1} = v_m \\ \Rightarrow \quad & p_m = \eta_m^{-1} \left(v_m - \beta_m p_{m-1} \right). \end{aligned}$$

Expression of x_m

Similarly, we can derive an update for z_m exploiting the structure of L_m (bidiagonal matrix)

$$z_m = \begin{pmatrix} z_{m-1} \\ \xi_m \end{pmatrix},$$

where $\xi_m = -\lambda_m \xi_{m-1}$.

Consequently we have

$$x_m = x_0 + (P_{m-1} p_m) \begin{pmatrix} z_{m-1} \\ \xi_m \end{pmatrix} = \underbrace{x_0 + P_{m-1} z_{m-1}}_{x_{m-1}} + \xi_m p_m$$

- Short Relation between x_m et x_{m-1} : $x_m = x_{m-1} + \xi_m p_m$
  $=>$ no more storage of V_m
- Memory usage : a vector for p_m and some scalars

D-Lanczos algorithm

1: Set $r_0 = b - Ax_0$; $\xi_1 = \beta = \|r_0\|$ and $v_1 = r_0/\beta$
2: Set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
3: **for** $j = 1, \ldots$ **do**
4: Compute $w_j = Av_j - \beta_j v_{j-1}$ and $\alpha_j = w_j^T v_j$
5: **if** $j > 1$ **then**
6: computed $\lambda_j = \dfrac{\beta_j}{\eta_{j-1}}$ and $\xi_j = -\lambda_j \xi_{j-1}$
7: **end if**
8: $\eta_j = \alpha_j - \lambda_j \beta_j$
9: $p_j = \eta_j^{-1}(v_j - \beta_j p_{j-1})$
10: $x_j = x_{j-1} + \xi_j p_j$
11: **if** converged **then**
12: Stop
13: **end if**
14: $w_j = w_j - \alpha_j v_j$
15: $\beta_{j+1} = \|w_j\|$, $v_{j+1} = w_j/\beta_{j+1}$
16: **end for**

Expression of the norm of the residual

Exercise

Show that: $\|b - Ax_m\| = \beta_{m+1} |\frac{\xi_m}{\eta_m}|$

Some important properties

Proposition

Let $r_m = b - Ax_m$ and p_m the vectors generated by the D-Lanczos algorithm. We have:

1. *the residual vector r_m is colinear to v_{m+1}. As a consequence, the residual vectors are orthogonal to each other.*
2. *the vectors p_i are A-conjugate, that is: $\forall i \neq j$, $p_i^T A p_j = 0$.*

Iterative Method: Conjugate Gradient

D-Lanczos algorithm

1: Set $r_0 = b - Ax_0$; $\xi_1 = \beta = \|r_0\|$ and $v_1 = r_0/\beta$
2: Set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
3: **for** $j = 1, \ldots$ **do**
4: Compute $w_j = Av_j - \beta_j v_{j-1}$ and $\alpha_j = w_j^T v_j$
5: **if** $j > 1$ **then**
6: computed $\lambda_j = \dfrac{\beta_j}{\eta_{j-1}}$ and $\xi_j = -\lambda_j \xi_{j-1}$
7: **end if**
8: $\eta_j = \alpha_j - \lambda_j \beta_j$
9: $p_j = \eta_j^{-1}(v_j - \beta_j p_{j-1})$
10: $x_j = x_{j-1} + \xi_j p_j$ // new iterate function of the previous one
11: **if** converged **then**
12: Stop
13: **end if**
14: $w_j = w_j - \alpha_j v_j$
15: $\beta_{j+1} = \|w_j\|$, $v_{j+1} = w_j/\beta_{j+1}$
16: **end for**

D-Lanczos algorithm

1: Set $r_0 = b - Ax_0$; $\xi_1 = \beta = \|r_0\|$ and $v_1 = r_0/\beta$
2: Set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
3: **for** $j = 1, \ldots$ **do**
4: Compute $w_j = Av_j - \beta_j v_{j-1}$ and $\alpha_j = w_j^T v_j$
5: **if** $j > 1$ **then**
6: computed $\lambda_j = \dfrac{\beta_j}{\eta_{j-1}}$ and $\xi_j = -\lambda_j \xi_{j-1}$
7: **end if**
8: $\eta_j = \alpha_j - \lambda_j \beta_j$
9: $p_j = \eta_j^{-1}(v_j - \beta_j p_{j-1})$ // we need to compute a new direction
10: $x_j = x_{j-1} + \xi_j p_j$ // new iterate function of the previous one
11: **if** converged **then**
12: Stop
13: **end if**
14: $w_j = w_j - \alpha_j v_j$
15: $\beta_{j+1} = \|w_j\|$, $v_{j+1} = w_j/\beta_{j+1}$
16: **end for**

D-Lanczos algorithm

1: Set $r_0 = b - Ax_0$; $\xi_1 = \beta = \|r_0\|$ and $v_1 = r_0/\beta$
2: Set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
3: **for** $j = 1, \ldots$ **do**
4: Compute $w_j = Av_j - \beta_j v_{j-1}$ and $\alpha_j = w_j^T v_j$
5: **if** $j > 1$ **then**
6: computed $\lambda_j = \dfrac{\beta_j}{\eta_{j-1}}$ and $\xi_j = -\lambda_j \xi_{j-1}$
7: **end if**
8: $\eta_j = \alpha_j - \lambda_j \beta_j$
9: $p_j = \eta_j^{-1}(cr_{j-1} - \beta_j p_{j-1})$ // r_{j-1} and v_j are colinears
10: $x_j = x_{j-1} + \xi_j p_j$
11: **if** converged **then**
12: Stop
13: **end if**
14: $w_j = w_j - \alpha_j v_j$
15: $\beta_{j+1} = \|w_j\|$, $v_{j+1} = w_j/\beta_{j+1}$
16: **end for**

Conjugate Gradient can be derived from D-Lanczos

With a translation of p_j index to conform with standard notation, we express step 10 of the algorithm:

$$x_{j+1} = x_j + \delta_j p_j \tag{10}$$

and as consequence

$$r_{j+1} = r_j - \delta_j A p_j. \tag{11}$$

Thus a first expression of δ_j

$$\delta_j = \frac{r_j^T r_j}{r_j^T A p_j}. \tag{12}$$

Conjugate Gradient can be derived from D-Lanczos (cont.)

We are looking for a new direction p_{j+1} as a linear combination of r_{j+1} et p_j (step 9 with translation of p_j index)

$$p_{j+1} = r_{j+1} + \gamma_j p_j, \tag{13}$$

We show that we can re-write δ_j

$$\delta_j = \frac{r_j^T r_j}{p_j^T A p_j}. \tag{14}$$

Then a first expression for γ_j is

$$\gamma_j = -\frac{r_{j+1}^T A p_j}{p_j^T A p_j}, \tag{15}$$

that can be eventually written as

$$\gamma_j = \frac{r_{j+1}^T r_{j+1}}{r_j^T r_j}. \tag{16}$$

Conjugate Gradient Algorithm

To use the standard notation, let's rename $\alpha_j = \delta_j$ and $\beta_j = \gamma_j$ and summarize the two previous slides:

1: $r_0 = b - Ax_0$, $p_0 = r_0$.
2: **for** $j = 0, 1, \ldots$ **do**
3: $\quad \alpha_j = (r_j^T r_j)/(p_j^T A p_j)$
4: $\quad x_{j+1} = x_j + \alpha_j p_j$
5: $\quad r_{j+1} = r_j - \alpha_j A p_j$
6: $\quad \beta_j = (r_{j+1}^T r_{j+1})/(r_j^T r_j)$
7: $\quad p_{j+1} = r_{j+1} + \beta_j p_j$
8: **end for**

- Conjugate Gradient is a Krylov solver!
- Beware: α_j and β_j of GC have nothing to do with those of D-Lanczos algorithm.
- Memory usage: 4 vectors (x, p, Ap, r) .vs. 5 for D-Lanczos (v_m, v_{m-1}, w, p, x).

Some important remarks

The algorithm relies on a LU factorization without pivoting of a symmetric tridiagonal matrix. This factorization might not exist and CG can break-down.

1. For symmetric positive definite (SPD) matrices, T_m is always non singular and a stable LU decomposition exists. **CG** is then the **method of choice** for this class of matrices.
2. For symmetric indefinite matrices a technique based on a **LQ** decomposition of T_m exists that is known as **SYMMLQ**.
3. For symmetric indefinite matrices, the solution of the linear system involving T_m can be replaced by the solution of a linear least-square involving $\bar{T}_m$. This method is known as **MINRES** and belong to the minimum norm error class of Krylov solvers.

Conjugate Gradient properties for SPD case (1)

Proposition

Because A is SPD, the bilinear form $x^T Ay$ define an inner product.

The Galerkin condition $b - Ax_m \perp \mathcal{K}_m$ can be written $A(x_m - x^\star) \perp \mathcal{K}_m$ which also reads $(x_m - x^\star) \perp_A \mathcal{K}_m$.

This latter condition implies that

$$\|x_m - x^\star\|_A$$

is minimal over $\mathcal{K}_m$

Conjugate Gradient properties for SPD case (2)

We have the following upper-bound for the convergence rate of CG in the SPD case.

Proposition

Let x_m be the m^{th} iterate generated by the CG algorithm then

$$\|x_m - x^\star\|_A \leq 2 \cdot \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^m \|x_0 - x^\star\|_A.$$

Iterative Methods: Algebraic preconditioning techniques

Iterative Methods: Algebraic preconditioning techniques

Some backgrounds

Some properties of Krylov solvers

- CG

$$\|x_m - x^\star\|_A = \min_{p \in \mathbb{P}_m,\, p(0)=1} \|p(A)(x_0 - x^\star)\|_A$$

- GMRES, MINRES

$$\|r_m\| = \min_{p \in \mathbb{P}_m,\, p(0)=1} \|p(A)(r_0)\|_2$$

- Bound on the rate of convergence of CG:

$$\|x_m - x^\star\|_A \leq 2 \cdot \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^m \|x_0 - x^\star\|_A.$$

Some properties of Krylov solvers

- Assumption: if A diagonalizable with m distinct eigenvalues then CG, GMRES, MINRES converges in at most m steps (Cayley-Hamilton theorem - minimal polynomial).
- Assumption: if r_0 has k components in the eigenbasis then CG, GMRES, MINRES converges in k steps.

Iterative Methods: Algebraic preconditioning techniques

Driving principles

Driving principles to design preconditioners

Find a non-singular matrix M such that $M.A$ has "better" properties v.s. the convergence behaviour of the selected Krylov solver:

- $M.A$ has less distinct eigenvalues
- $M.A \approx I$ in some sense

The preconditioner should

- be cheap to compute and to store,
- be cheap to apply,
- ensure a fast convergence.

With a good preconditioner the solution time for the preconditioned system should be significantly less that for the unpreconditioned system.

Iterative Methods: Algebraic preconditioning techniques

Preconditioned CG

The particular case of CG

- for CG, M must be symmetric (to keep a symmetric linear system)
- M be given in a factorized form CC^T
- CG can be applied to $\tilde{A}\tilde{x} = \tilde{b}$
  $\tilde{A} = C^T AC$, $C\tilde{x} = x$ and $\tilde{b} = C^T b$ (Saad 9.4 with different notations).

Notations

- Let $x_k = C\tilde{x}_k$; $p_k = C\tilde{p}_k$; $\tilde{r}_k = C^T r_k$; $z_k = CC^T r_k = Mr_k$
- Conjugate Gradient algorithm for preconditioned system:

1: $\tilde{r}_0 = \tilde{b} - \tilde{A}\tilde{x}_0$, $\tilde{p}_0 = \tilde{r}_0$.
2: **for** $k = 0, 1, \ldots$ **do**
3: $\quad \alpha_k = (\tilde{r}_k{}^T \tilde{r}_k)/(\tilde{p}_k{}^T \tilde{A}\tilde{p}_k)$
4: $\quad \tilde{x}_{k+1} = \tilde{x}_k + \alpha_k \tilde{p}_k$
5: $\quad \tilde{r}_{k+1} = \tilde{r}_k - \alpha_k \tilde{A}\tilde{p}_k$
6: $\quad \beta_k = (\tilde{r}_{k+1}^T \tilde{r}_{k+1})/(\tilde{r}_k{}^T \tilde{r}_k)$
7: $\quad \tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k \tilde{p}_k$
8: $\quad$ **if** Convergence Stop
9: **end for**

- Let's re-arrange the algorithm => PCG algorithm

PCG Algorithm

PCG Algorithm

Writing the algorithm only using the unpreconditioned variables leads to:

Preconditioned Conjugate Gradient algorithm

1. Compute $r_0 = b - Ax_0$, $z_0 = Mr_0$ and $p_0 = z_0$
2. For k=0,2, ... Do
3. $\alpha_k = r_k^T z_k / p_k^T A p_k$
4. $x_{k+1} = x_k + \alpha_k p_k$
5. $r_{k+1} = r_k - \alpha_k A p_k$
6. $z_{k+1} = M r_{k+1}$
7. $\beta_k = r_{k+1}^T z_{k+1} / r_k^T z_k$
8. $p_{k+1} = z_{k+1} + \beta_k p_k$
9. if x_k accurate enough then stop
10. EndDo

Properties

Exercise

1. *A-norm minimization*
   - *A-norm of preconditioned system* $\|\tilde{x}_k - \tilde{x}^\star\|_{\tilde{A}}$ *is minimal over* $\mathcal{K}(\tilde{A}, \tilde{b}, k)$
   - $\|x_k - x^\star\|_A$*?*
2. *Orthogonality*
   - p_k*?*
   - r_k*?*

Iterative Methods: Algebraic preconditioning techniques

Preconditioner taxonomy and examples

Preconditioner taxonomy and examples

There are two main classes of preconditioners

- **Implicit preconditioners:**
  approximate A with a matrix M such that solving the linear system $Mz = r$ is easy.
- **Explicit preconditioners:**
  approximate A^{-1} with a matrix M and just perform $z = Mr$.

The governing ideas in the design of the preconditioners are very similar to those followed to define iterative stationary schemes. Consequently, all the stationary methods can be used to define preconditioners.

Stationary methods

Let x_0 be given and $M \in \mathbb{R}^{n\times n}$ a nonsingular matrix, compute

$$x_k = x_{k-1} + M(b - Ax_{k-1}).$$

Note that $b - Ax_{k-1} = A(x^* - x_{k-1}) \Rightarrow$ the best M is A^{-1}.

The stationary sheme converges to $x^* = A^{-1}b$ for any x_0 iff $\rho(I - MA) < 1$, where $\rho(\cdot)$ denotes the spectral radius.

Let $A = L + D + U$

- $M = I$: Richardson method,
- $M = D^{-1}$: Jacobi method,
- $M = (L + D)^{-1}$: Gauss-Seidel method.

Notice that M has always a special structure and the inverse must never been explicitely computed ($z = B^{-1}y$ reads *solve the linear system* $Bz = y$).

Preconditioner location

Several possibilities exist to solve $Ax = b$:

- Left preconditioner
$$MAx = Mb.$$
- Right preconditioner
$$AMy = b \text{ with } x = My.$$
- Split preconditioner if $M = M_1 M_2$
$$M_2 A M_1 y = M_2 b \text{ with } x = M_1 y.$$

Notice that the spectrum of MA, AM and $M_2 A M_1$ are identical (for any matrices B and C, the eigenvalues of BC are the same as those of CB)

Preconditioner location v.s. stopping criterion

The stoping criterion are based on backward error

$$\eta_{A,b}^{N} = \frac{\|b - Ax\|}{\|A\|\|x\| + \|b\|} < \varepsilon \text{ or } \eta_{\mathrm{b}}^{\mathrm{N}} = \frac{\|\mathrm{b} - \mathrm{Ax}\|}{\|\mathrm{b}\|} < \varepsilon.$$

In PCG we can still compute η.
For GMRES, using a preconditioner means runing GMRES on

Left precond.	Right precond.	Split precond.
$MAx = Mb$	$AMy = b$	$M_2AM_1y = M_2b$

The free estimate of the residual norm of GMRES is associated with the preconditioned system.

Preconditioner location v.s. stopping criterion (cont)

	Left precond.	Right precond.	Split precond.
$\eta_M(A,b)$	$\frac{\|MAx-Mb\|}{\|MA\|\|x\|+\|Mb\|}$	$\frac{\|AMy-b\|}{\|AM\|\|x\|+\|b\|}$	$\frac{\|M_2AM_1y-M_2b\|}{\|M_2AM_1\|\|y\|+\|M_2b\|}$
$\eta_M(b)$	$\frac{\|MAx-Mb\|}{\|Mb\|}$	$\frac{\|AMy-b\|}{\|b\|}$	$\frac{\|M_2AM_1y-M_2b\|}{\|M_2b\|}$

Using $\eta_M(b)$ for right preconditioned linear system will monitor the convergence as if no preconditioner was used.

Iterative Methods: Algebraic preconditioning techniques

Some classical algebraic preconditioners

Some classical algebraic preconditioner

- Incomplete factorization : IC, $ILU(p)$, $ILU(p, \tau)$
- SPAI (Sparse Approximate Inverse): compute the sparse approximate inverse by minimizing the Frobenius norm $\|MA - I\|_F$
- FSAI (Factorized Sparse Approximate inverse): compute the sparse approximate inverse of the Cholesky factor by minimizing the Frobenius norm $\|I - GL\|_F$
- AINV (Approximate Inverse): compute the sparse approximate inverse of the LDU or LDL^T factors using an incomplete biconjugation process

Incomplete factorizations

One variant of the LU factorization writes:

IKJ variant - Top looking variant	
1.	**for** $i = 2, ..., n$ **do**
2.	**for** $k = 1, ..., i-1$ **do**
3.	$a_{i,k} = a_{i,k}/a_{k,k}$
4.	**for** $j = k+1, ..., n$ **do**
5.	$a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$
5.	**end for**
6.	**end for**
7.	**end for**

Zero fill-in ILU - $ILU(0)$

Let denote $NZ(A)$ the set of (rwo,column) index of the nonzero entries of A.

ILU(0)	
1.	**for** $i = 2, ..., n$ **do**
2.	**for** $k = 1, ..., i-1$ and $(i,k) \in NZ(A)$ **do**
3.	$a_{i,k} = a_{i,k} / a_{k,k}$
4.	**for** $j = k+1, ...,$n and $(i,j) \in NZ(A)$ **do**
5.	$a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$
6.	**end for**
7.	**end for**
8.	**end for**

Level of fill in ILU - $ILU(p)$

Definition

The initial level of fill of an entry $a_{i,j}$ is defined by:

$$lev(i,j) = \begin{cases} 0 & \text{if } a_{i,j} \neq 0 \text{ or i} = \text{j}, \\ \infty & \text{otherwise.} \end{cases}$$

Each time this entry is modified in line 5 of the LU top looking algorithm, its level fill is updated by

$$lev(i,j) = \min\{lev(i,j), lev(i,k) + lev(k,j) + 1\}. \tag{17}$$

A first example

$$A = \begin{pmatrix} x & x & x & x \\ x & x & & \\ x & & x & \\ x & & & x \end{pmatrix} \quad lev = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \infty & \infty \\ 0 & \infty & 0 & \infty \\ 0 & \infty & \infty & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

ILU(p)	
1.	**for** all nonzero entries $a_{i,j}$ set $lev_{i,j} i = 0$
2.	**for** $i = 2, ..., n$ **do**
3.	**for** $k = 1, ..., i-1$ and $lev_{i,k} \leq p$ **do**
4.	$a_{i,k} = a_{i,k}/a_{k,k}$
5.	**for** $j = k+1, ...,$n **do**
6.	$a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$
7.	$lev(i,j) = \min\{lev(i,j), lev(i,k) + lev(k,j) + 1\}$
8.	**if** $lev(i,j) > p$ **then** $a_{i,j} = 0$
9.	**end for**
10.	**end for**
11.	**end for**

$ILU(p, \tau)$ dual treshold strategy

- Fix a drop tolerance τ and a number of fill p to be allowed in each row of the incomplete LU factors. At each step of the elimination process, drop all fill-ins that are smaller than τ times the 2-norm of the current row; for all the remaining ones keep only the p largest.
- Trade-off between amount of fill-in (construction time and application time for the preconditioner) and decrease of number of iterations.

Iterative Methods: Algebraic preconditioning techniques

Spectral preconditioners

Spectral preconditioners

Motivations

- The convergence of Krylov methods for solving the linear system often depends to a large extent on the eigenvalue distribution. In many cases, it is observed that "removing" the smallest eigenvalues can greatly improve the convergence
- Many preconditioners are able to cluster most of the eigenvalues close to one but still leave a few close to the origin
- "Moving" these eigenvalues by tuning the parameters that control these preconditioners is often difficult and might lead to very expensive preconditioners to set-up and to apply
- Performing a low rank update might enable to shift the smallest eigenvalues.

Iterative Methods: Algebraic preconditioning techniques

Multigrid preconditioners

Multigrid preconditioners

Part 2 with Carola Kruse

Iterative Methods: Algebraic preconditioning techniques

Preconditioners built from a Domain Decomposition Method

Preconditioners built from a Domain Decomposition Method

Part 3 with Alena Kopanicakova