Computational Fluid Dynamics: Sparse Linear systems

Aleksandar Donev
Courant Institute, NYU
donev@courant.nyu.edu

February 2013

Outline

Banded Matrices

- **Banded matrices** are a very special but common type of sparse matrix, e.g., **tridiagonal matrices**

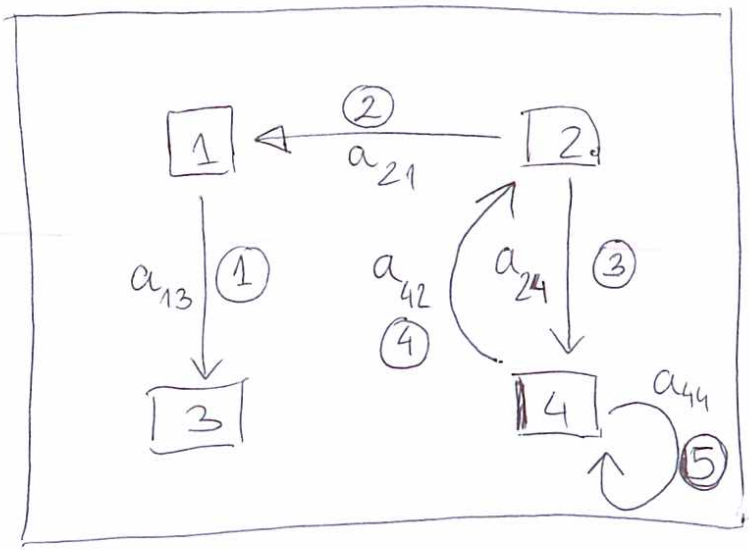
$$\begin{bmatrix} a_1 & c_1 & & \mathbf{0} \\ b_2 & a_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ \mathbf{0} & & b_n & a_n \end{bmatrix}$$

- There exist special techniques for banded matrices that are much faster than the general case, e.g, only $8n$ FLOPS and no additional memory for tridiagonal matrices.
- A general matrix should be considered sparse if it has sufficiently many zeros that exploiting that fact is advantageous:
usually only the case for **large matrices** (what is large?)!

Sparse Matrices

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

Sparse matrix



Directed Graph representation:

→ Nodes (vertices) are variables or equations

→ Arcs (edges) are the non-zeros

Undirected graph for symmetric matrices

Sparse matrices in MATLAB

```
>> A = sparse( [1 2 2 4 4], [3 1 4 2 3], 1:5 )
A =
   (2,1)        2
   (4,2)        4
   (1,3)        1
   (4,3)        5
   (2,4)        3
>> nnz(A)
ans =     5
>> whos A
  A          4x4                   120  double     sparse

>> A = sparse([],[],[],4,4,5); % Pre-allocate memory
>> A(2,1)=2; A(4,2)=4; A(1,3)=1; A(4,3)=5; A(2,4)=3;
```

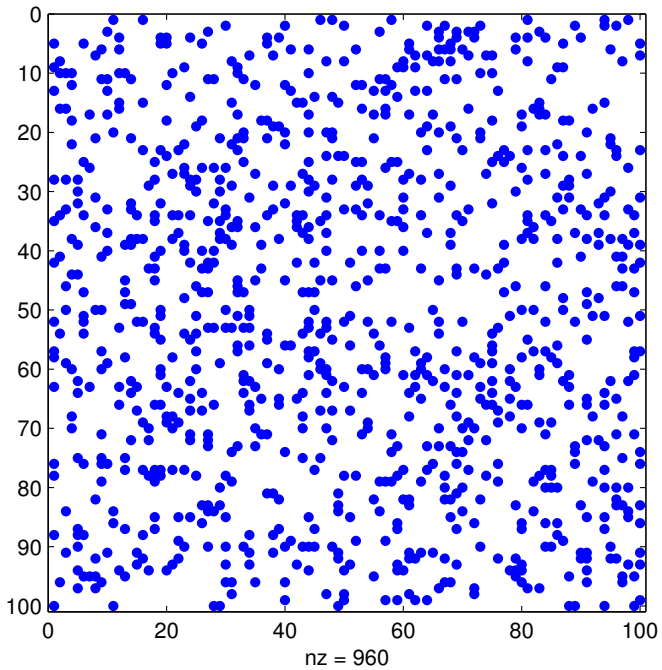
Sparse matrix factorization

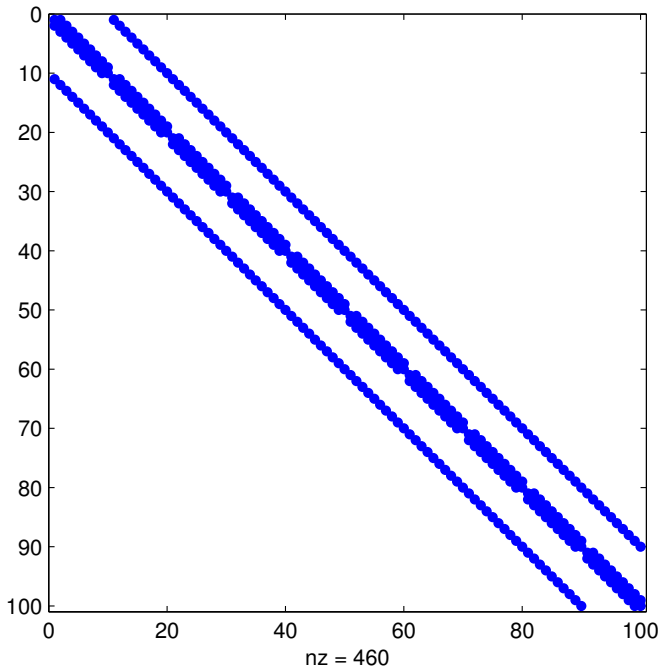
```
>> B=sprand(4,4,0.25); % Density of 25%
>> full(B)
ans =
         0         0         0    0.7655
         0    0.7952         0         0
         0    0.1869         0         0
    0.4898         0         0         0

>> B=sprand(100,100,0.1); spy(B)
>> X=gallery('poisson',10); spy(X)
>> [L,U,P]=lu(B); spy(L)
>> p = symrcm(B); % Symmetric Reverse Cuthill-McKee or
>> PBP=B(p,p); spy(PBP);
>> [L,U,P]=lu(PBP); spy(L);
```

Random matrix **B** and structured matrix **X**

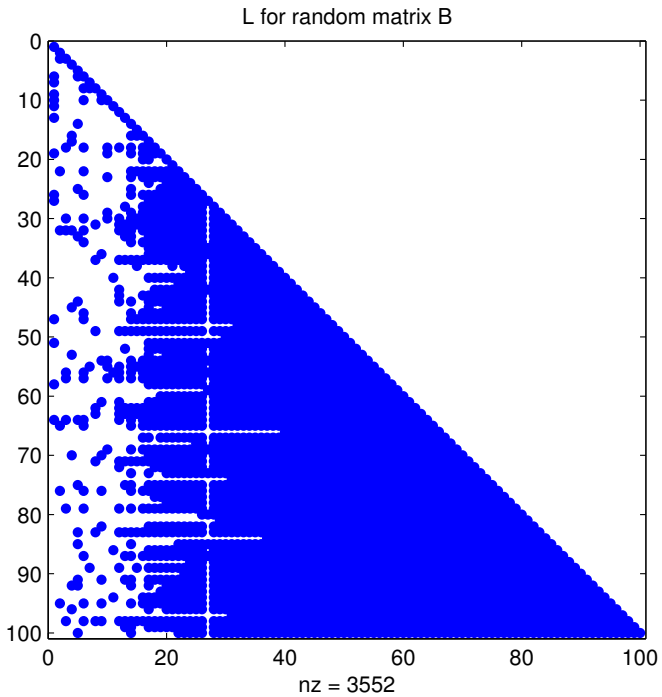
The MATLAB function *spy* shows where the nonzeros are as a plot

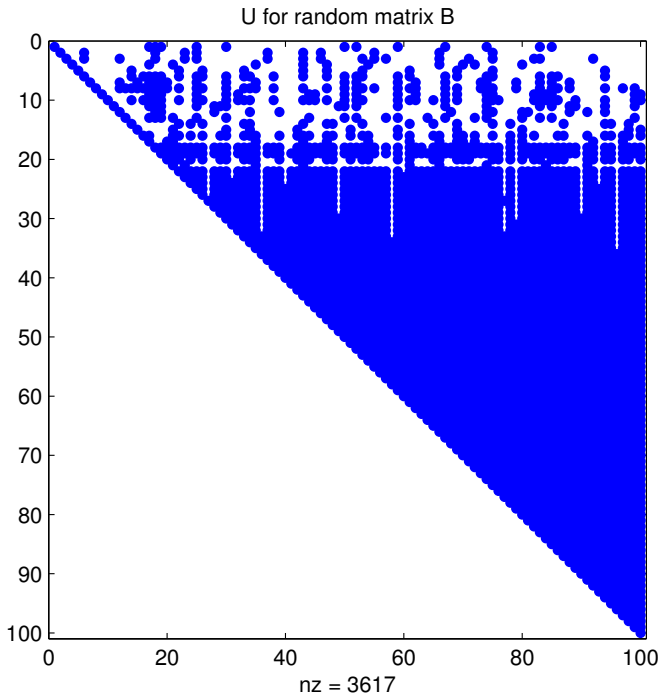




LU factors of random matrix $\mathbf{B}$

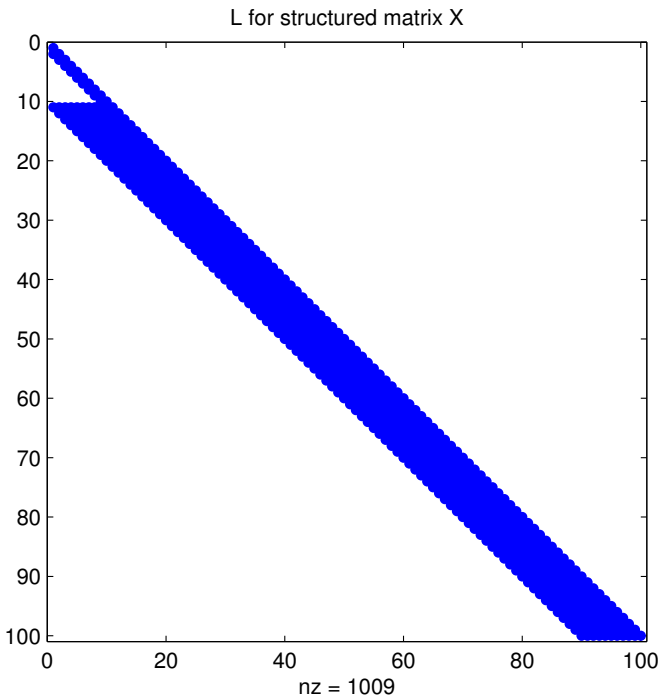
Fill-in (generation of lots of nonzeros) is large for a random sparse matrix

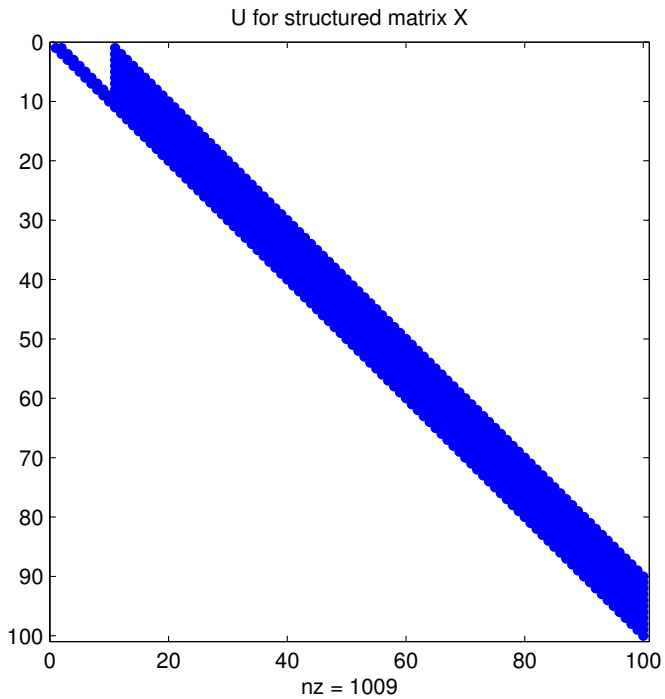




LU factors of structured matrix $\mathbf{X}$

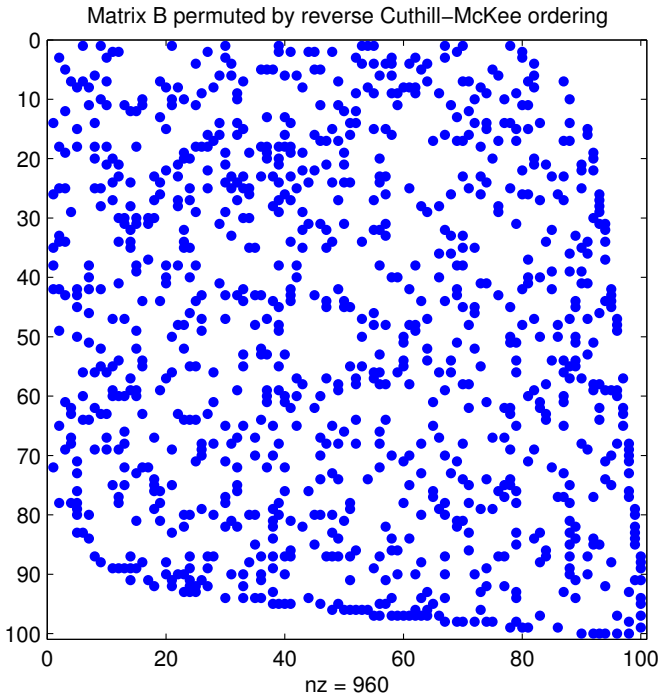
Fill-in is much smaller for the sparse matrix but still non-negligible.

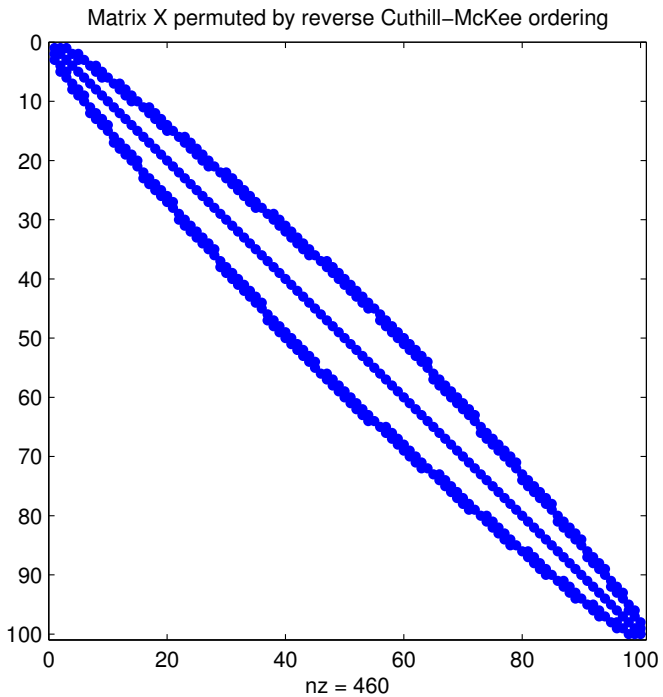




Matrix reordering

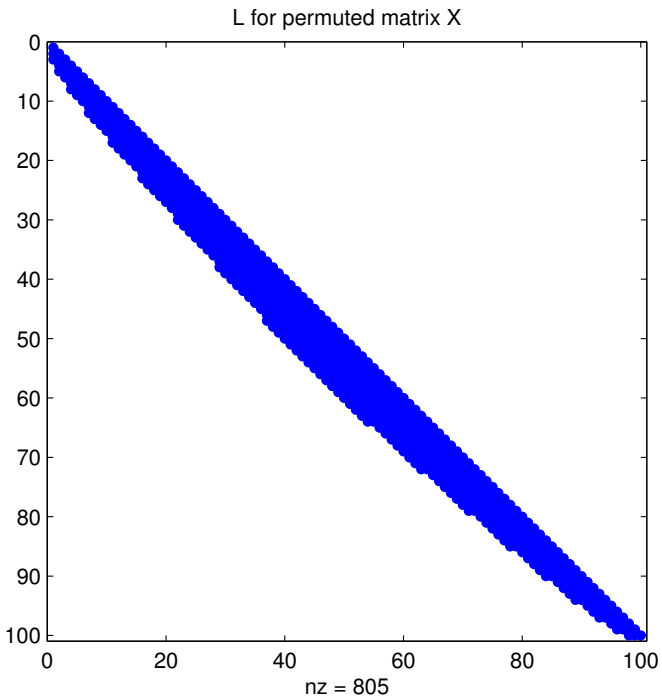
Matrix reordering cannot do much for the random matrix **B**, but it can help for structured ones!

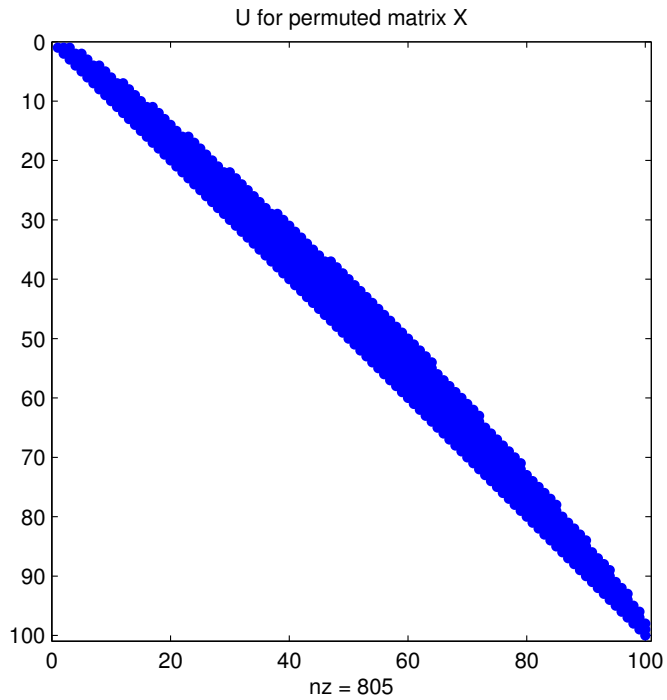




Reducing fill-in by reordering **X**

Fill-in was reduced by about 20% (from 1000 nonzeros to 800) by the reordering for the structured **X**, but does not help much for **B**.
The actual numbers are different for different classes of matrices!





Importance of Sparse Matrix Structure

- Important to remember: While there are general techniques for dealing with sparse matrices that help greatly, it all depends on the structure (origin) of the matrix.
- Pivoting has a dual, sometimes conflicting goal:
  1. Reduce fill-in, i.e., **improve memory use**: *Still active subject of research!*
  2. Reduce roundoff error, i.e., **improve stability**. Typically some **threshold pivoting** is used only when needed.
- Pivoting for symmetric non-positive definite matrices is trickier: One can permute the diagonal entries only to **preserve symmetry**, but small diagonal entries require special treatment.
- For many sparse matrices **iterative methods** (briefly covered next lecture) are required to large fill-in.

Why iterative methods?

- Direct solvers are great for dense matrices and can be made to avoid roundoff errors to a large degree. They can also be implemented very well on modern machines.
- **Fill-in** is a major problem for certain sparse matrices and leads to extreme memory requirements (e.g., three-d.
- Some matrices appearing in practice are **too large** to even be represented explicitly (e.g., the Google matrix).
- Often linear systems only need to be **solved approximately**, for example, the linear system itself may be a linear approximation to a nonlinear problem.
- Direct solvers are much harder to implement and use on (massively) **parallel computers**.

Stationary Linear Iterative Methods of First Order

- In iterative methods the core computation is **iterative matrix-vector multiplication** starting from an **initial guess** $\mathbf{x}^{(0)}$.
- Prototype is the **linear recursion**:

$$\mathbf{x}^{(k+1)} = \mathbf{B}\mathbf{x}^{(k)} + \mathbf{f},$$

  where $\mathbf{B}$ is an **iteration matrix** somehow related to $\mathbf{A}$.
- For this method to be **consistent**, we must have that the actual solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ is a **stationary point** of the iteration:

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{f} \quad \Rightarrow \quad \mathbf{A}^{-1}\mathbf{b} = \mathbf{B}\mathbf{A}^{-1}\mathbf{b} + \mathbf{f}$$

$$\mathbf{f} = \mathbf{A}^{-1}\mathbf{b} - \mathbf{B}\mathbf{A}^{-1}\mathbf{b} = (\mathbf{I} - \mathbf{B})\,\mathbf{x}$$

- For this method to be **stable**, and thus **convergent**, the error $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}$ must decrease:

$$\mathbf{e}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{x} = \mathbf{B}\mathbf{x}^{(k)} + \mathbf{f} - \mathbf{x} = \mathbf{B}\left(\mathbf{x} + \mathbf{e}^{(k)}\right) + (\mathbf{I} - \mathbf{B})\,\mathbf{x} - \mathbf{x} = \mathbf{B}\mathbf{e}^{(k)}$$

Convergence of simple iterative methods

- We saw that the error propagates from iteration to iteration as

$$\mathbf{e}^{(k)} = \mathbf{B}^k \mathbf{e}^{(0)}.$$

- When does this converge? Taking norms,

$$\left\| \mathbf{e}^{(k)} \right\| \leq \|\mathbf{B}\|^k \left\| \mathbf{e}^{(0)} \right\|$$

  which means that $\|\mathbf{B}\| < 1$ is a **sufficient condition** for convergence.
- More precisely, $\lim_{k\to\infty} \mathbf{e}^{(k)} = \mathbf{0}$ for any $\mathbf{e}^{(0)}$ iff $\mathbf{B}^k \to \mathbf{0}$.
- Theorem: The method converges iff the **spectral radius** of the iteration matrix is less than unity:

$$\rho(\mathbf{B}) < 1.$$

Spectral Radius

- The **spectral radius** $\rho(\mathbf{A})$ of a matrix $\mathbf{A}$ can be thought of as the smallest consistent matrix norm

$$\rho(\mathbf{A}) = \max_{\lambda} |\lambda| \leq \|\mathbf{A}\|$$

- The spectral radius often **determines convergence of iterative schemes** for linear systems and eigenvalues and even methods for solving PDEs because it estimates the asymptotic rate of error propagation:

$$\rho(\mathbf{A}) = \lim_{k \to \infty} \left\|\mathbf{A}^k\right\|^{1/k}$$

Termination

- The iterations of an iterative method can be terminated when:
  1. The **residual** becomes small,
  $$\left\|\mathbf{r}^{(k)}\right\| \leq \varepsilon \|\mathbf{b}\|$$
  This is good for well-conditioned systems.
  2. The solution $\mathbf{x}^{(k)}$ stops changing, i.e., the **increment** becomes small,
  $$[1-\rho(\mathbf{B})]\left\|\mathbf{e}^{(k)}\right\| \leq \left\|\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right\| \leq \varepsilon \|\mathbf{b}\|,$$
  which can be seen to be good if convergence is rapid, $\rho(\mathbf{B}) \ll 1$.
- Usually a careful **combination** of the two strategies is employed along with some **safeguards**.

Fixed-Point Iteration

- A naive but often successful method for solving

$$x = f(x)$$

  is the **fixed-point iteration**

$$x_{n+1} = f(x_n).$$

- In the case of a linear system, consider rewriting $\mathbf{Ax} = \mathbf{b}$ as:

$$\mathbf{x} = (\mathbf{I} - \mathbf{A})\,\mathbf{x} + \mathbf{b}$$

- Fixed-point iteration gives the consistent iterative method

$$\mathbf{x}^{(k+1)} = (\mathbf{I} - \mathbf{A})\,\mathbf{x}^{(k)} + \mathbf{b}$$

Preconditioning

- The above method is consistent but it may not converge or may converge very slowly

$$\mathbf{x}^{(k+1)} = (\mathbf{I} - \mathbf{A})\,\mathbf{x}^{(k)} + \mathbf{b}.$$

- As a way to speed it up, consider having a **good approximate solver**

$$\mathbf{P}^{-1} \approx \mathbf{A}^{-1}$$

called the **preconditioner** ($\mathbf{P}$ is the preconditioning matrix), and transform

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b}$$

- Now apply fixed-point iteration to this modified system:

$$\mathbf{x}^{(k+1)} = \left(\mathbf{I} - \mathbf{P}^{-1}\mathbf{A}\right)\mathbf{x}^{(k)} + \mathbf{P}^{-1}\mathbf{b},$$

which now has an iteration matrix $\mathbf{I} - \mathbf{P}^{-1}\mathbf{A} \approx \mathbf{0}$, which means more **rapid convergence**.

Preconditioned Iteration

$$\mathbf{x}^{(k+1)} = \left(\mathbf{I} - \mathbf{P}^{-1}\mathbf{A}\right)\mathbf{x}^{(k)} + \mathbf{P}^{-1}\mathbf{b}$$

- In practice, we solve linear systems with the matrix $\mathbf{P}$ instead of inverting it:

$$\mathbf{P}\mathbf{x}^{(k+1)} = (\mathbf{P} - \mathbf{A})\,\mathbf{x}^{(k)} + \mathbf{b} = \mathbf{P}\mathbf{x}^{(k)} + \mathbf{r}^{(k)},$$

  where $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$ is the **residual vector**.
- Finally, we obtain the usual form of a **preconditioned stationary iterative solver**

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{P}^{-1}\mathbf{r}^{(k)}.$$

- Note that convergence will be faster if we have a **good initial guess** $\mathbf{x}^{(0)}$.

Some Standard Examples

$$\text{Splitting: } \mathbf{A} = \mathbf{L}_A + \mathbf{U}_A + \mathbf{D}$$

- Since diagonal systems are trivial to solve, we can use the **Jacobi method**

$$\mathbf{P} = \mathbf{D}.$$

- Or since triangular systems are easy to solve by forward/backward substitution, we can use **Gauss-Seidel method**

$$\mathbf{P} = \mathbf{L}_A + \mathbf{D}.$$

- Both of these converge for strictly **diagonally-dominant matrices**.
- Gauss-Seidel converges for **positive-definite matrices** (maybe slowly though!).

A Good Preconditioner

- Note that the matrix **A** is only used when calculating the residual through the **matrix-vector product** $\mathbf{A}\mathbf{x}^{(k)}$.
- We must be able to do a **direct** linear solver for the preconditioner

$$\mathbf{P}\left(\Delta\mathbf{x}\right) = \mathbf{r}^{(k)},$$

  so it must be in some sense simpler to deal with than **A**.
- Preconditioning is all about a **balance between fewer iterations to convergence and larger cost per iteration**.
- Making good preconditioners is in many ways an art and very **problem-specific**:
  The goal is to make $\mathbf{P}^{-1}\mathbf{A}$ as close to being a normal (diagonalizable) matrix with **clustered eigenvalues** as possible.

In the Real World

- Some general preconditioning strategies have been designed, for example, **incomplete** *LU* **factorization** (MATLAB's *cholinc*).
- There are many **more-sophisticated iterative methods** (non-stationary, higher-order, etc) but most have the same **basic structure**:
  At each iteration, solve a preconditioning linear system, do a matrix-vector calculation, and a convergence test.
- For positive-(semi)definite matrices the **Preconditioned Conjugate Gradient** method is good (MATLAB's *pcg*).
- For certain types of matrices specialized methods have been designed, such as **multigrid methods** for linear systems on large grids (PDE solvers in Numerical Methods II).