Scientific Computing WS 2018/2019

Lecture 5

Jürgen Fuhrmann

juergen.fuhrmann@wias-berlin.de

Recap from last time

Matrix + Vector norms

- Vector norms: let $x = (x_i) \in \mathbb{R}^n$
  - $||x||_1 = \sum_{i=1}^{n} |x_i|$: sum norm, l_1-norm
  - $||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$: Euclidean norm, l_2-norm
  - $||x||_\infty = \max_{i=1}^{n} |x_i|$: maximum norm, l_∞-norm
- Matrix $A = (a_{ij}) \in \mathbb{R}^n \times \mathbb{R}^n$
  - Representation of linear operator $\mathcal{A} : \mathbb{R}^n \to \mathbb{R}^n$ defined by $\mathcal{A} : x \mapsto y = Ax$ with

$$y_i = \sum_{j=1}^{n} a_{ij} x_j$$

  - Induced matrix norm:

$$\begin{aligned} ||A||_\nu &= \max_{x \in \mathbb{R}^n, x \neq 0} \frac{||Ax||_\nu}{||x||_\nu} \\ &= \max_{x \in \mathbb{R}^n, ||x||_\nu = 1} \frac{||Ax||_\nu}{||x||_\nu} \end{aligned}$$

Matrix norms

- $||A||_1 = \max_{j=1}^{n} \sum_{i=1}^{n} |a_{ij}|$ maximum of column sums
- $||A||_\infty = \max_{i=1}^{n} \sum_{j=1}^{n} |a_{ij}|$ maximum of row sums
- $||A||_2 = \sqrt{\lambda_{max}}$ with λ_{max}: largest eigenvalue of $A^T A$.

Matrix condition number and error propagation

- Problem: solve $Ax = b$, where b is inexact
- Let Δb be the error in b and Δx be the resulting error in x such that

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

- Since $Ax = b$, we get $A\Delta x = \Delta b$
- Therefore

$$\left\{ \begin{array}{ll} \Delta x & = A^{-1}\Delta b \\ Ax & = b \end{array} \right\} \Rightarrow \left\{ \begin{array}{rl} ||A|| \cdot ||x|| & \geq ||b|| \\ ||\Delta x|| & \leq ||A^{-1}|| \cdot ||\Delta b|| \end{array} \right.$$

$$\Rightarrow \frac{||\Delta x||}{||x||} \leq \kappa(A) \frac{||\Delta b||}{||b||}$$

where $\kappa(A) = ||A|| \cdot ||A^{-1}||$ is the *condition number* of A.

Solution of linear systems of equations

Approaches to linear system solution

Let A: $n \times n$ matrix, $b \in \mathbb{R}^n$.

Solve $Ax = b$

- Direct methods:
  - Exact
    - up to machine precision
    - condition number
  - Expensive (in time and space)
    - where does this matter ?
- Iterative methods:
  - Only approximate
    - with good convergence and proper accuracy control, results are not worse than for direct methods
  - May be cheaper in space and (possibly) time
  - Convergence guarantee is problem dependent and can be tricky

Complexity: "big O notation"

- Let $f, g : \mathbb{V} \to \mathbb{R}^+$ be some functions, where $\mathbb{V} = \mathbb{N}$ or $\mathbb{V} = \mathbb{R}$.

  We write

  $$f(x) = O(g(x)) \quad (x \to \infty)$$

  if there exist a constant $C > 0$ and $x_0 \in \mathbb{V}$ such that

  $$\forall x > x_0, \quad |f(x)| \leq C|g(x)|$$

- Often, one skips the part "$(x \to \infty)$"
- Examples:
  - Addition of two vectors: $O(n)$
  - Matrix-vector multiplication (for matrix where all entries are assumed to be nonzero): $O(n^2)$

Really bad example of direct method

Solve $Ax = b$ by Cramer's rule

$$x_i = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1i-1} & b_1 & a_{1i+1} & \dots & a_{1n} \\ a_{21} & & \dots & & b_2 & & \dots & a_{2n} \\ \vdots & & & & \vdots & & & \vdots \\ a_{n1} & & \dots & & b_n & & \dots & a_{nn} \end{vmatrix} / |A| \quad (i = 1 \dots n)$$

This takes $O(n!)$ operations...

Gaussian elimination

- Essentially the only feasible direct solution method
- Solve $Ax = b$ with square matrix A.
- While formally, the algorithm is always the same, its implementation depends on
  - data structure to store matrix
  - possibility to ignore zero entries for matrices with many zeroes
  - sorting of elements

Gaussian elemination: pass 1

$$\begin{pmatrix} 6 & -2 & 2 \\ 12 & -8 & 6 \\ 3 & -13 & 3 \end{pmatrix} x = \begin{pmatrix} 16 \\ 26 \\ -19 \end{pmatrix}$$

Step 1: $\text{equation}_2 \leftarrow \text{equation}_2 - 2\,\text{equation}_1$
$\text{equation}_3 \leftarrow \text{equation}_3 - \frac{1}{2}\,\text{equation}_1$

$$\begin{pmatrix} 6 & -2 & 2 \\ 0 & -4 & 2 \\ 0 & -12 & 2 \end{pmatrix} x = \begin{pmatrix} 16 \\ -6 \\ -27 \end{pmatrix}$$

Step 2: $\text{equation}_3 \leftarrow \text{equation}_3 - 3\,\text{equation}_2$

$$\begin{pmatrix} 6 & -2 & 2 \\ 0 & -4 & 2 \\ 0 & 0 & -4 \end{pmatrix} x = \begin{pmatrix} 16 \\ -6 \\ -9 \end{pmatrix}$$

Gaussian elimination: pass 2

Solve upper triangular system

$$\begin{pmatrix} 6 & -2 & 2 \\ 0 & -4 & 2 \\ 0 & 0 & -4 \end{pmatrix} x = \begin{pmatrix} 16 \\ -6 \\ -9 \end{pmatrix}$$

$$\begin{aligned} -4x_3 &= -9 && && \Rightarrow x_3 = \frac{9}{4} \\ -4x_2 + 2x_3 &= -6 && \Rightarrow -4x_2 = -\frac{21}{2} && \Rightarrow x_2 = \frac{21}{8} \\ 6x_1 - 2x_2 + 2x_3 &= 2 && \Rightarrow 6x_1 = 2 + \frac{21}{4} - \frac{18}{4} = \frac{11}{4} && \Rightarrow x_1 = \frac{11}{4} \end{aligned}$$

LU factorization

Pass 1 expressed in matrix operation

$$L_1 A x = \begin{pmatrix} 6 & -2 & 2 \\ 0 & -4 & 2 \\ 0 & -12 & 2 \end{pmatrix} x = \begin{pmatrix} 16 \\ -6 \\ -27 \end{pmatrix} = L_1 b, \qquad L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix}$$

$$L_2 L_1 A x = \begin{pmatrix} 6 & -2 & 2 \\ 0 & -4 & 2 \\ 0 & 0 & -4 \end{pmatrix} x = \begin{pmatrix} 16 \\ -6 \\ -9 \end{pmatrix} = L_2 L_1 b, \qquad L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix}$$

- Let $L = L_1^{-1} L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & 3 & 1 \end{pmatrix}$, $U = L_2 L_1 A$. Then $A = LU$
- Inplace operation. Diagonal elements of L are always 1, so no need to store them $\Rightarrow$ work on storage space for A and overwrite it.

LU factorization

Solve $Ax = b$

- Pass 1: factorize $A = LU$ such that L, U are lower/upper triangular
- Pass 2: obtain $x = U^{-1}L^{-1}b$ by solution of lower/upper triangular systems
  - 1. solve $L\tilde{x} = b$
  - 2. solve $Ux = \tilde{x}$
- We never calculate A^{-1} as this would be more expensive

Problem example

- Consider $\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1+\epsilon \\ 1 \end{pmatrix}$
- Solution: $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
- Machine arithmetic: Let $\epsilon << 1$ such that $1 + \epsilon = 1$.
- Equation system in machine arithmetic:
  $1 \cdot \epsilon + 1 \cdot 1 = 1 + \epsilon$

  $1 \cdot 1 + 1 \cdot 1 = 2$
- Still fulfilled!

Problem example II: Gaussian elimination

- Ordinary elimination: $\text{equation}_2 \leftarrow \text{equation}_2 - \frac{1}{\epsilon}\,\text{equation}_1$

$$\begin{pmatrix} \epsilon & 1 \\ 0 & 1-\frac{1}{\epsilon} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1+\epsilon \\ 2-\frac{1+\epsilon}{\epsilon} \end{pmatrix}$$

- In exact arithmetic:

$$\Rightarrow x_2 = \frac{1-\frac{1}{\epsilon}}{1-\frac{1}{\epsilon}} = 1 \Rightarrow x_1 = \frac{1+\epsilon-x_2}{\epsilon} = 1$$

- In floating point arithmetic: $1+\epsilon = 1$, $1-\frac{1}{\epsilon} = -\frac{1}{\epsilon}$, $2-\frac{1}{\epsilon} = -\frac{1}{\epsilon}$:

$$\begin{pmatrix} \epsilon & 1 \\ 0 & -\frac{1}{\epsilon} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ -\frac{1}{\epsilon} \end{pmatrix}$$

$\Rightarrow x_2 = 1 \Rightarrow \epsilon x_1 + 1 = 1 \Rightarrow x_1 = 0$

Problem example III: Partial Pivoting

- Before elimination step, look at the element with largest absolute value in current column and put the corresponding row "on top" as the "pivot"
- This prevents near zero divisions and increases stability

$$\begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1+\epsilon \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 1 \\ 0 & 1-\epsilon \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1-\epsilon \end{pmatrix}$$

- Independent of ϵ:

$$x_2 = \frac{1-\epsilon}{1-\epsilon} = 1, \qquad x_1 = 2 - x_2 = 1$$

- Instead of A, factorize PA: $PA = LU$, where P is a permutation matrix which can be encoded using an integer vector

Gaussian elimination and LU factorization

- Full pivoting: in addition to row exchanges, perform column exchanges to ensure even larger pivots. Seldomly used in practice.
- Gaussian elimination with partial pivoting is the "working horse" for direct solution methods
- Complexity of LU-Factorization: $O(N^3)$, some theoretically better algorithms are known with e.g. $O(N^{2.736})$
- Complexity of triangular solve: $O(N^2)$
  $\Rightarrow$ overall complexity of linear system solution is $O(N^3)$

Cholesky factorization

- $A = LL^T$ for symmetric, positive definite matrices

BLAS, LAPACK

- BLAS: Basic Linear Algebra Subprograms `http://www.netlib.org/blas/`
  - Level 1 - vector-vector: $\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$
  - Level 2 - matrix-vector: $\mathbf{y} \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$
  - Level 3 - matrix-matrix: $C \leftarrow \alpha AB + \beta C$
- LAPACK: Linear Algebra PACKage `http://www.netlib.org/lapack/`
  - Linear system solution, eigenvalue calculation etc.
  - `dgetrf`: LU factorization
  - `dgetrs`: LU solve
- Used in overwhelming number of codes (e.g. matlab, scipy etc.). Also, C++ matrix libraries use these routines. Unless there is special need, they should be used.
- Reference implementations in Fortran, but many more implementations available which carefully work with cache lines etc.

Matrices from PDEs

- So far, we assumed that matrices are stored in a two-dimensional, $n \times n$ array of numbers
- This kind of matrices are also called *dense* matrices
- As we will see, matrices from PDEs (can) have a number of structural properties one can take advantage of when storing a matrix and solving the linear system

1D heat conduction

- v_L, v_R: ambient temperatures, α: heat transfer coefficient
- Second order boundary value problem in $\Omega = [0, 1]$:

$$\begin{aligned} -u''(x) &= f(x) && \text{in}\Omega \\ -u'(0) + \alpha(u(0) - v_L) &= 0 \\ u'(1) + \alpha(u(1) - v_R) &= 0 \end{aligned}$$

- Let $h = \frac{1}{n-1}$, $x_i = x_0 + (i-1)h$ $i = 1 \ldots n$ be discretization points, let u_i approximations for $u(x_i)$ and $f_i = f(x_i)$
- Finite difference approximation:

$$\begin{aligned} -u'(0) + \alpha(u(0) - v_L) &\approx \frac{1}{h}(u_0 - u_1) + \alpha(u_0 - v_L) \\ -u''(x_i) - f(x_i) &\approx \frac{1}{h^2}(-u_{i+1} + 2u_i - u_{i-1}) - f_i \quad (i = 2 \ldots n-1) \\ u'(1) + \alpha(u(1) - v_R) &\approx \frac{1}{h}(u_n - u_{n-1}) + \alpha(u_n - v_R) \end{aligned}$$

1D heat conduction: discretization matrix

- equations $2 \ldots n-1$ multiplied by h
- only nonzero entries written

$$\begin{pmatrix} \alpha + \frac{1}{h} & -\frac{1}{h} & & & & \\ -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & & \\ & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & \\ & & & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} \\ & & & & -\frac{1}{h} & \frac{1}{h} + \alpha \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \\ u_N \end{pmatrix} = \begin{pmatrix} \alpha v_L \\ hf_2 \\ hf_3 \\ \vdots \\ hf_{N-2} \\ hf_{N-1} \\ \alpha v_R \end{pmatrix}$$

- Each row contains ≤ 3 elements
- Only $3n-2$ of n^2 elements are non-zero

General tridiagonal matrix

$$\begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ & & & a_n & b_n \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{pmatrix}$$

- To store matrix, it is sufficient to store only nonzero elements in three one-dimensional arrays for a_i, b_i, c_i, respectively

Gaussian elimination for tridiagonal systems

Gaussian elimination using arrays a, b, c as matrix storage ?

From what we have seen, this question arises in a quite natural way, and historically, the answer has been given several times

- TDMA (tridiagonal matrix algorithm)
- "Thomas algorithm" (Llewellyn H. Thomas, 1949 (?))
- "Progonka method" (from Russian "run through"; Gelfand, Lokutsievski, 1952, published 1960)

Progonka: derivation

- $a_i u_{i-1} + b_i u_i + c_i u_{i+1} = f_i \quad (i = 1 \dots n)$; $a_1 = 0$, $c_N = 0$
- For $i = 1 \dots n-1$, assume there are coefficients α_i, β_i such that $u_i = \alpha_{i+1} u_{i+1} + \beta_{i+1}$.
- Then, we can express u_{i-1} and u_i via u_{i+1}:
  $(a_i \alpha_i \alpha_{i+1} + b_i \alpha_{i+1} + c_i) u_{i+1} + a_i \alpha_i \beta_{i+1} + a_i \beta_i + b_i \beta_{i+1} - f_i = 0$
- This is true independently of u if

$$\begin{cases} a_i \alpha_i \alpha_{i+1} + b_i \alpha_{i+1} + c_i & = 0 \\ a_i \alpha_i \beta_{i+1} + a_i \beta_i + b_i \beta_{i+1} - f_i & = 0 \end{cases}$$

- or for $i = 1 \dots n-1$:

$$\begin{cases} \alpha_{i+1} & = -\frac{c_i}{a_i \alpha_i + b_i} \\ \beta_{i+1} & = \frac{f_i - a_i \beta_i}{a_i \alpha_i + b_i} \end{cases}$$

Progonka: realization

- Forward sweep:

$$\begin{cases} \alpha_2 &= -\frac{c_1}{b_1} \\ \beta_2 &= \frac{f_i}{b_1} \end{cases}$$

for $i = 2 \ldots n-1$

$$\begin{cases} \alpha_{i+1} &= -\frac{c_i}{a_i \alpha_i + b_i} \\ \beta_{i+1} &= \frac{f_i - a_i \beta_i}{a_i \alpha_i + b_i} \end{cases}$$

- Backward sweep:

$$u_n = \frac{f_n - a_n \beta_n}{a_n \alpha_n + b_n}$$

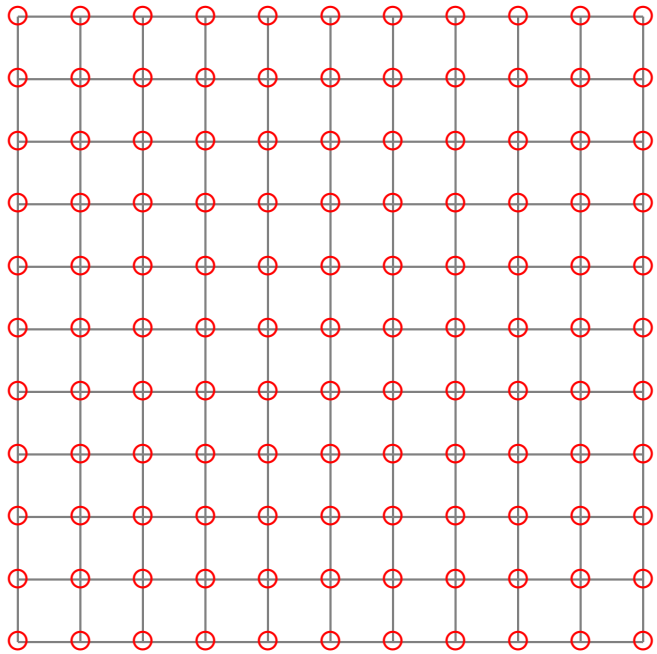
for $n-1 \ldots 1$:

$$u_i = \alpha_{i+1} u_{i+1} + \beta_{i+1}$$

Progonka: properties

- n unknowns, one forward sweep, one backward sweep
  $\Rightarrow O(n)$ operations vs. $O(n^3)$ for algorithm using full matrix
- No pivoting $\Rightarrow$ stability issues
  - Stability for diagonally dominant matrices ($|b_i| > |a_i| + |c_i|$)
  - Stability for symmetric positive definite matrices

2D finite difference grid



- Each discretization point has not more then 4 neighbours
- Matrix can be stored in five diagonals,
  LU factorization not anymore $\equiv$ "fill-in"
- Certain iterative methods can take advantage of the regular and hierachical structure (multigrid) and are able to solve system in $O(n)$ operations
- Another possibility: fast Fourier transform with $O(n \log n)$ operations

Sparse matrices

- Tridiagonal and five-diagonal matrices can be seen as special cases of *sparse matrices*
- Generally they occur in finite element, finite difference and finite volume discretizations of PDEs on structured and unstructured grids
- Definition: Regardless of number of unknowns n, the number of non-zero entries per row remains limited by n_r
- If we find a scheme which allows to store only the non-zero matrix entries, we would need $nn_r = O(n)$ storage locations instead of n^2
- The same would be true for the matrix-vector multiplication if we program it in such a way that we use every nonzero element just once: matrix-vector multiplication would use $O(n)$ instead of $O(n^2)$ operations

Sparse matrix questions

- What is a good storage format for sparse matrices?
- Is there a way to implement Gaussian elimination for general sparse matrices which allows for linear system solution with $O(n)$ operation ?
- Is there a way to implement Gaussian elimination *with pivoting* for general sparse matrices which allows for linear system solution with $O(n)$ operations?
- Is there *any algorithm* for sparse linear system solution with $O(n)$ operations?

Coordinate (triplet) format

- Store all nonzero elements along with their row and column indices
- One real, two integer arrays, length = nnz= number of nonzero elements

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

AA	12.	9.	7.	5.	1.	2.	11.	3.	6.	4.	8.	10.
JR	5	3	3	2	1	1	4	2	3	2	3	4
JC	5	5	3	4	1	4	4	1	1	2	4	3

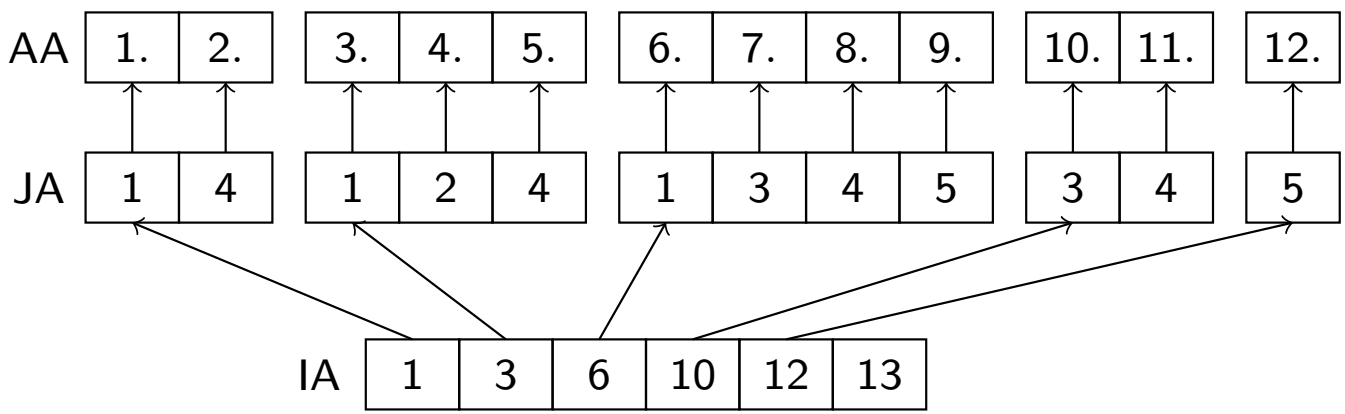
Y.Saad, Iterative Methods, p.92

Compressed Row Storage (CRS) format

(aka Compressed Sparse Row (CSR) or IA-JA etc.)

- real array `AA`, length nnz, containing all nonzero elements row by row
- integer array `JA`, length nnz, containing the column indices of the elements of `AA`
- integer array `IA`, length n+1, containing the start indizes of each row in the arrays `IA` and `JA` and `IA(n+1)=nnz+1`

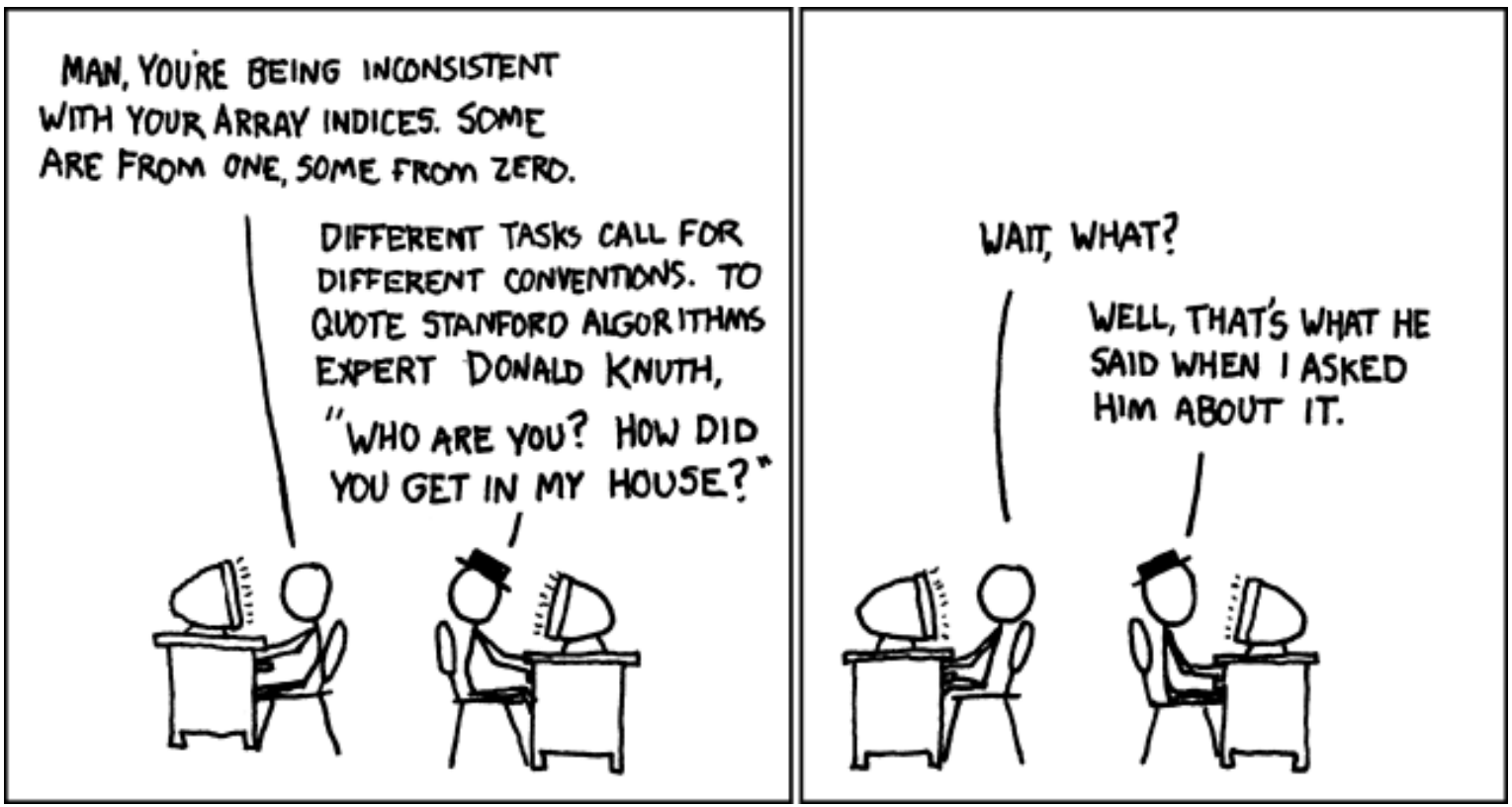
$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$



- Used in most sparse matrix solver packages
- CSC (Compressed Column Storage) uses similar principle but stores the matrix column-wise.

The big schism

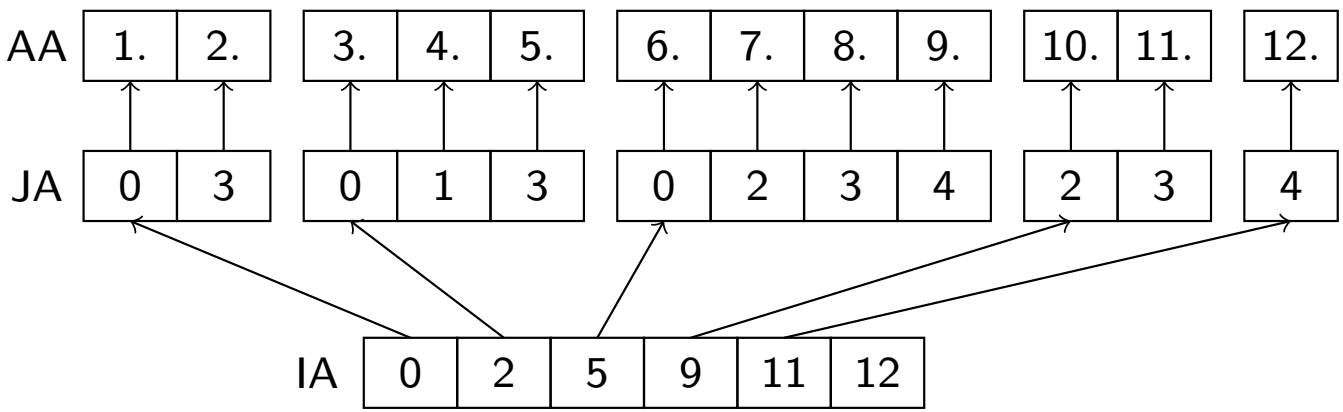
- Should array indices count from zero or from one ?
- Fortran, Matlab, Julia count from one
- C/C++, python count from zero
- It matters when passing index arrays to sparse matrix packages



http://xkcd.com/1739/

CRS format with zero array offsets ...

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0. & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$



- some package APIs provide the possibility to specify array offset
- index shift is not very expensive compared to the rest of the work

Sparse direct solvers

- Sparse direct solvers implement Gaussian elimination with different pivoting strategies
  - UMFPACK
  - Pardiso (omp + MPI parallel)
  - SuperLU (omp parallel)
  - MUMPS (MPI parallel)
  - Pastix
- Quite efficient for 1D/2D problems
- Essentially they implement the LU factorization algorithm
- They suffer from *fill-in*, especially for 3D problems: Let $nz(M)$ be the number of nonzero elements of a matrix A. Let $A = LU$ be an LU-Factorization. Then, as a rule, $nz(L + U) > NZ(A)$.
  - $\Rightarrow$ increased memory usage to store L,U
  - $\Rightarrow$ high operation cout

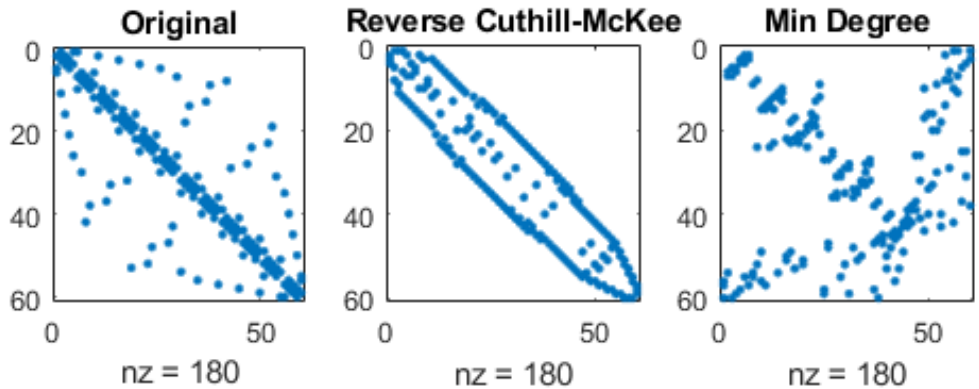
Sparse direct solvers: solution steps (Saad Ch. 3.6)

1. Pre-ordering
   - Decrease amount of non-zero elements generated by fill-in by re-ordering of the matrix
   - Several, graph theory based heuristic algorithms exist
2. Symbolic factorization
   - If pivoting is ignored, the indices of the non-zero elements are calculated and stored
   - Most expensive step wrt. computation time
3. Numerical factorization
   - Calculation of the numerical values of the nonzero entries
   - Moderately expensive, once the symbolic factors are available
4. Upper/lower triangular system solution
   - Fairly quick in comparison to the other steps

- Separation of steps 2 and 3 allows to save computational costs for problems where the sparsity structure remains unchanged, e.g. time dependent problems on fixed computational grids
- With pivoting, steps 2 and 3 have to be performed together
- Instead of pivoting, *iterative refinement* may be used in order to maintain accuracy of the solution

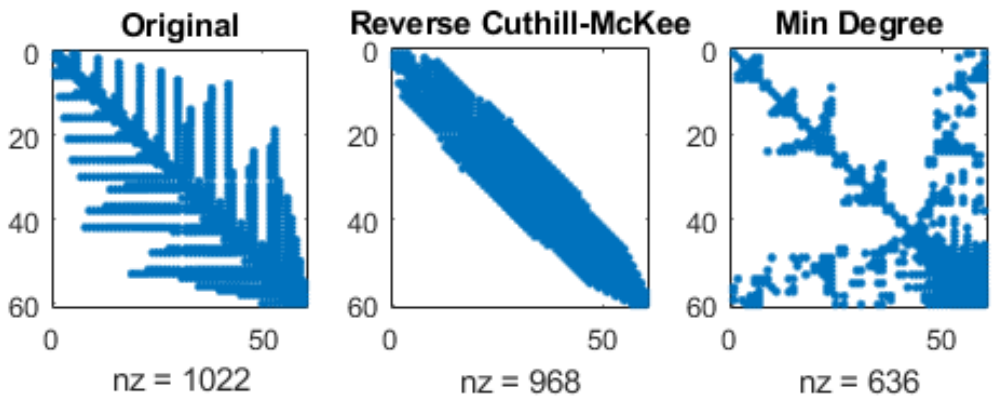
Sparse direct solvers: influence of reordering

- Sparsity patterns for original matrix with three different orderings of unknowns – number of nonzero elements (of course) independent of ordering:



https://de.mathworks.com

- Sparsity patterns for corresponding LU factorizations – number of nonzero elements depend original ordering!



https://de.mathworks.com

Sparse direct solvers: Complexity

- Complexity estimates depend on storage scheme, reordering etc.
- Sparse matrix - vector multiplication has complexity $O(N)$
- Some estimates can be given for from graph theory for discretizations of heat equation with $N = n^d$ unknowns on close to cubic grids in space dimension d
  - sparse LU factorization:

d	work	storage
1	$O(N) \mid O(n)$	$O(N) \mid O(n)$
2	$O(N^{\frac{3}{2}}) \mid O(n^3)$	$O(N \log N) \mid O(n^2 \log n)$
3	$O(N^2) \mid O(n^6)$	$O(N^{\frac{4}{3}}) \mid O(n^4)$

  - triangular solve: work dominated by storage complexity

d	work
1	$O(N) \mid O(n)$
2	$O(N \log N) \mid O(n^2 \log n)$
3	$O(N^{\frac{4}{3}}) \mid O(n^4)$

Source: J. Poulson, PhD thesis, http://hdl.handle.net/2152/ETD-UT-2012-12-6622

Iterative methods

Elements of iterative methods (Saad Ch.4)

Let $V = \mathbb{R}^n$ be equipped with the inner product $(\cdot, \cdot)$, let A be an $n \times n$ nonsingular matrix.

Solve $Au = b$ iteratively

- Preconditioner: a matrix $M \approx A$ "approximating" the matrix A but with the property that the system $Mv = f$ is easy to solve
- Iteration scheme: algorithmic sequence using M and A which updates the solution step by step

Simple iteration with preconditioning

Idea: $A\hat{u} = b \Rightarrow$

$$\hat{u} = \hat{u} - M^{-1}(A\hat{u} - b)$$

$\Rightarrow$ iterative scheme

$$u_{k+1} = u_k - M^{-1}(Au_k - b) \quad (k = 0, 1 \dots)$$

1. Choose initial value u_0, tolerance ε, set $k = 0$
2. Calculate *residuum* $r_k = Au_k - b$
3. Test convergence: if $||r_k|| < \varepsilon$ set $u = u_k$, finish
4. Calculate *update*: solve $Mv_k = r_k$
5. Update solution: $u_{k+1} = u_k - v_k$, set $k = i + 1$, repeat with step 2.

The Jacobi method

- Let $A = D - E - F$, where D: main diagonal, E: negative lower triangular part F: negative upper triangular part
- Preconditioner: $M = D$, where D is the main diagonal of $A \Rightarrow$

$$u_{k+1,i} = u_{k,i} - \frac{1}{a_{ii}} \left(\sum_{j=1 \ldots n} a_{ij} u_{k,j} - b_i \right) \quad (i = 1 \ldots n)$$

- Equivalent to the succesive (row by row) solution of

$$a_{ii} u_{k+1,i} + \sum_{j=1 \ldots n, j \neq i} a_{ij} u_{k,j} = b_i \quad (i = 1 \ldots n)$$

- Already calculated results not taken into account
- Alternative formulation with $A = M - N$:

$$\begin{aligned} u_{k+1} &= D^{-1}(E + F)u_k + D^{-1}b \\ &= M^{-1}Nu_k + M^{-1}b \end{aligned}$$

- Variable ordering does not matter

The Gauss-Seidel method

- Solve for main diagonal element row by row
- Take already calculated results into account

$$a_{ii}u_{k+1,i} + \sum_{j<i} a_{ij}u_{k+1,j} + \sum_{j>i} a_{ij}u_{k,j} = b_i \qquad (i = 1 \dots n)$$

$$(D - E)u_{k+1} - Fu_k = b$$

- May be it is faster
- Variable order probably matters
- Preconditioners: forward $M = D - E$, backward: $M = D - F$
- Splitting formulation: $A = M - N$
  forward: $N = F$, backward: $M = E$
- Forward case:

$$\begin{aligned} u_{k+1} &= (D - E)^{-1}Fu_k + (D - E)^{-1}b \\ &= M^{-1}Nu_k + M^{-1}b \end{aligned}$$

Gauss an Gerling I

[6.]

[Über Stationsausgleichungen.]

GAUSS an GERLING. Göttingen, 26. December 1823.

Mein Brief ist zu spät zur Post gekommen und mir zurückgebracht. Ich erbreche ihn daher wieder, um noch die praktische Anweisung zur Elimination beizufügen. Freilich gibt es dabei vielfache kleine Localvortheile, die sich nur ex usu lernen lassen.

Ich nehme Ihre Messungen auf Orber-Reisig zum Beispiel[*].

Ich mache zuerst

$$[\text{Richtung nach}]\ 1 = 0,$$

nachher aus 1.3

$$3 = 77^0\,57'\,53''{,}107$$

(ich ziehe dies vor, weil 1.3 mehr Gewicht hat als 1.2);

dann aus

$$\left.\begin{array}{c|c|l} 13 & 1.2 & 2 = 26^0\,44'\ 7''{,}423 \\ 50 & 2.3 & 2 = \quad\quad 6''{,}507 \end{array}\right\} 2 = 26^0\,44'\ 6''{,}696;$$

endlich aus

$$\left.\begin{array}{c|c|l} 26 & 1.4 & 4 = 136^0\,21'\,13''{,}481 \\ 6 & 2.4 & 4 = \quad\quad 8{,}529 \\ 78 & 3.4 & 4 = \quad\quad 11{,}268 \end{array}\right\} 4 = 136^0\,21'\,11''{,}641.$$

Ich suche, um die Annäherung erst noch zu vergrössern, aus

[*] Die von GERLING mitgetheilten Winkelmessungen waren (nach einem in GAUSS' Nachlass befindlichen Blatte), wenn 1 Berger Warte, 2 Johannisberg, 3 Taufstein und 4 Milseburg bezeichnet:

Rep.	Winkel
13	1.2 = 26° 44′ 7″,423
28	1.3 = 77 57 53, 107
26	1.4 = 136 21 13, 481
50	2.3 = 51 13 46, 600
6	2.4 = 109 37 1, 833
78	3.4 = 58 23 18, 161.]

$$\left.\begin{array}{c|c|l} 13 & 1.2 & 1 = -0''{,}727 \\ 28 & 1.3 & 1 = \quad 0 \\ 26 & 1.4 & 1 = -1{,}840 \end{array}\right\} 1 = -0''{,}855.$$

Da jede gemeinschaftliche Änderung aller Richtungen erlaubt ist, so lange es nur die relative Lage gilt, so ändere ich alle vier um $+0''{,}855$ und setze

$$\begin{aligned} 1 &= \quad 0^0\ 0'\ 0''{,}000 + a \\ 2 &= \quad 26\ 44\ \ 7{,}551 + b \\ 3 &= \quad 77\ 57\ 53{,}962 + c \\ 4 &= 136\ 21\ 12{,}496 + d. \end{aligned}$$

Es ist beim indirecten Verfahren sehr vortheilhaft, jeder Richtung eine Veränderung beizulegen. Sie können sich davon leicht überzeugen, wenn Sie dasselbe Beispiel ohne diesen Kunstgriff durchrechnen, wo Sie überdies die grosse Bequemlichkeit, an der Summe der absoluten Glieder $= 0$ immer eine Controlle zu haben, verlieren. Jetzt formire ich die vier Bedingungsgleichungen und zwar nach diesem Schema (bei eigener Anwendung und wenn die Glieder zahlreicher sind, trenne ich wohl die positiven und negativen Glieder), [wobei die Constanten in Einheiten der dritten Decimalstelle angesetzt sind:]

$ab - 1664$	$ba + 1664$	$ca + 23940$	$da - 25610$
$ac - 23940$	$bc + 9450$	$cb - 9450$	$db + 18672$
$ad + 25610$	$bd - 18672$	$cd - 29094$	$dc + 29094.$

Die Bedingungsgleichungen sind also:

$$\begin{aligned} 0 &= + \quad 6 + 67a - 13b - 28c - 26d \\ 0 &= - \ 7558 - 13a + 69b - 50c - 6d \\ 0 &= -14604 - 28a - 50b + 156c - 78d \\ 0 &= +22156 - 26a - 6b - 78c + 110d; \end{aligned}$$

$$\text{Summe} = 0.$$

Um nun indirect zu eliminiren, bemerke ich, dass, wenn 3 der Grössen a, b, c, d gleich 0 gesetzt werden, die vierte den grössten Werth bekommt, wenn d dafür gewählt wird. Natürlich muss jede Grösse aus ihrer eigenen Gleichung, also d aus der vierten, bestimmt werden. Ich setze also $d = -201$

Gauss an Gerling II

und substituire diesen Werth. Die absoluten Theile werden dann: $+5232$, -6352, $+1074$, $+46$; das Übrige bleibt dasselbe.

Jetzt lasse ich b an die Reihe kommen, finde $b = +92$, substituire und finde die absoluten Theile: $+4036$, -4, -3526, -506. So fahre ich fort, bis nichts mehr zu corrigiren ist. Von dieser ganzen Rechnung schreibe ich aber in der Wirklichkeit bloss folgendes Schema:

	$d=-201$	$b=+92$	$a=-60$	$c=+12$	$a=+5$	$b=-2$	$a=-1$
+ 6	+5232	+4036	+ 16	−320	+ 15	+41	−26
− 7558	−6352	− 4	+ 776	+176	+111	−27	−14
−14604	+1074	−3526	−1846	+ 26	−114	−14	+14
+22156	+ 46	− 506	+1054	+118	− 12	0	+26.

Insofern ich die Rechnung nur auf das nächste 2000^{tel} [der] Secunde führe, sehe ich, dass jetzt nichts mehr zu corrigiren ist. Ich sammle daher

$a=-60$	$b=+92$	$c=+12$	$d=-201$
$+5$	-2		
-1			
-56	$+90$	$+12$	-201

und füge die Correctio communis $+56$ bei, wodurch wird:

$$a = 0 \quad b = +146 \quad c = +68 \quad d = -145,$$

also die Werthe [der Richtungen]

1	$0^0\ 0'\ 0''{,}000$
2	$26\ 44\ 7{,}697$
3	$77\ 57\ 54{,}030$
4	$136\ 21\ 12{,}351.$

Fast jeden Abend mache ich eine neue Auflage des Tableaus, wo immer leicht nachzuhelfen ist. Bei der Einförmigkeit des Messungsgeschäfts gibt dies immer eine angenehme Unterhaltung; man sieht dann auch immer gleich, ob etwas zweifelhaftes eingeschlichen ist, was noch wünschenswerth bleibt, etc. Ich empfehle Ihnen diesen Modus zur Nachahmung. Schwerlich werden Sie je wieder direct eliminiren, wenigstens nicht, wenn Sie mehr als 2 Unbekannte

haben. Das indirecte Verfahren lässt sich halb im Schlafe ausführen, oder man kann während desselben an andere Dinge denken.

. . . . . .

GAUSS an SCHUMACHER. Göttingen, 22. December 1827.

Die Einheit in meinem Coordinatenverzeichnisse ist 443,307885 [Pariser] Linien; der Logarithm zur Reduction auf Toisen

$$= 9{,}710\,1917.$$

Inzwischen gründet sich das absolute nur auf Ihre Basis, oder vielmehr auf die von CAROC mir angegebene Entfernung zwischen Hamburg und Hohenhorn, log = 4,141 1930, wofür ich also genommen habe: 4,431 0013. Sollte nach der Definitivbestimmung Ihrer Stangen Ihre Basis, und damit die obige Angabe der Entfernung Hamburg-Hohenhorn, eine Veränderung erleiden, so werden in demselben Verhältnisse auch alle meine Coordinaten zu verändern sein.

In der Form der Behandlung ist ein wichtiges Moment, dass von jedem Beobachtungsplatz ein Tableau aufgestellt wird, worin alle Azimuthe (in meinem Sinn) geordnet enthalten sind. Man hat so zum bequemsten Gebrauch fertig alles, was man von den Beobachtungen nöthig hat, so dass man nur ausnahmsweise, um diesen oder jenen Zweifel zu lösen, zu den Originalprotocollen recurrirt. .... Ist der Standpunkt von dem Zielpunkt verschieden, so reducire ich keineswegs die Beobachtungen auf letztern (Centrirung), da sie ohne diese Reduction ebenso bequem gebraucht werden können (insofern nemlich von vielen Schnitten untergeordneter Punkte die Rede ist, die nicht wieder Standpunkte sind).

Die Bildung eines solchen Tableaus beruht nun wieder auf mehrern Momenten, wozu eine Anweisung nur auf mehrere Briefe vertheilt werden kann, daher Sie vielleicht wohl thun, dieses Tableau erst selbst gleichsam zu studiren und mit den Beobachtungen zusammenzuhalten, damit Sie mir beson-

SOR and SSOR

- SOR: Successive overrelaxation: solve $\omega A = \omega B$ and use splitting

$$\omega A = (D - \omega E) - (\omega F + (1 - \omega D))$$
$$M = \frac{1}{\omega}(D - \omega E)$$

leading to

$$(D - \omega E)u_{k+1} = (\omega F + (1 - \omega D))u_k + \omega b$$

- SSOR: Symmetric successive overrelaxation

$$(D - \omega E)u_{k+\frac{1}{2}} = (\omega F + (1 - \omega D))u_k + \omega b$$
$$(D - \omega F)u_{k+1} = (\omega E + (1 - \omega D))u_{k+\frac{1}{2}} + \omega b$$

- Preconditioner:

$$M = \frac{1}{\omega(2 - \omega)}(D - \omega E)D^{-1}(D - \omega F)$$

- Gauss-Seidel and symmetric Gauss-Seidel are special cases for $\omega = 1$.

Block methods

- Jacobi, Gauss-Seidel, (S)SOR methods can as well be used block-wise, based on a partition of the system matrix into larger blocks,
- The blocks on the diagonal should be square matrices, and invertible
- Interesting variant for systems of partial differential equations, where multiple species interact with each other

Convergence

- Let $\hat{u}$ be the solution of $Au = b$.
- Let $e_k = u_j - \hat{u}$ be the error of the k-th iteration step

$$\begin{aligned} u_{k+1} &= u_k - M^{-1}(Au_k - b) \\ &= (I - M^{-1}A)u_k + M^{-1}b \\ u_{k+1} - \hat{u} &= u_k - \hat{u} - M^{-1}(Au_k - A\hat{u}) \\ &= (I - M^{-1}A)(u_k - \hat{u}) \\ &= (I - M^{-1}A)^k(u_0 - \hat{u}) \end{aligned}$$

  resulting in

$$e_{k+1} = (I - M^{-1}A)^k e_0$$

- So when does $(I - M^{-1}A)^k$ converge to zero for $k \to \infty$?

Jordan canonical form of a matrix A

- λ_i $(i = 1 \ldots p)$: eigenvalues of A
- $\sigma(A) = \{\lambda_1 \ldots \lambda_p\}$: spectrum of A
- μ_i: algebraic multiplicity of λ_i:
  multiplicity as zero of the characteristic polynomial $\det(A - \lambda I)$
- γ_i geometric multiplicity of λ_i: dimension of $\mathrm{K}er(A - \lambda I)$
- l_i: index of the eigenvalue: the smallest integer for which
  $\mathrm{K}er(A - \lambda I)^{l_i+1} = \mathrm{K}er(A - \lambda I)^{l_i}$
- $l_i \leq \mu_i$

Theorem (Saad, Th. 1.8) Matrix A can be transformed to a block diagonal matrix consisting of p diagonal blocks, each associated with a distinct eigenvalue λ_i.

- Each of these diagonal blocks has itself a block diagonal structure consisting of γ_i *Jordan blocks*
- Each of the Jordan blocks is an upper bidiagonal matrix of size not exceeding l_i with λ_i on the diagonal and 1 on the first upper diagonal.

Jordan canonical form of a matrix II

$$X^{-1}AX = J = \begin{pmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_p \end{pmatrix}$$

$$J_i = \begin{pmatrix} J_{i,1} & & & \\ & J_{i,2} & & \\ & & \ddots & \\ & & & J_{i,\gamma_i} \end{pmatrix}$$

$$J_{i,k} = \begin{pmatrix} \lambda_i & 1 & & \\ & \lambda_i & 1 & \\ & & \ddots & 1 \\ & & & \lambda_i \end{pmatrix}$$

Each $J_{i,k}$ is of size l_i and corresponds to a different eigenvector of A.

Spectral radius and convergence

Definition The spectral radius $\rho(A)$ is the largest absolute value of any eigenvalue of A: $\rho(A) = \max_{\lambda \in \sigma(A)} |\lambda|$.

Theorem (Saad, Th. 1.10) $\lim_{k \to \infty} A^k = 0 \Leftrightarrow \rho(A) < 1$.

Proof, $\Rightarrow$: Let u_i be a unit eigenvector associated with an eigenvalue λ_i. Then

$$
\begin{aligned}
A u_i &= \lambda_i u_i \\
A^2 u_i &= \lambda_i A_i u_i = \lambda^2 u_i \\
&\vdots \\
A^k u_i &= \lambda^k u_i \\
\text{therefore} \quad ||A^k u_i||_2 &= |\lambda^k| \\
\text{and} \quad \lim_{k \to \infty} |\lambda^k| &= 0
\end{aligned}
$$

so we must have $\rho(A) < 1$

Spectral radius and convergence II

Proof, $\Leftarrow$: Jordan form $X^{-1}AX = J$. Then $X^{-1}A^kX = J^k$.
Sufficient to regard Jordan block $J_i = \lambda_i I + E_i$ where $|\lambda_i| < 1$ and $E_i^{l_i} = 0$.
Let $k \geq l_i$. Then

$$J_i^k = \sum_{j=0}^{l_i-1} \binom{k}{j} \lambda^{k-j} E_i^j$$

$$||J_i||^k \leq \sum_{j=0}^{l_i-1} \binom{k}{j} |\lambda|^{k-j} ||E_i||^j$$

One has $\binom{k}{j} = \frac{k!}{j!(k-j)!} = \sum_{i=0}^{j} \begin{bmatrix} j \\ i \end{bmatrix} \frac{k^i}{j!}$ is a polynomial of degree j in k
where the Stirling numbers of the first kind are given by
$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1, \quad \begin{bmatrix} j \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ j \end{bmatrix} = 0, \quad \begin{bmatrix} j+1 \\ i \end{bmatrix} = j \begin{bmatrix} j \\ i \end{bmatrix} + \begin{bmatrix} j \\ i-1 \end{bmatrix}$.
Thus, $\binom{k}{j} |\lambda|^{k-j} \to 0$ $(k \to \infty)$ as exponential decay beats polynomial growth
$\square$.

Corollary from proof

Theorem (Saad, Th. 1.12)

$$\lim_{k \to \infty} ||A^k||^{\frac{1}{k}} = \rho(A)$$

□

Back to iterative methods

Sufficient condition for convergence: $\rho(I - M^{-1}A) < 1$.

Convergence rate

Assume λ with $|\lambda| = \rho(I - M^{-1}A) < 1$ is the largest eigenvalue and has a single Jordan block of size l. Then the convergence rate is dominated by this Jordan block, and therein by the term with the lowest possible power in λ which due to $E^l = 0$ is

$$\lambda^{k-l+1} \binom{k}{l-1} E^{l-1}$$

$$||(I - M^{-1}A)^k(u_0 - \hat{u})|| = O\left(|\lambda^{k-l+1}| \binom{k}{l-1}\right)$$

and the "worst case" convergence factor ρ equals the spectral radius:

$$\begin{aligned}
\rho &= \lim_{k\to\infty} \left(\max_{u_0} \frac{||(I - M^{-1}A)^k(u_0 - \hat{u})||}{||u_0 - \hat{u}||} \right)^{\frac{1}{k}} \\
&= \lim_{k\to\infty} ||(I - M^{-1}A)^k||^{\frac{1}{k}} \\
&= \rho(I - M^{-1}A)
\end{aligned}$$

Depending on u_0, the rate may be faster, though