WISB356, Utrecht, 9 oktober 2012

Scientific Computing

Gerard Sleijpen
Rob Bisseling
Alessandro Sbrizzi

Universiteit Utrecht
Department of Mathematics

http://www.staff.science.uu.nl/∼sleij101/

Aspect werkelijkheid

Modelleer ⇓

Wiskundig model

Discretiseer ⇓

Discreet model

⇓

Computer model

Implementeer ⇓

Simulatie

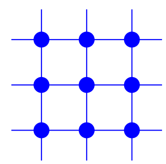
Stroming grondwater

⇓

$$-\nabla(K\nabla\phi) = Q \text{ op } \Omega$$

$$-K\frac{\partial\phi}{\partial x}\cdot n = \gamma(\phi - \phi_0) \text{ op } \partial\Omega$$

⇓ Eindige differences, . . .



$$\ldots \quad \frac{\phi(x+\Delta x,y)-\phi(x-\Delta x,y)}{2\Delta x} \quad \ldots$$

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

⇓ Iteratieve lineaire solver

Rekenschema voor het oplossen van **x** uit $\mathbf{A}\mathbf{x} = \mathbf{b}$

⇓ C++

Simulatie

WISB356, Utrecht, 9 oktober 2012

Preconditioneren van iteratieve methoden

Gerard Sleijpen

Universiteit Utrecht
Department of Mathematics

http://www.staff.science.uu.nl/~sleij101/

Program

- Flexible GCR
- Preconditioning
- D-ILU
- Incomplete LU-decomposition
- Why preconditioning?
- Costs
- How to include a preconditioner
- Savings

The costs of GCR and of Gaussian elimination are comparable for our equations

$$\mathbf{Ax} = \mathbf{b}$$

from 2 dimensional advection-diffusion.
(For problems from 3 d, GCR is the clear winner).

An additional action is required to make iterative methods more efficient.

Program

- Flexible GCR
- Preconditioning
- D-ILU
- Incomplete LU-decomposition
- Why preconditioning?
- Costs
- How to include a preconditioner
- Savings

GCR

```
Choose tol > 0, x, k_max,
Compute r = b - Ax
For k = 0,1,2,...,k_max
    Stop if ||r||_2 <= tol||b||_2
    u_k = r
    c_k = A u_k
    For j = 0,1,2,...,k-1
        β ← c_j^* c_k / σ_j
        u_k = u_k - β u_j
        c_k = c_k - β c_j
    end for
    σ_k = c_k^* c_k,  α ← c_k^* r / σ_k
    x ← x + α u_k
    r ← r - α c_k
end for
```

Choose $tol > 0$, $\mathbf{x}$, $k_{\max}$,

Compute $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$

For $k = 0, 1, 2, \ldots, k_{\max}$

- Stop if $\|\mathbf{r}\|_2 \leq tol\|\mathbf{b}\|_2$
- $\mathbf{u}_k = \mathbf{r}$
- $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$
- For $j = 0, 1, 2, \ldots, k-1$
  - $\beta \leftarrow \mathbf{c}_j^* \mathbf{c}_k / \sigma_j$
  - $\mathbf{u}_k = \mathbf{u}_k - \beta\,\mathbf{u}_j$
  - $\mathbf{c}_k = \mathbf{c}_k - \beta\,\mathbf{c}_j$
- end for
- $\sigma_k = \mathbf{c}_k^* \mathbf{c}_k$, $\alpha \leftarrow \mathbf{c}_k^* \mathbf{r} / \sigma_k$
- $\mathbf{x} \leftarrow \mathbf{x} + \alpha\,\mathbf{u}_k$
- $\mathbf{r} \leftarrow \mathbf{r} - \alpha\,\mathbf{c}_k$

end for

GCR

Choose $tol > 0$, $\mathbf{x}$, $k_{\max}$,
Compute $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$
For $k = 0, 1, 2, \ldots, k_{\max}$

    Stop if $\|\mathbf{r}\|_2 \le tol\|\mathbf{b}\|_2$
    Solve $\mathbf{A}\mathbf{u}_k = \mathbf{r}$ for $\mathbf{u}_k$
    $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$
    For $j = 0, 1, 2, \ldots, k-1$
        $\beta \leftarrow \mathbf{c}_j^* \mathbf{c}_k / \sigma_j$
        $\mathbf{u}_k = \mathbf{u}_k - \beta\,\mathbf{u}_j$
        $\mathbf{c}_k = \mathbf{c}_k - \beta\,\mathbf{c}_j$
    end for
    $\sigma_k = \mathbf{c}_k^* \mathbf{c}_k$, $\alpha \leftarrow \mathbf{c}_k^* \mathbf{r} / \sigma_k$
    $\mathbf{x} \leftarrow \mathbf{x} + \alpha\,\mathbf{u}_k$
    $\mathbf{r} \leftarrow \mathbf{r} - \alpha\,\mathbf{c}_k$

end for

If, in GCR, we replace the line

$$\mathbf{u}_k = \mathbf{r}_k$$

in, say, the fourth step ($k = 4$) by

$$\text{Solve} \quad \mathbf{A}\mathbf{u}_k = \mathbf{r}_k \quad \text{for } \mathbf{u}_k,$$

then $\mathbf{r}_{k+1} = \mathbf{0}$.

If, in GCR, we replace the line

$$\mathbf{u}_k = \mathbf{r}_k$$

in, say, the fourth step ($k = 4$) by

$$\text{Solve} \quad \mathbf{A}\mathbf{u}_k = \mathbf{r}_k \quad \text{for } \mathbf{u}_k,$$

then $\mathbf{r}_{k+1} = \mathbf{0}$.

However,

solving $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$ is as hard as solving $\mathbf{A}\mathbf{x} = \mathbf{b}$.

If, in GCR, we replace the line

$$\mathbf{u}_k = \mathbf{r}_k$$

in, say, the fourth step ($k = 4$) by

$$\text{Solve } \mathbf{A}\mathbf{u}_k = \mathbf{r}_k \text{ for } \mathbf{u}_k,$$

then $\mathbf{r}_{k+1} = \mathbf{0}$.

However,

solving $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$ is as hard as solving $\mathbf{A}\mathbf{x} = \mathbf{b}$.

But it suggests that (cheaply) finding an approximate solution of $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$ might be a good **idea**.

Flexible GCR

```
Choose tol > 0, x, k_max,
Compute r = b − Ax
For k = 0, 1, 2, ..., k_max
    Stop if ||r||_2 ≤ tol||b||_2
    Find an appropriate search vector u_k
    c_k = A u_k
    For j = 0, 1, 2, ..., k − 1
        β ← c_j^* c_k / σ_j
        u_k = u_k − β u_j
        c_k = c_k − β c_j
    end for
    σ_k = c_k^* c_k,  α ← c_k^* r / σ_k
    x ← x + α u_k
    r ← r − α c_k
end for
```

Find an appropriate search vector $\mathbf{u}_k$

In principle, any appropriate vector $\mathbf{u}_k$ can be “injected” in the search subspace.

Example.

- $\mathbf{u}_0$ is the vector variant of the pressure function in the neighbourhood of a pump.
- $\mathbf{u}_0 = \widetilde{\mathbf{x}}$, with $\widetilde{\mathbf{x}}$ the solution before installing a pump, or before the river started carrying water.
- The solution of $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$ as obtained with m steps of GCR (GCR is **nested** here with itself).
- Eigenvectors of $\mathbf{A}$ that correspond to small eigenvalues.

Find an appropriate search vector $\mathbf{u}_k$

In principle, any appropriate vector $\mathbf{u}_k$ can be "injected" in the search subspace.

Remark. In this generality
flexible GCR does **not** form a Krylov subspace.

Find an appropriate search vector $\mathbf{u}_k$

In principle, any appropriate vector $\mathbf{u}_k$ can be "injected" in the search subspace.

A systematic way to find appropriate vectors $\mathbf{u}_k$
(that is, vectors that are more effective than $\mathbf{u}_k = \mathbf{r}_k$)
is with a so-called **preconditioner**.

Program

- Flexible GCR
- Preconditioning
- D-ILU
- Incomplete LU-decomposition
- Why preconditioning?
- Costs
- How to include a preconditioner
- Savings

Preconditioning

An n by n matrix $\mathbf{M}$ is called a **preconditioner** if

- the system $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ can efficiently be solved and
- $\mathbf{M}$ approximates $\mathbf{A}$ (to some degree).

that is, $\mathbf{u}_k = \mathbf{M}^{-1}\mathbf{r}_k$ *is more effective than* $\mathbf{u}_k = \mathbf{r}_k$ in finding an approximate solution of $\mathbf{A}\mathbf{u} = \mathbf{r}_k$.

Preconditioned GCR

```
Choose tol > 0, x, k_max,
Compute r = b - Ax
For k = 0,1,2,...,k_max
    Stop if ||r||_2 <= tol ||b||_2
    Solve M u_k = r_k  for u_k
    c_k = A u_k
    For j = 0,1,2,...,k-1
        β ← c_j^* c_k / σ_j
        u_k = u_k - β u_j
        c_k = c_k - β c_j
    end for
    σ_k = c_k^* c_k,  α ← c_k^* r / σ_k
    x ← x + α u_k
    r ← r - α c_k
end for
```

Choose $tol > 0$, $\mathbf{x}$, $k_{\max}$,
Compute $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$
For $k = 0, 1, 2, \ldots, k_{\max}$
  Stop if $\|\mathbf{r}\|_2 \leq tol \|\mathbf{b}\|_2$
  Solve $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ for $\mathbf{u}_k$
  $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$
  For $j = 0, 1, 2, \ldots, k-1$
    $\beta \leftarrow \mathbf{c}_j^* \mathbf{c}_k / \sigma_j$
    $\mathbf{u}_k = \mathbf{u}_k - \beta\, \mathbf{u}_j$
    $\mathbf{c}_k = \mathbf{c}_k - \beta\, \mathbf{c}_j$
  end for
  $\sigma_k = \mathbf{c}_k^* \mathbf{c}_k$, $\alpha \leftarrow \mathbf{c}_k^* \mathbf{r} / \sigma_k$
  $\mathbf{x} \leftarrow \mathbf{x} + \alpha\, \mathbf{u}_k$
  $\mathbf{r} \leftarrow \mathbf{r} - \alpha\, \mathbf{c}_k$
end for

Preconditioning

An n by n matrix $\mathbf{M}$ is called a **preconditioner** if

- the system $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ can efficiently be solved and
- $\mathbf{M}$ approximates $\mathbf{A}$ (to some degree).

Examples.

- **Diagonal preconditioning**. $\mathbf{M} \equiv \mathbf{D}_A \equiv \mathrm{diag}(\mathbf{A})$.

Preconditioning

An n by n matrix $\mathbf{M}$ is called a **preconditioner** if

- the system $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ can efficiently be solved and
- $\mathbf{M}$ approximates $\mathbf{A}$ (to some degree).

Examples.

- **Diagonal preconditioning**. $\mathbf{M} \equiv \mathbf{D}_A \equiv \mathrm{diag}(\mathbf{A})$.

Usually this does not lead to a 'great' reduction in the number of required iteration steps. But, on the other hand, application of this preconditioner is extremely cheap.

Solving $\mathbf{D}_A\mathbf{u}_k = \mathbf{r}_k$ *costs* n *flop extra per step.*
In k *steps this is* kn *flop.*
With a reduction of the required number of steps from, say, 100 to 98 the 'gain' would be $1200n$ *flop*
with a 'loss' of only $100n$ *flop*

Preconditioning

An n by n matrix $\mathbf{M}$ is called a **preconditioner** if

- the system $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ can efficiently be solved and
- $\mathbf{M}$ approximates $\mathbf{A}$ (to some degree).

Examples.

- **Diagonal preconditioning**. $\mathbf{M} \equiv \mathbf{D}_A \equiv \text{diag}(\mathbf{A})$.
- **Gauss–Seidel**. $\mathbf{M} \equiv \mathbf{L}_A + \mathbf{D}_A$
  where $\mathbf{L}_A$ is the strict lower triangular part $\mathbf{A}$:

$$\mathbf{L}_{i,j} = \mathbf{A}_{i,j} \ \text{ if } \ i > j \ \text{ and } \ \mathbf{L}_{i,j} = 0 \ \text{ else.}$$

Preconditioning

An n by n matrix $\mathbf{M}$ is called a **preconditioner** if

- the system $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ can efficiently be solved and
- $\mathbf{M}$ approximates $\mathbf{A}$ (to some degree).

Examples.

- **Diagonal preconditioning**. $\mathbf{M} \equiv \mathbf{D}_A \equiv \mathrm{diag}(\mathbf{A})$.
- **Gauss–Seidel**. $\mathbf{M} \equiv \mathbf{L}_A + \mathbf{D}_A$.
- A variant: $\mathbf{M} = \mathbf{D}_A + \mathbf{U}_A$
  with $\mathbf{U}_A$ the strict upper triangular part of $\mathbf{A}$.
- A variant called **Successive overrelaxation**:
  $\mathbf{M} \equiv \mathbf{L}_A + \frac{1}{\omega}\mathbf{D}_A$ with ω a **relaxation** parameter.

Preconditioning

An n by n matrix $\mathbf{M}$ is called a **preconditioner** if

- the system $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ can efficiently be solved and
- $\mathbf{M}$ approximates $\mathbf{A}$ (to some degree).

Examples.

- **Diagonal preconditioning**. $\mathbf{M} \equiv \mathbf{D}_A \equiv \mathrm{diag}(\mathbf{A})$.
- **Gauss–Seidel**. $\mathbf{M} \equiv \mathbf{L}_A + \mathbf{D}_A$.
- **Symmetric Successive overrelaxation**.

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_U)$$

with $\mathbf{D} \equiv \frac{1}{\omega}\mathbf{D}_A$ for a **relaxation** parameter ω.

These "classical" preconditioners have been introduced (and used until ±1975 only) in combination with Richardson iteration. From ±1985 on they where used as preconditioner.

Preconditioning

An n by n matrix $\mathbf{M}$ is called a **preconditioner** if

- the system $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ can efficiently be solved and
- $\mathbf{M}$ approximates $\mathbf{A}$ (to some degree).

Examples.

- **Diagonal preconditioning**. $\mathbf{M} \equiv \mathbf{D}_A \equiv \mathrm{diag}(\mathbf{A})$.
- **Gauss–Seidel**. $\mathbf{M} \equiv \mathbf{L}_A + \mathbf{D}_A$.
- **Symmetric Successive overrelaxation**.

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_U)$$

with $\mathbf{D} \equiv \frac{1}{\omega}\mathbf{D}_A$ for a **relaxation** parameter ω.

Note that $\mathbf{M} = \mathbf{A} + \mathbf{R}$ for

$$\mathbf{R} \equiv (\tfrac{1}{\omega} - 1)\mathbf{D}_A + \mathbf{L}_A\mathbf{D}^{-1}\mathbf{U}_A$$

Program

- Flexible GCR
- Preconditioning
- D-ILU
- Incomplete LU-decomposition
- Why preconditioning?
- Costs
- How to include a preconditioner
- Savings

Preconditioning

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A) = \mathbf{A} + \mathbf{R}$$

The system $\mathbf{M}\mathbf{u} = \mathbf{r}$ can be solved in three steps.

- Solve $(\mathbf{L}_A + \mathbf{D})\mathbf{u}' = \mathbf{r}$ for $\mathbf{u}'$.
- Compute $\mathbf{u}'' = \mathbf{D}\mathbf{u}'$.
- Solve $(\mathbf{D} + \mathbf{U}_A)\mathbf{u} = \mathbf{u}''$ for $\mathbf{u}$.

Assignment. Write a function subroutine

$$\mathbf{u} = \texttt{Msolve}(\mathbf{A}, \mathbf{D}, \mathbf{r})$$

that incorporates the above steps. Try to make the routine as efficient as possible also concerning use of memory.

Hint. For testing purposes, you can initially take

$$\mathbf{D} = \mathbf{D}_A \text{ of } \mathbf{D} = \frac{1}{\omega}\mathbf{D}_A .$$

Preconditioning

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A) = \mathbf{A} + \mathbf{R}$$

The system $\mathbf{M}\mathbf{u} = \mathbf{r}$ can be solved in three steps.

- Solve $(\mathbf{L}_A + \mathbf{D})\mathbf{u}' = \mathbf{r}$ for $\mathbf{u}'$.
- Compute $\mathbf{u}'' = \mathbf{D}\mathbf{u}'$.
- Solve $(\mathbf{D} + \mathbf{U}_A)\mathbf{u} = \mathbf{u}''$ for $\mathbf{u}$.

Assignment. Write a function subroutine

$$\mathbf{u} = \texttt{Msolve}(\mathbf{A}, \mathbf{D}, \mathbf{r})$$

that incorporates the above steps. Try to make the routine as efficient as possible also concerning use of memory.

Incorporate `Msolve` in GCR: write a routine PGCR

$$\mathbf{x} = \texttt{PGCR}(\mathbf{A}, \mathbf{b}, \mathbf{x}_0, \mathit{tol}, k_{\mathrm{max}}, \mathbf{D})$$

Preconditioning

Find a diagonal matrix $\mathbf{D}$ such that with

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A) = \mathbf{A} + \mathbf{R}$$

the "error"

$$\mathbf{R} \equiv \mathbf{D} - \mathbf{D}_A + \mathbf{L}_A \mathbf{D}^{-1}\mathbf{U}_A$$

is small in some sense.

Preconditioning

Find a diagonal matrix $\mathbf{D}$ such that with

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A) = \mathbf{A} + \mathbf{R}$$

the "error"

$$\mathbf{R} \equiv \mathbf{D} - \mathbf{D}_A + \mathbf{L}_A \mathbf{D}^{-1}\mathbf{U}_A$$

is small in some sense.

Examples.

- **Diagonal-Incomplete LU**: diag($\mathbf{R}$) $= \mathbf{0}$.

Preconditioning

Find a diagonal matrix $\mathbf{D}$ such that with

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A) = \mathbf{A} + \mathbf{R}$$

the "error"

$$\mathbf{R} \equiv \mathbf{D} - \mathbf{D}_A + \mathbf{L}_A \mathbf{D}^{-1} \mathbf{U}_A$$

is small in some sense.

Examples.

- **Diagonal-Incomplete LU**: $\text{diag}(\mathbf{R}) = \mathbf{0}$.
- **D-Modified ILU**: $\mathbf{R}\mathbf{1} = \mathbf{0}$, with $\mathbf{1} \equiv (1, 1, \ldots, 1)^\mathsf{T}$

Preconditioning

Find a diagonal matrix $\mathbf{D}$ such that with

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A) = \mathbf{A} + \mathbf{R}$$

the "error"

$$\mathbf{R} \equiv \mathbf{D} - \mathbf{D}_A + \mathbf{L}_A \mathbf{D}^{-1} \mathbf{U}_A$$

is small in some sense.

Examples.

- **Diagonal-Incomplete LU**: $\mathrm{diag}(\mathbf{R}) = \mathbf{0}$.
- **D-Modified ILU**: $\mathbf{R1} = \mathbf{0}$, with $\mathbf{1} \equiv (1, 1, \ldots, 1)^{\mathsf{T}}$
- **D-Relaxed ILU**: a mix of ILU and MILU

Program

- Flexible GCR
- Preconditioning
- D-ILU
- Incomplete LU-decomposition
- Why preconditioning?
- Costs
- How to include a preconditioner
- Savings

LU-decomposition

$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity.

Let ℓ_j be the jth column of $\mathbf{L}$ and
$\mathbf{e}_j$ the jth standard basis vector
($\ell_j = \mathbf{L}(:, j)$, $\mathbf{e}_j = \mathbf{I}(:, j)$ in MATLAB notation).

Exercise. Prove

- $(\mathbf{I} - \ell_j \mathbf{e}_j^*)^{-1} = \mathbf{I} + \ell_j \mathbf{e}_j^*$
- $(\mathbf{I} + \ell_j \mathbf{e}_j^*)(\mathbf{I} + \ell_k \mathbf{e}_k^*) = \mathbf{I} + \ell_j \mathbf{e}_j^* + \ell_k \mathbf{e}_k^*$ if $j < k$.
- $(\mathbf{I} + \mathbf{L})^{-1} = (\mathbf{I} - \ell_{n-1} \mathbf{e}_{n-1}^*)(\mathbf{I} - \ell_{n-2} \mathbf{e}_{n-2}^*) \ldots (\mathbf{I} - \ell_1 \mathbf{e}_1^*)$

Interpretation. If $\mathbf{U}' = (\mathbf{I} - \ell_1 \mathbf{e}_1^*)\mathbf{U}$, then

$$\mathbf{U}'(i, :) = \mathbf{U}(i, :) - \ell_1(i)\mathbf{U}(1, :)$$

a multiple of the 1st row of $\mathbf{U}$ is subtracted from the ith row.

LU-decomposition

$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:, j)$, $\mathbf{e}_j \equiv \mathbf{I}(:, j)$.

LU-decomposition or **Gaussian elimination**:

$\mathbf{U}^{(0)} \equiv \mathbf{A}, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n-1)} = \mathbf{U}$ such that

$$\mathbf{U}^{(j)} = (\mathbf{I} - \ell_j \, \mathbf{e}_j^*)\mathbf{U}^{(j-1)} \qquad (j = 1, \ldots, n)$$

and the jth column of $\mathbf{U}^{(j)}$ below the diagonal is zero:
with $p_j \equiv \mathbf{U}^{(j-1)}(j, j)$, $\ell_j(i) = \mathbf{U}^{(j-1)}(i, j)/p_j$ for $i > j$.

LU-decomposition

$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:,j)$, $\mathbf{e}_j \equiv \mathbf{I}(:,j)$.

LU-decomposition or **Gaussian elimination**:

$\mathbf{U}^{(0)} \equiv \mathbf{A}, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n-1)} = \mathbf{U}$ such that

$$\mathbf{U}^{(j)} = (\mathbf{I} - \ell_j\, \mathbf{e}_j^*)\mathbf{U}^{(j-1)} \qquad (j = 1, \ldots, n)$$

and the jth column of $\mathbf{U}^{(j)}$ below the diagonal is zero:
with $p_j \equiv \mathbf{U}^{(j-1)}(j,j)$, $\ell_j(i) = \mathbf{U}^{(j-1)}(i,j)/p_j$ for $i > j$.

Theorem. If the **pivots** $p_j \neq 0$ all j, then

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

LU-decomposition

$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:,j)$, $\mathbf{e}_j \equiv \mathbf{I}(:,j)$.

LU-decomposition or **Gaussian elimination**:

$\mathbf{U}^{(0)} \equiv \mathbf{A}, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n-1)} = \mathbf{U}$ such that

$$\mathbf{U}^{(j)} = (\mathbf{I} - \ell_j \, \mathbf{e}_j^*)\mathbf{U}^{(j-1)} \qquad (j = 1, \ldots, n)$$

and the jth column of $\mathbf{U}^{(j)}$ below the diagonal is zero:
with $p_j \equiv \mathbf{U}^{(j-1)}(j,j)$, $\ell_j(i) = \mathbf{U}^{(j-1)}(i,j)/p_j$ for $i > j$.

Theorem. If the **pivots** $p_j \neq 0$ all j, then

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

Sparsity pattern of $\mathbf{A}$; $\quad \mathcal{F}_A \equiv \{(i,j) \mid \mathbf{A}(i,j) \neq 0\}$

Fill: $\quad \{(i,j) \notin \mathcal{F}_A \mid \mathbf{L}(i,j) \neq 0 \text{ or } \mathbf{U}^{(k)}(i,j) \neq 0\}$

LU-decomposition

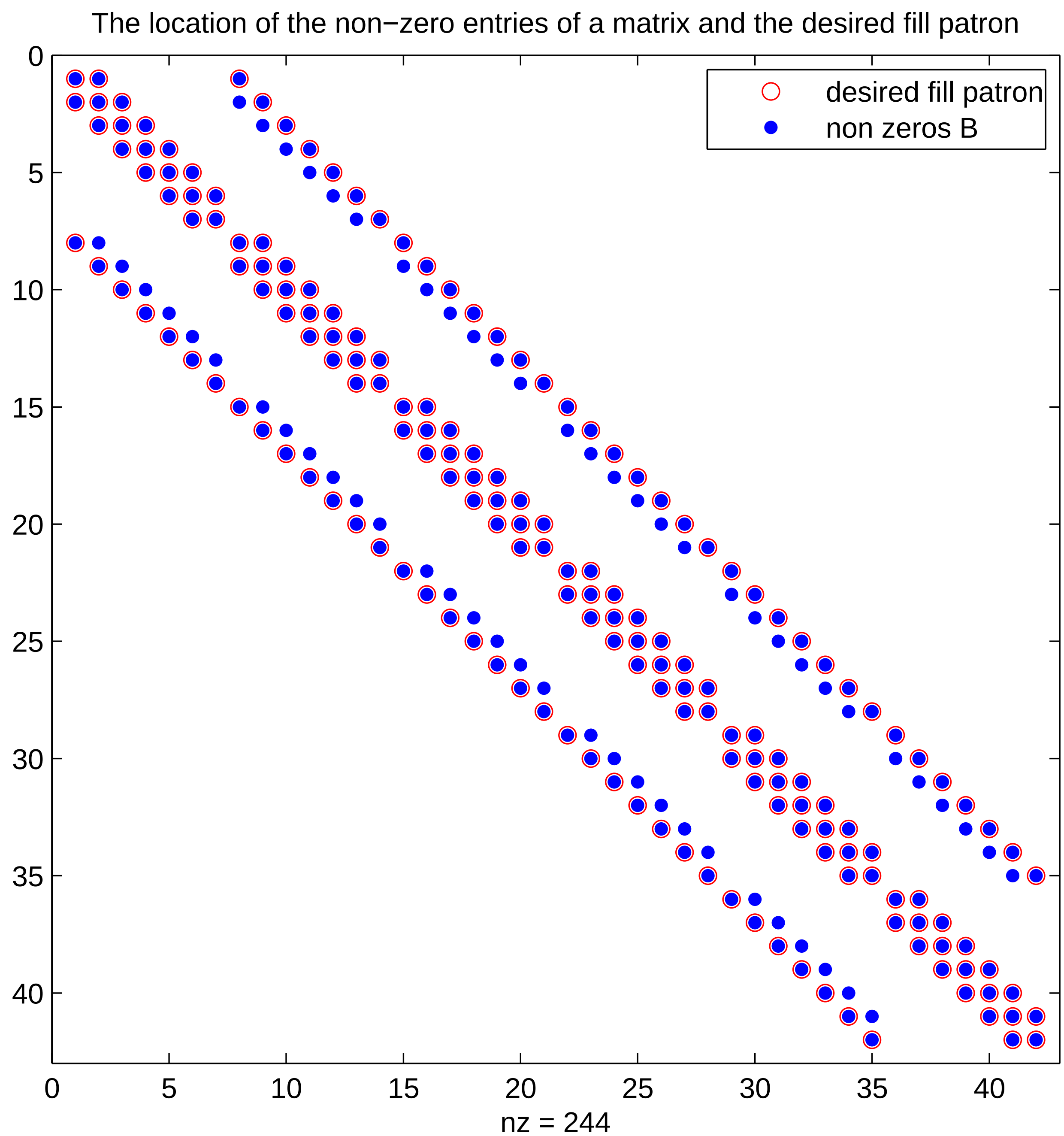
$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:, j)$, $\mathbf{e}_j \equiv \mathbf{I}(:, j)$.

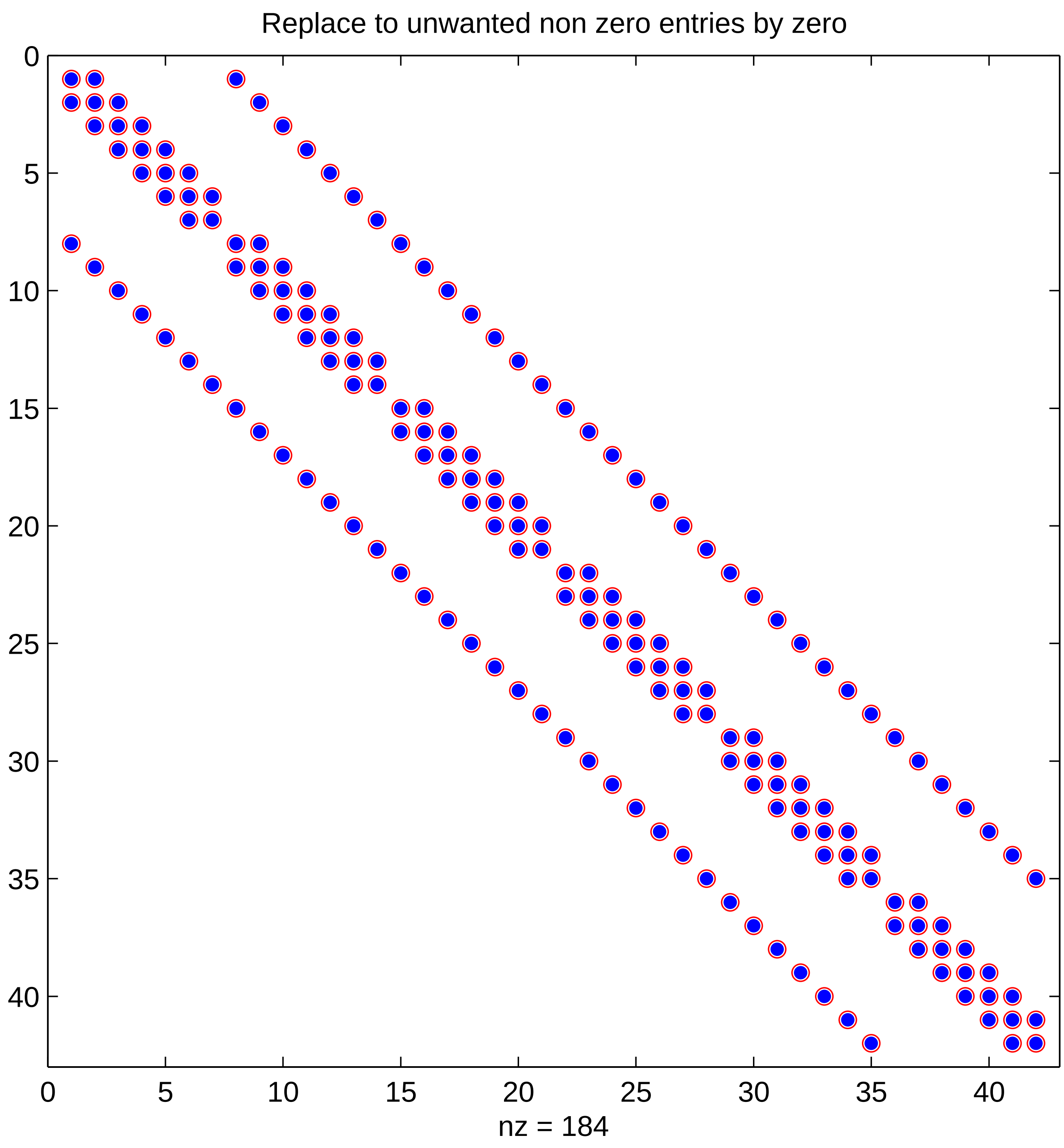
Incomplete LU-decomposition.
Select a **fill pattern** $\mathcal{F} \subset \{(i, j) \mid i, j = 1, \ldots, n\}$.
If $\mathbf{B}$ is an $n \times n$ matrix, then $\mathbf{B}'$ is the matrix with entries $\mathbf{B}'(i, j) = \mathbf{B}(i, j)$ if $(i, j) \in \mathcal{F}$ and $\mathbf{B}'(i, j) = 0$ if $(i, j) \notin \mathcal{F}$.
Put $\Pi(\mathbf{B}) = \mathbf{B}'$.

$\mathbf{U}^{(0)} \equiv \mathbf{A}$, $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n-1)} = \mathbf{U}$ such that

$$\widetilde{\mathbf{U}}^{(j)} = (\mathbf{I} - \ell_j \mathbf{e}_j^*)\mathbf{U}^{(j-1)}, \qquad \mathbf{U}^{(j)} = \Pi(\widetilde{\mathbf{U}}^{(j)})$$

and the j column of $\widetilde{\mathbf{U}}^{(j)}$ below the diagonal is zero:
with $p_j \equiv \mathbf{U}^{(j-1)}(j, j)$, $\ell_j(i) = \mathbf{U}^{(j-1)}(i, j)/p_j$ for $i > j$.


The location of the non−zero entries of a matrix and the desired fill patron
desired fill patron
non zeros B
0
5
10
15
20
25
30
35
40
nz = 244



Replace to unwanted non zero entries by zero
0
5
10
15
20
25
30
35
40
nz = 184

LU-decomposition

$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:, j)$, $\mathbf{e}_j \equiv \mathbf{I}(:, j)$.

Incomplete LU-decomposition.
Select a **fill pattern** $\quad \mathcal{F} \subset \{(i, j) \mid i, j = 1, \ldots, n\}$.
If $\mathbf{B}$ is an $n \times n$ matrix, then $\mathbf{B}'$ is the matrix with entries $\mathbf{B}'(i, j) = \mathbf{B}(i, j)$ if $(i, j) \in \mathcal{F}$ and $\mathbf{B}'(i, j) = 0$ if $(i, j) \notin \mathcal{F}$.
Put $\quad \Pi(\mathbf{B}) = \mathbf{B}'$.

$\mathbf{U}^{(0)} \equiv \mathbf{A}$, $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n-1)} = \mathbf{U}$ such that

$$\widetilde{\mathbf{U}}^{(j)} = (\mathbf{I} - \ell_j \mathbf{e}_j^*) \mathbf{U}^{(j-1)}, \qquad \mathbf{U}^{(j)} = \Pi(\widetilde{\mathbf{U}}^{(j)})$$

and the j column of $\widetilde{\mathbf{U}}^{(j)}$ below the diagonal is zero:
with $\; p_j \equiv \mathbf{U}^{(j-1)}(j, j)$, $\; \ell_j(i) = \mathbf{U}^{(j-1)}(i, j)/p_j \;$ for $\; i > j$.

Theorem. With ILU and $\mathbf{M} = \mathbf{LU}$,
we have that $\; \mathbf{A}(i, j) = \mathbf{M}(i, j) \;$ for all $(i, j) \in \mathcal{F}$

LU-decomposition

$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:, j)$, $\mathbf{e}_j \equiv \mathbf{I}(:, j)$.

Modified ILU-decomposition. Select a
fill pattern $\mathcal{F} \subset \{(i, j) \mid i, j = 1, \ldots, n\}$ with $\{(i, i)\} \subset \mathcal{F}$.
If $\mathbf{B}$ is an $n \times n$ matrix, then $\widetilde{\mathbf{B}}$ is the matrix with entries

$$\widetilde{\mathbf{B}}(i, j) = \mathbf{B}(i, j) \text{ if } (i, j) \in \mathcal{F}, i \neq j$$

$$\widetilde{\mathbf{B}}(i, j) = 0 \text{ if } (i, j) \notin \mathcal{F},$$

$$\widetilde{\mathbf{B}}(i, i) = \mathbf{B}(i, i) + \sum_{j,(i,j) \notin \mathcal{F}} \mathbf{B}(i, j)$$

Put $\quad \Pi_M(\mathbf{B}) = \widetilde{\mathbf{B}}$.

$\mathbf{U}^{(0)} \equiv \mathbf{A}, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n-1)} = \mathbf{U}$ such that

$$\widetilde{\mathbf{U}}^{(j)} = (\mathbf{I} - \ell_j \mathbf{e}_j^*)\mathbf{U}^{(j-1)}, \qquad \mathbf{U}^{(j)} = \Pi_M(\widetilde{\mathbf{U}}^{(j)})$$

and the jth column of $\widetilde{\mathbf{U}}^{(j)}$ below the diagonal is zero:
with $p_j \equiv \mathbf{U}^{(j-1)}(j, j)$, $\ell_j(i) = \mathbf{U}^{(j-1)}(i, j)/p_j$ for $i > j$.

LU-decomposition

L is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
I is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:, j)$, $\mathbf{e}_j \equiv \mathbf{I}(:, j)$.

Modified ILU-decomposition. Select a
fill pattern $\mathcal{F} \subset \{(i,j) \mid i,j = 1, \ldots, n\}$ with $\{(i,i)\} \subset \mathcal{F}$.
If **B** is an $n \times n$ matrix, then $\widetilde{\mathbf{B}}$ is the matrix with entries

$$\widetilde{\mathbf{B}}(i,j) = \mathbf{B}(i,j) \text{ if } (i,j) \in \mathcal{F}, i \neq j$$

$$\widetilde{\mathbf{B}}(i,j) = 0 \text{ if } (i,j) \notin \mathcal{F},$$

$$\widetilde{\mathbf{B}}(i,i) = \mathbf{B}(i,i) + \textstyle\sum_{j,(i,j) \notin \mathcal{F}} \mathbf{B}(i,j)$$

Put $\quad \Pi_M(\mathbf{B}) = \widetilde{\mathbf{B}}$. $\qquad$ **Note**. $\mathbf{B1} = \Pi_M(\mathbf{B})\mathbf{1}$.

$\mathbf{U}^{(0)} \equiv \mathbf{A}$, $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n-1)} = \mathbf{U}$ such that

$$\widetilde{\mathbf{U}}^{(j)} = (\mathbf{I} - \ell_j \mathbf{e}_j^*)\mathbf{U}^{(j-1)}, \qquad \mathbf{U}^{(j)} = \Pi_M(\widetilde{\mathbf{U}}^{(j)})$$

and the jth column of $\widetilde{\mathbf{U}}^{(j)}$ below the diagonal is zero:
with $p_j \equiv \mathbf{U}^{(j-1)}(j,j)$, $\ell_j(i) = \mathbf{U}^{(j-1)}(i,j)/p_j$ for $i > j$.

LU-decomposition

$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:, j)$, $\mathbf{e}_j \equiv \mathbf{I}(:, j)$.

Theorem. With MILU-decomposition and $\mathbf{M} \equiv \mathbf{LU}$,
we have that $\mathbf{M1} = \mathbf{A1}$, where $\mathbf{1} \equiv (1, 1, \ldots, 1)^\mathsf{T}$.

LU-decomposition

$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:, j)$, $\mathbf{e}_j \equiv \mathbf{I}(:, j)$.

Relaxed ILU-decomposition. Select an $\omega \in [0, 1]$ and a **fill pattern** $\mathcal{F} \subset \{(i, j) \mid i, j = 1, \ldots, n\}$ with $\{(i, i)\} \subset \mathcal{F}$.
If $\mathbf{B}$ is an $n \times n$ matrix, then $\widetilde{\mathbf{B}}$ is the matrix with entries

$$\widetilde{\mathbf{B}}(i,j) = \mathbf{B}(i,j) \text{ if } (i,j) \in \mathcal{F}, i \neq j$$

$$\widetilde{\mathbf{B}}(i,j) = 0 \text{ if } (i,j) \notin \mathcal{F},$$

$$\widetilde{\mathbf{B}}(i,i) = \mathbf{B}(i,i) + \omega \sum_{j,(i,j) \notin \mathcal{F}} \mathbf{B}(i,j)$$

Put $\quad \Pi_\omega(\mathbf{B}) = \widetilde{\mathbf{B}}$.

$\mathbf{U}^{(0)} \equiv \mathbf{A}, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(n-1)} = \mathbf{U}$ such that

$$\widetilde{\mathbf{U}}^{(j)} = (\mathbf{I} - \ell_j \mathbf{e}_j^*) \mathbf{U}^{(j-1)}, \qquad \mathbf{U}^{(j)} = \Pi_\omega(\widetilde{\mathbf{U}}^{(j)})$$

and the jth column of $\widetilde{\mathbf{U}}^{(j)}$ below the diagonal is zero:
with $p_j \equiv \mathbf{U}^{(j-1)}(j, j)$, $\ell_j(i) = \mathbf{U}^{(j-1)}(i, j)/p_j$ for $i > j$.

LU-decomposition

$\mathbf{L}$ is strict lower triangular $n \times n$ (i.e., $\mathbf{L}_{ij} = 0$ if $i \leq j$).
$\mathbf{I}$ is the $n \times n$ identity. $\ell_j \equiv \mathbf{L}(:, j)$, $\mathbf{e}_j \equiv \mathbf{I}(:, j)$.

Remark. RILU(0)=ILU, RILU(1)=MILU.

Diagonal ILU decomposition

Write $\mathbf{A} = \mathbf{L}_A + \mathbf{D}_A + \mathbf{U}_A$ with

$\mathbf{L}_A$ the strict lower triangular part of $\mathbf{A}$

($\mathbf{L}_A(i,j) = \mathbf{A}(i,j)$ if $i > j$, $\mathbf{L}_A(i,j) = 0$ if $i \leq j$)

$\mathbf{D}_A = \mathrm{diag}(\mathbf{A})$ (in Matlab: `D_A=diag(diag(A));`)

$\mathbf{U}_A$ the strict upper triangular part of $\mathbf{A}$.

For an $n \times n$ diagonal matrix $\mathbf{D}$ consider

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A)$$

D-ILU: $\mathbf{D}$ is such that $\mathrm{diag}(\mathbf{M}) = \mathrm{diag}\mathbf{A})$.

Diagonal ILU decomposition

Write $\mathbf{A} = \mathbf{L}_A + \mathbf{D}_A + \mathbf{U}_A$ with

$\mathbf{L}_A$ the strict lower triangular part of $\mathbf{A}$

$(\mathbf{L}_A(i,j) = \mathbf{A}(i,j)$ if $i > j$, $\mathbf{L}_A(i,j) = 0$ if $i \leq j)$

$\mathbf{D}_A = \text{diag}(\mathbf{A})$ (in Matlab: `D_A=diag(diag(A));`)

$\mathbf{U}_A$ the strict upper triangular part of $\mathbf{A}$.

For an $n \times n$ diagonal matrix $\mathbf{D}$ consider

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A)$$

D-ILU: $\mathbf{D}$ is such that $\text{diag}(\mathbf{M}) = \text{diag}\mathbf{A})$.

Theorem. If $\mathbf{A}$ is the matrix from a 5-point stencil (2-d advection diffusion) or from a 7-point stencil (3-d advection diffusion), then D-ILU$=$ILU, i.e., if $\mathbf{L}$ and $\mathbf{U}$ are from ILU, then

$$\mathbf{L} = \mathbf{L}_A\mathbf{D}^{-1} + \mathbf{I} \quad \text{and} \quad \mathbf{U} = \mathbf{D} + \mathbf{U}_A\,.$$

Diagonal ILU decomposition

Write $\mathbf{A} = \mathbf{L}_A + \mathbf{D}_A + \mathbf{U}_A$ with

$\mathbf{L}_A$ the strict lower triangular part of $\mathbf{A}$

$(\mathbf{L}_A(i,j) = \mathbf{A}(i,j)$ if $i > j$, $\mathbf{L}_A(i,j) = 0$ if $i \leq j)$

$\mathbf{D}_A = \mathrm{diag}(\mathbf{A})$ (in Matlab: `D_A=diag(diag(A));`)

$\mathbf{U}_A$ the strict upper triangular part of $\mathbf{A}$.

For an $n \times n$ diagonal matrix $\mathbf{D}$ consider

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A)$$

D-MILU: $\mathbf{D}$ is such that $\mathbf{M1} = \mathbf{A1}$.

Diagonal ILU decomposition

Write $\mathbf{A} = \mathbf{L}_A + \mathbf{D}_A + \mathbf{U}_A$ with

$\mathbf{L}_A$ the strict lower triangular part of $\mathbf{A}$

($\mathbf{L}_A(i,j) = \mathbf{A}(i,j)$ if $i > j$, $\mathbf{L}_A(i,j) = 0$ if $i \leq j$)

$\mathbf{D}_A = \text{diag}(\mathbf{A})$ (in Matlab: `D_A=diag(diag(A));`)

$\mathbf{U}_A$ the strict upper triangular part of $\mathbf{A}$.

For an $n \times n$ diagonal matrix $\mathbf{D}$ consider

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A)$$

D-MILU: $\mathbf{D}$ is such that $\mathbf{M1} = \mathbf{A1}$.

Theorem. If $\mathbf{A}$ is the matrix from a 5-point stencil (2-d advection diffusion) or from a 7-point stencil (3-d advection diffusion), then D-MILU = MILU, i.e., if $\mathbf{L}$ and $\mathbf{U}$ are from MILU, then

$$\mathbf{L} = \mathbf{L}_A\mathbf{D}^{-1} + \mathbf{I} \quad \text{and} \quad \mathbf{U} = \mathbf{D} + \mathbf{U}_A\,.$$

Other ILU-decompositions

The idea behind ILU is to form a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$ such that

- $\mathbf{M} \equiv \mathbf{LU}$ approximates $\mathbf{A}$ well in some sense.
- The systems $\mathbf{Lu}' = \mathbf{r}$ and $\mathbf{Uu} = \mathbf{u}'$ can efficiently be solved,
- **but the L and U should also be efficiently computable.**

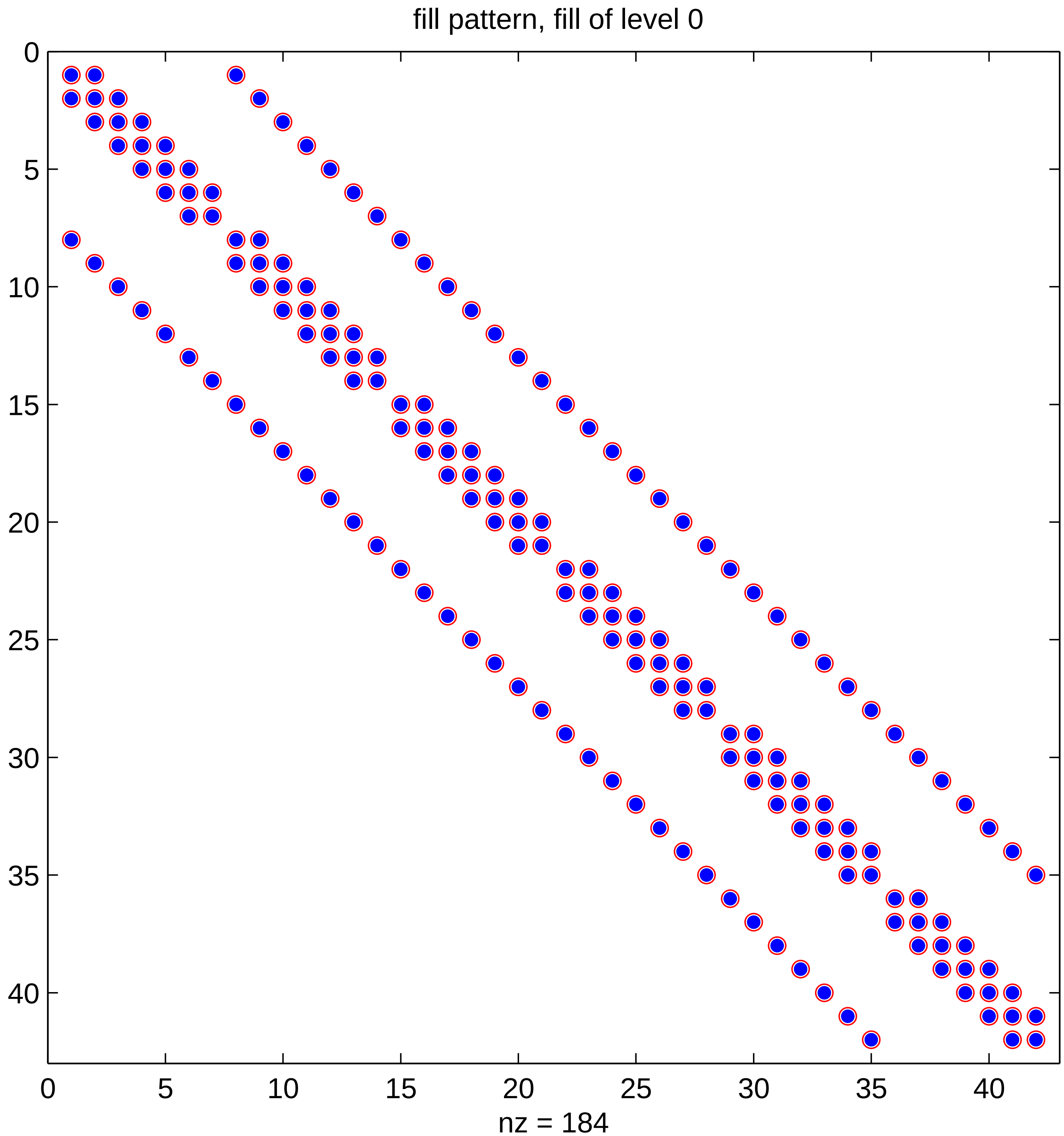
In practise, the first condition and the last have to be balanced and the meaning of "efficient" and "approximates well" often depends on the application.

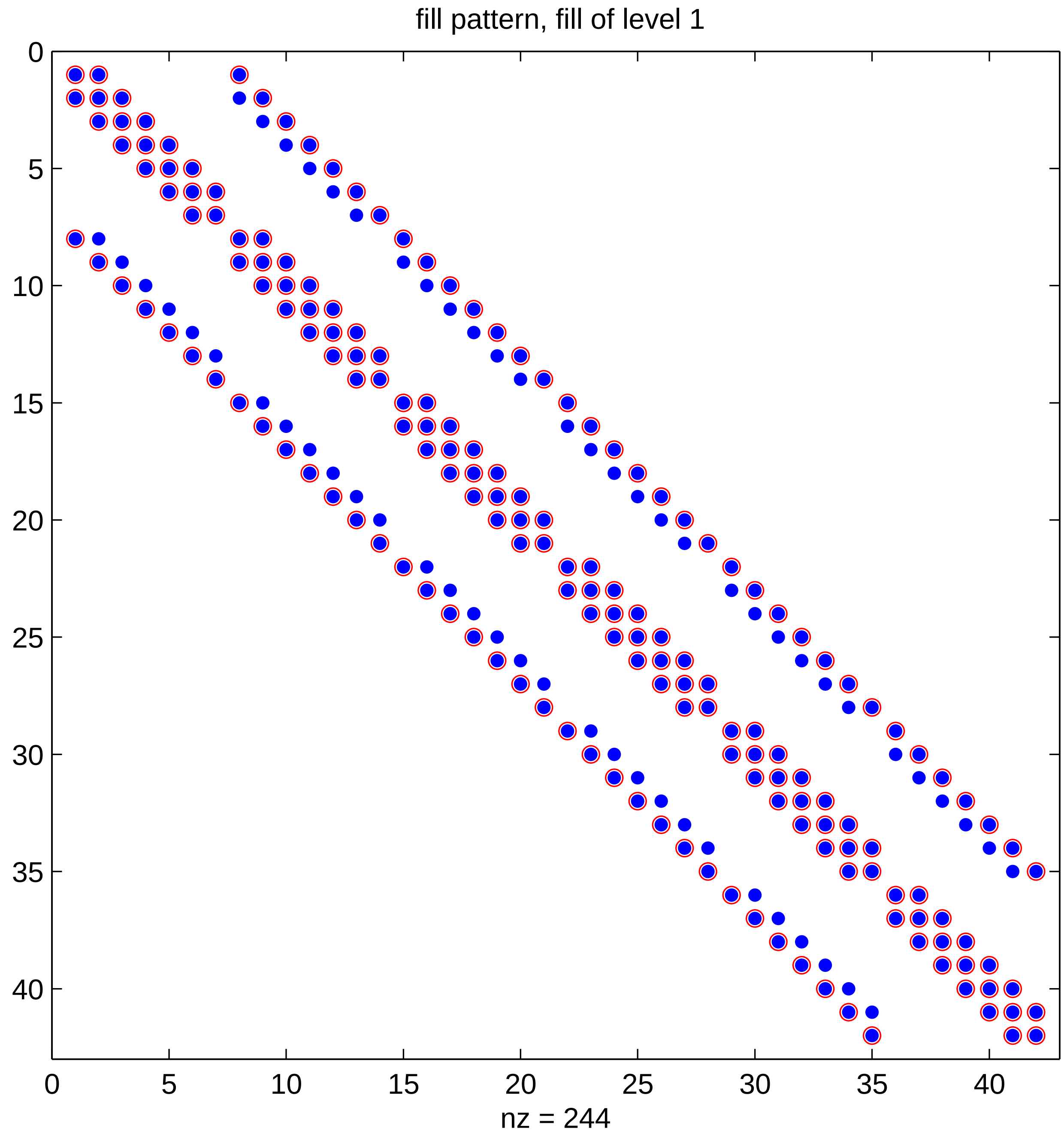
As an extreme example, if $\mathbf{Mx} = \mathbf{Ax}$, *then preconditioned GCR started with* $\mathbf{x}_0 = \mathbf{0}$ *finds* $\mathbf{x}$ *in one step even if* $\mathbf{R} = \mathbf{A} - \mathbf{M}$ *is large.*

Other ILU-decompositions

For a fill pattern $\mathcal{F} \subset \{(i,j) \mid i,j = 1,\ldots,n\}$, define

$$\mathcal{F}^{+} \equiv \{(i,j) \mid (i,k),(k,j) \in \mathcal{F} \ \text{ for some } k < i,\, k < j\}$$


fill pattern, fill of level 0
0
5
10
15
20
25
30
35
40
0
5
10
15
20
25
30
35
40
nz = 184


fill pattern, fill of level 1
0
5
10
15
20
25
30
35
40
nz = 244

Other ILU-decompositions

For a fill pattern $\mathcal{F} \subset \{(i,j) \mid i,j = 1,\ldots,n\}$, define

$$\mathcal{F}^+ \equiv \{(i,j) \mid (i,k),(k,j) \in \mathcal{F} \text{ for some } k < i,\ k < j\}$$

Interpretation. In the kth step of the Gaussian elimination, the matrix entries at the position (i,k) and (k,j) are used to form the entry at position (i,j):

$$\mathbf{U}^{(k)}(i,j) = \mathbf{U}^{(k-1)}(i,j) - \mathbf{U}^{(k-1)}(i,k)\mathbf{U}^{(k-1)}(k,j)/p_k$$

with pivot $p_k = \mathbf{U}^{(k-1)}(k,k)$.

If $\mathcal{F} = \mathcal{F}_A$, then $\mathcal{F}^+$ contains the indices of possible non-zero matrix entries formed directly from non-zeros of the original matrix.

$\mathcal{F} = \mathcal{F}_A$ is **level 0** fill, $\mathcal{F}^+$ is **level 1** fill.

Other ILU-decompositions

For a fill pattern $\mathcal{F} \subset \{(i,j) \mid i,j = 1,\ldots,n\}$, define

$$\mathcal{F}^+ \equiv \{(i,j) \mid (i,k),(k,j) \in \mathcal{F} \text{ for some } k < i,\, k < j\}$$

Terminology. With $\mathcal{F}_A(0) \equiv \mathcal{F}_A \equiv \{(i,j) \mid \mathbf{A}(i,j) \neq 0\}$,

$$\mathcal{F}_A(\ell) \equiv \mathcal{F}_A(\ell-1)^+ \text{ for } \ell = 1,2,\ldots$$

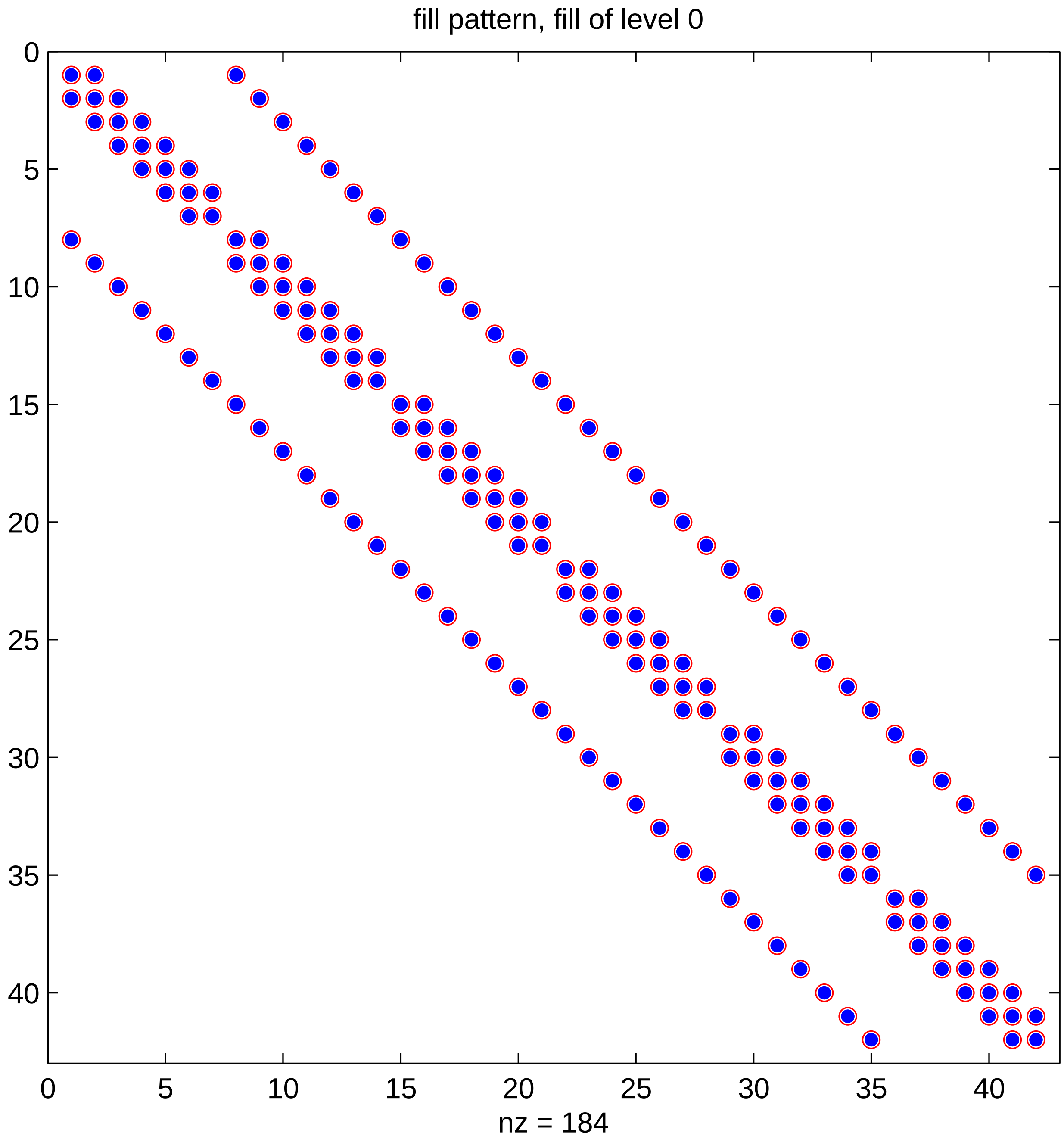
$\mathcal{F}_A(\ell)$ is **fill of level ℓ**.

Note that to determine $\mathcal{F}_A(\ell)$ no specific values for the entries of $\mathbf{A}$ are required.

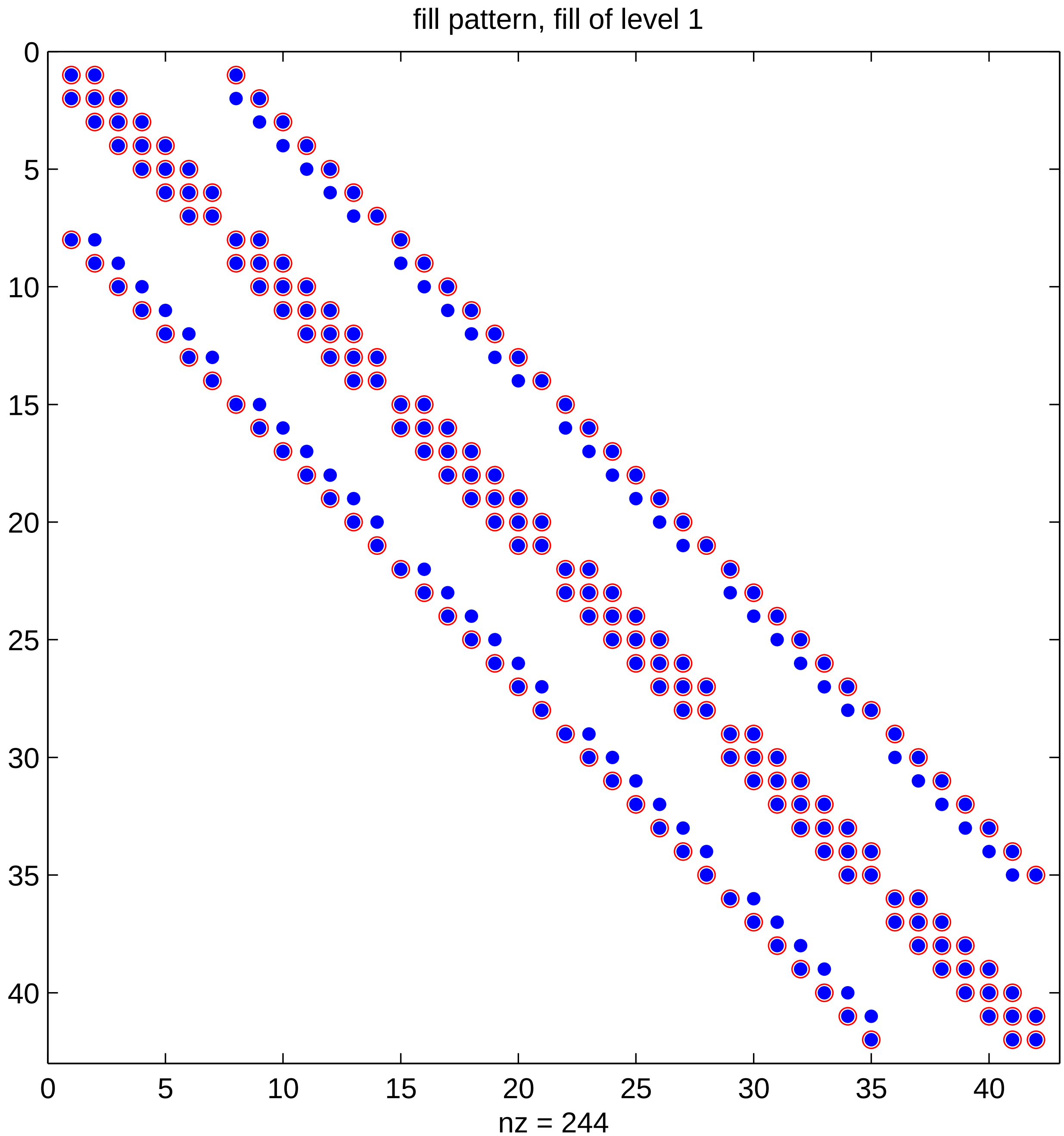
ILU(ℓ), that is, ILU for $\mathbf{A}$ with fill pattern $\mathcal{F}_A(\ell)$,

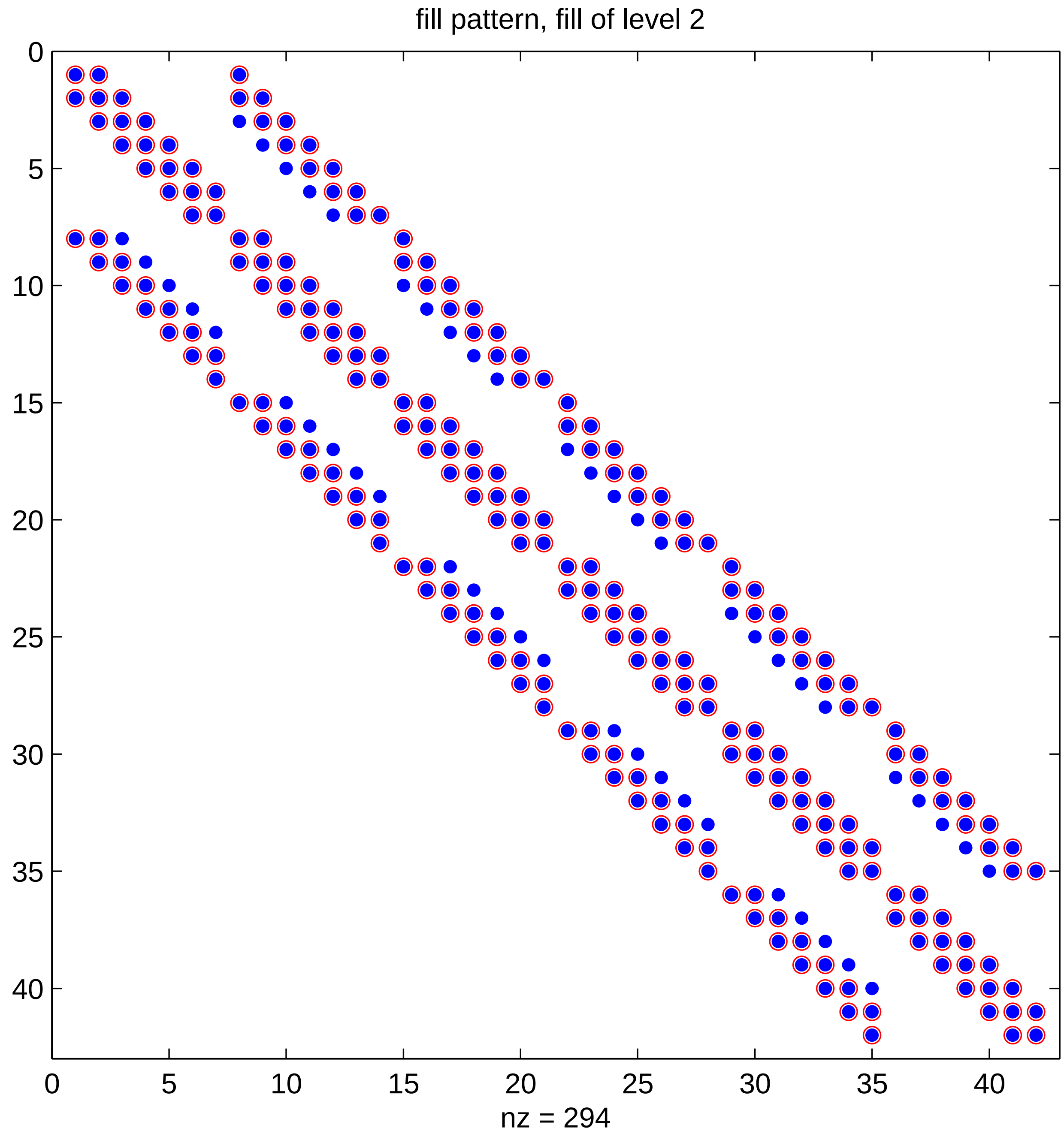
is called **ILU of level ℓ**.

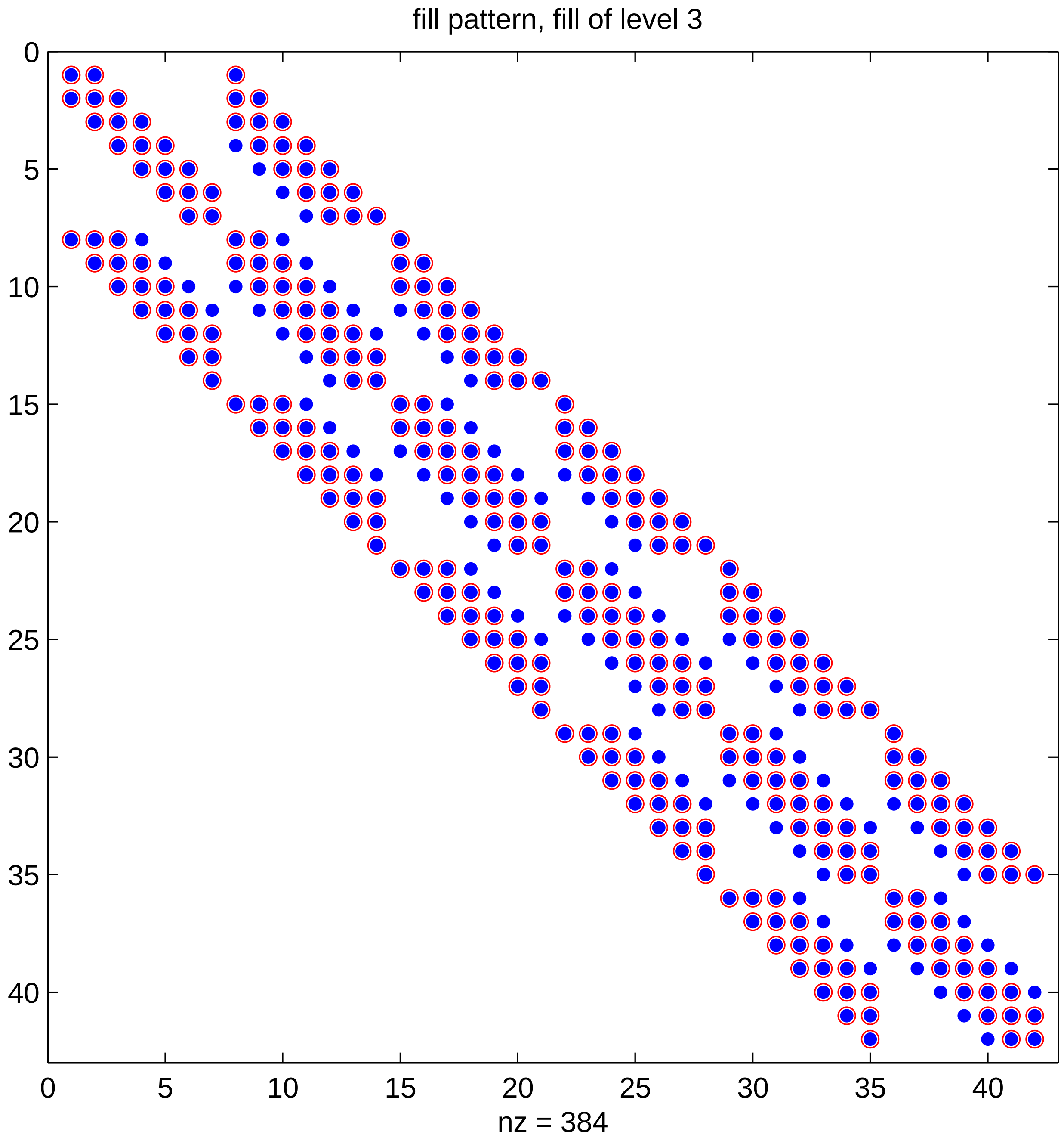
ILU(0) = ILU.

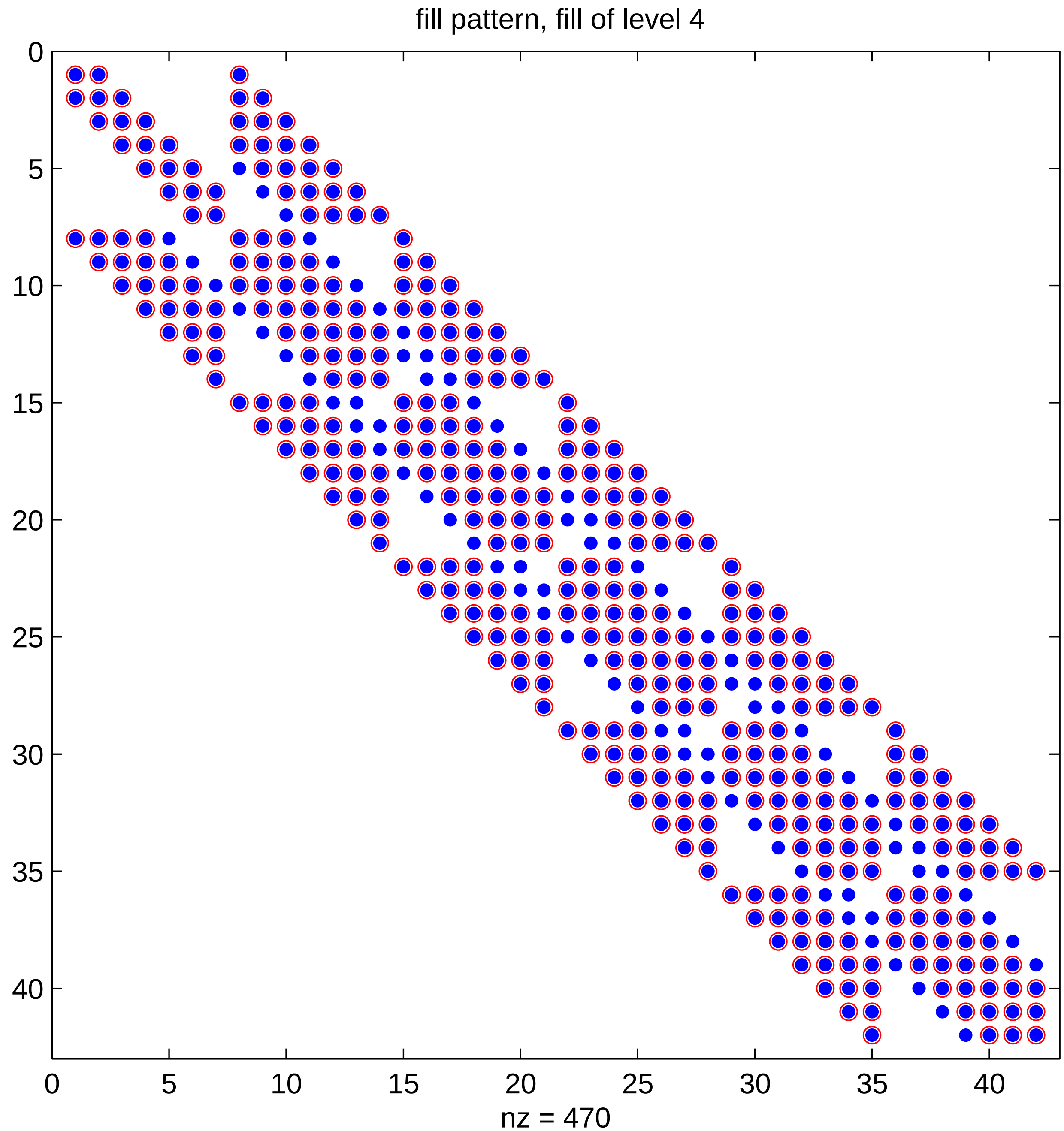


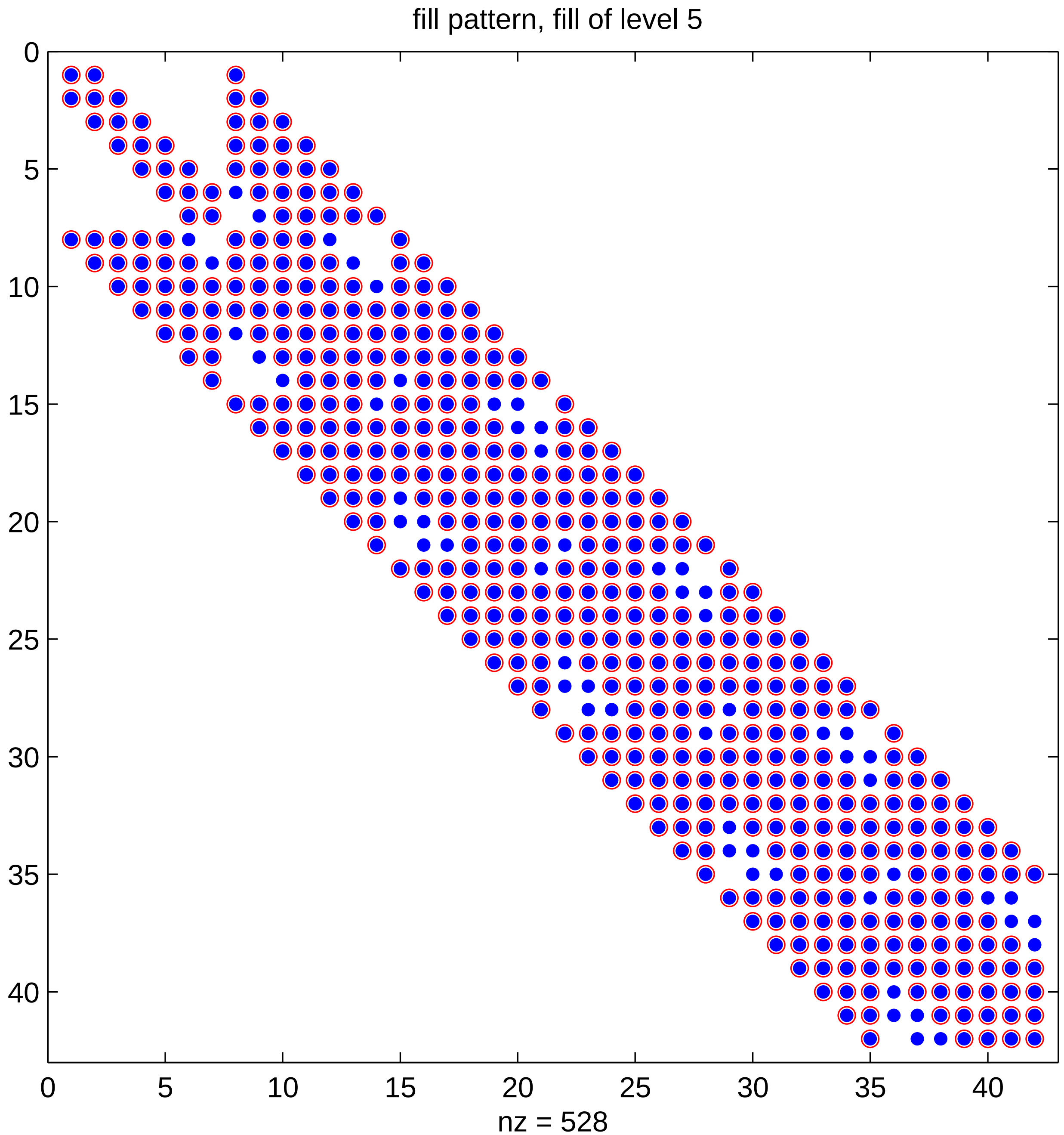
fill pattern, fill of level 0
0
5
10
15
20
25
30
35
40
0
5
10
15
20
25
30
35
40
nz = 184


fill pattern, fill of level 1
0
5
10
15
20
25
30
35
40
nz = 244

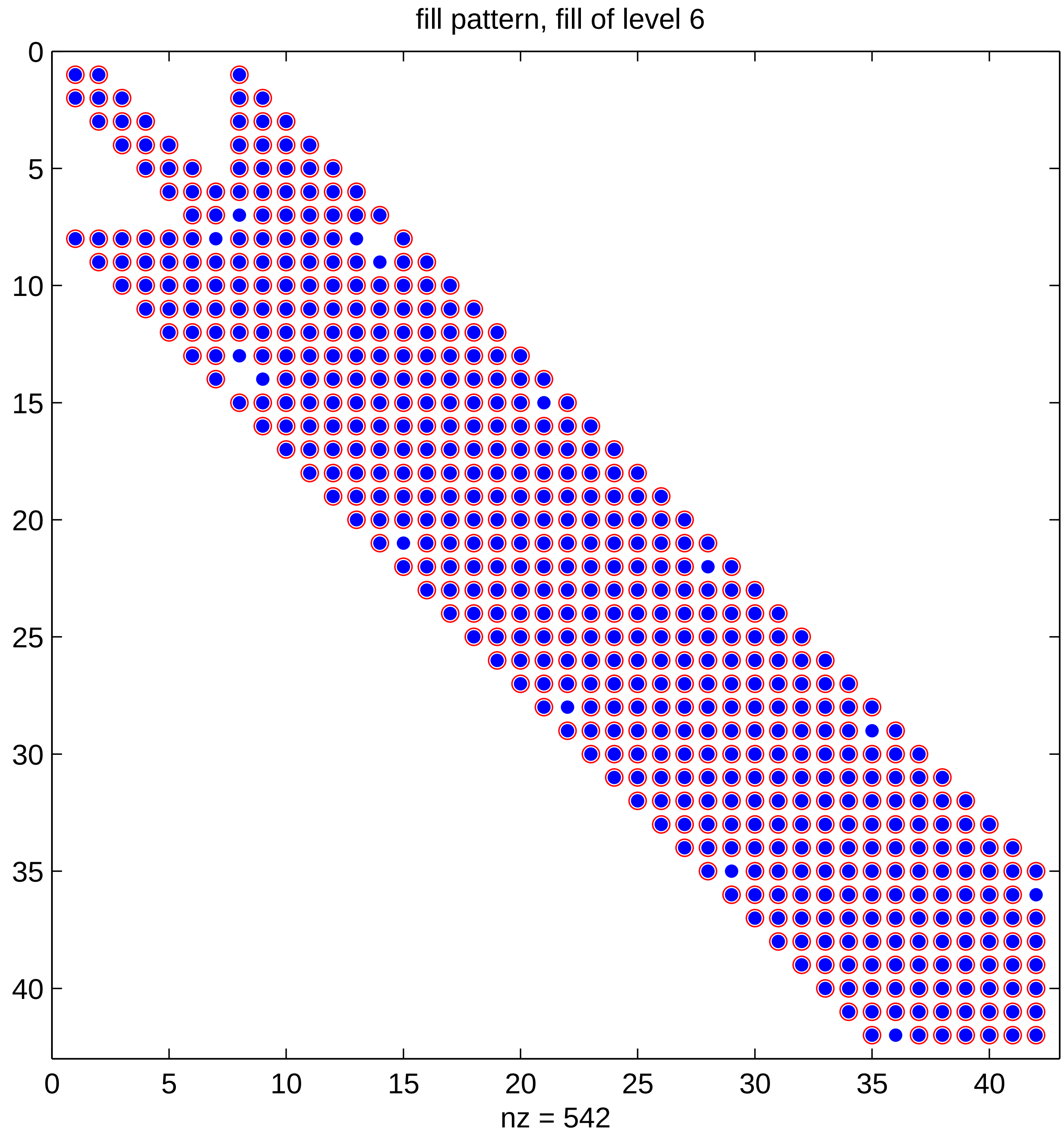

fill pattern, fill of level 2
0
5
10
15
20
25
30
35
40
nz = 294

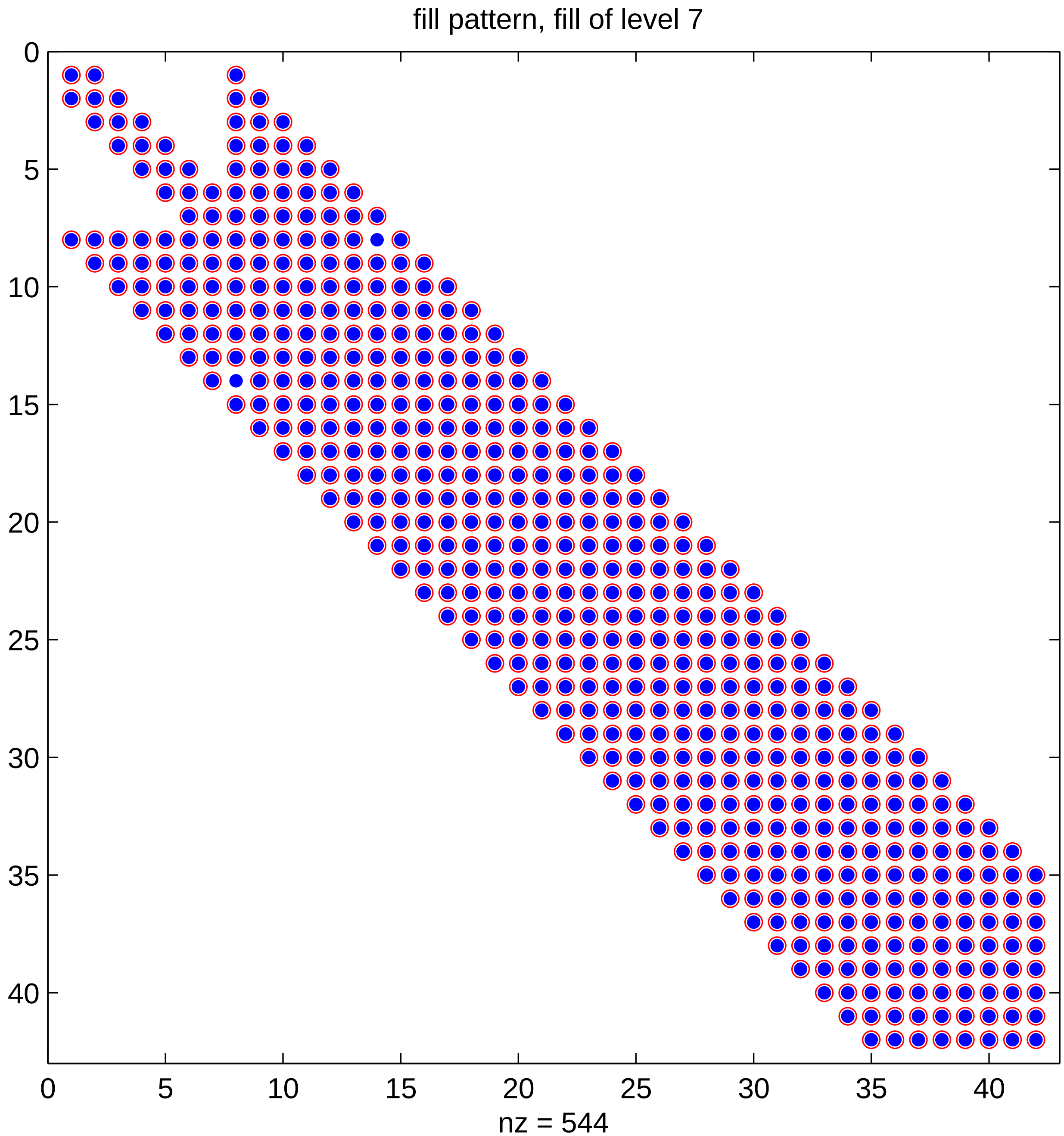

fill pattern, fill of level 3
0
5
10
15
20
25
30
35
40
nz = 384


fill pattern, fill of level 4
0
5
10
15
20
25
30
35
40
nz = 470



fill pattern, fill of level 5
0
5
10
15
20
25
30
35
40
nz = 528


fill pattern, fill of level 6
0
5
10
15
20
25
30
35
40
nz = 542



fill pattern, fill of level 7
0
5
10
15
20
25
30
35
40
0
5
10
15
20
25
30
35
40
nz = 544

Other ILU-decompositions

Select an $\varepsilon > 0$ (**the drop tolerance**).

If $\mathbf{B}$ is an $n \times n$ matrix, then $\widetilde{\mathbf{B}}$ is the matrix with entries

$$\widetilde{\mathbf{B}}(i,j) = \mathbf{B}(i,j) \quad \text{if} \quad |\mathbf{B}(i,j)| > \varepsilon$$
$$\widetilde{\mathbf{B}}(i,j) = 0 \qquad \text{if} \quad |\mathbf{B}(i,j)| \leq \varepsilon$$

Put $\quad \Pi_\varepsilon(\mathbf{B}) \equiv \widetilde{\mathbf{B}}$.

Using Π_ε in each step of the Gaussian elimination process leads to ILU(ε), **ILU with drop tolerance**

Advanced ILU.

- Drop tolerance and level strategies can be combined.
- The value of the drop tolerance can be selected
to depend on the level, on the size of the matrix entries,
- . . .

Program

- Flexible GCR
- Preconditioning
- D-ILU
- Incomplete LU-decomposition
- Why preconditioning?
- Costs
- How to include a preconditioner
- Savings

Why preconditioning?

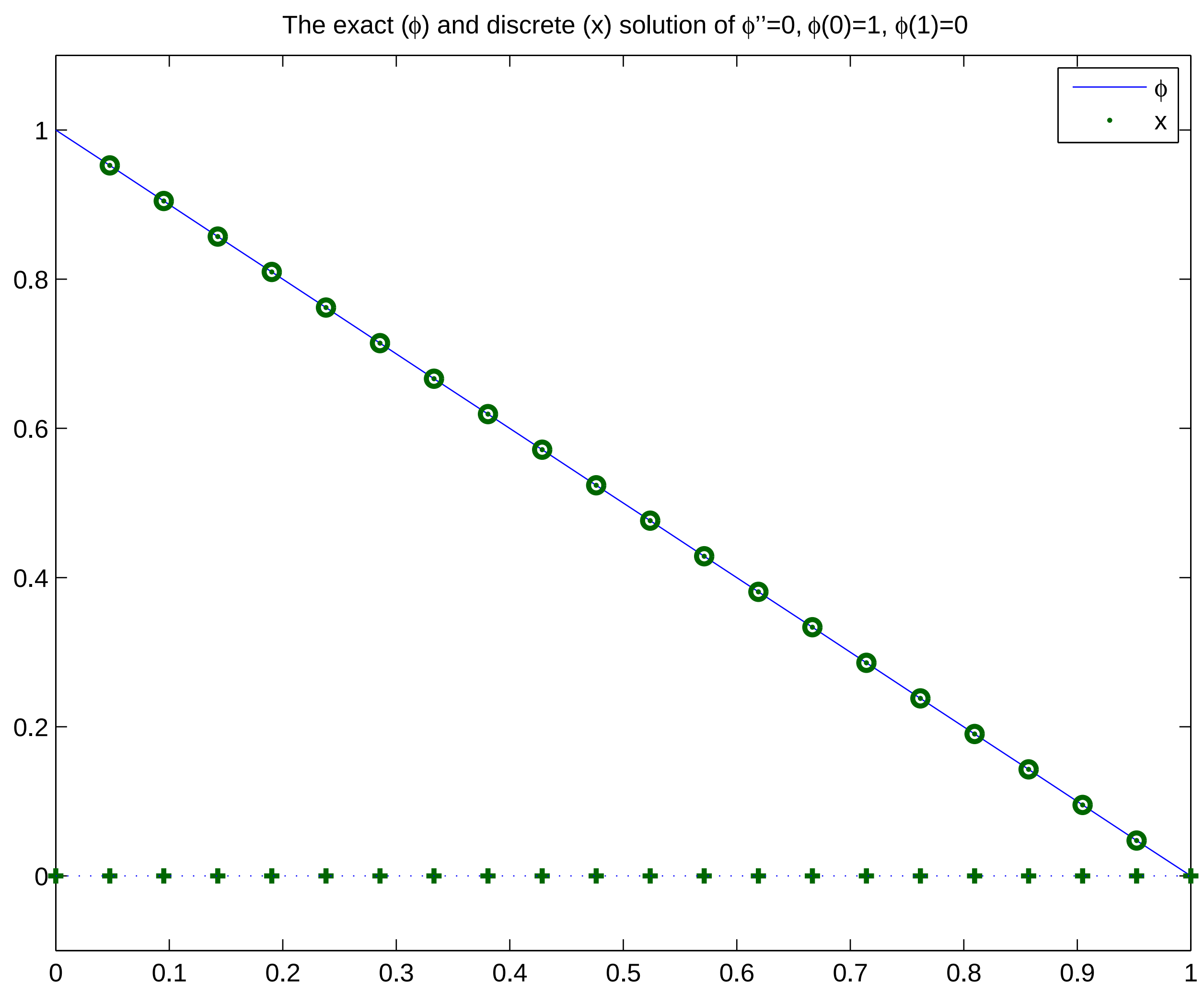
Example.

$$\begin{cases} -\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$$

Why preconditioning?

Example. $$\begin{cases} -\dfrac{\partial}{\partial x}\dfrac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$$

Exact solution $\phi(x) = 1 - x$.


The exact (ϕ) and discrete (x) solution of ϕ''=0, ϕ(0)=1, ϕ(1)=0
ϕ
x
0
0.2
0.4
0.6
0.8
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

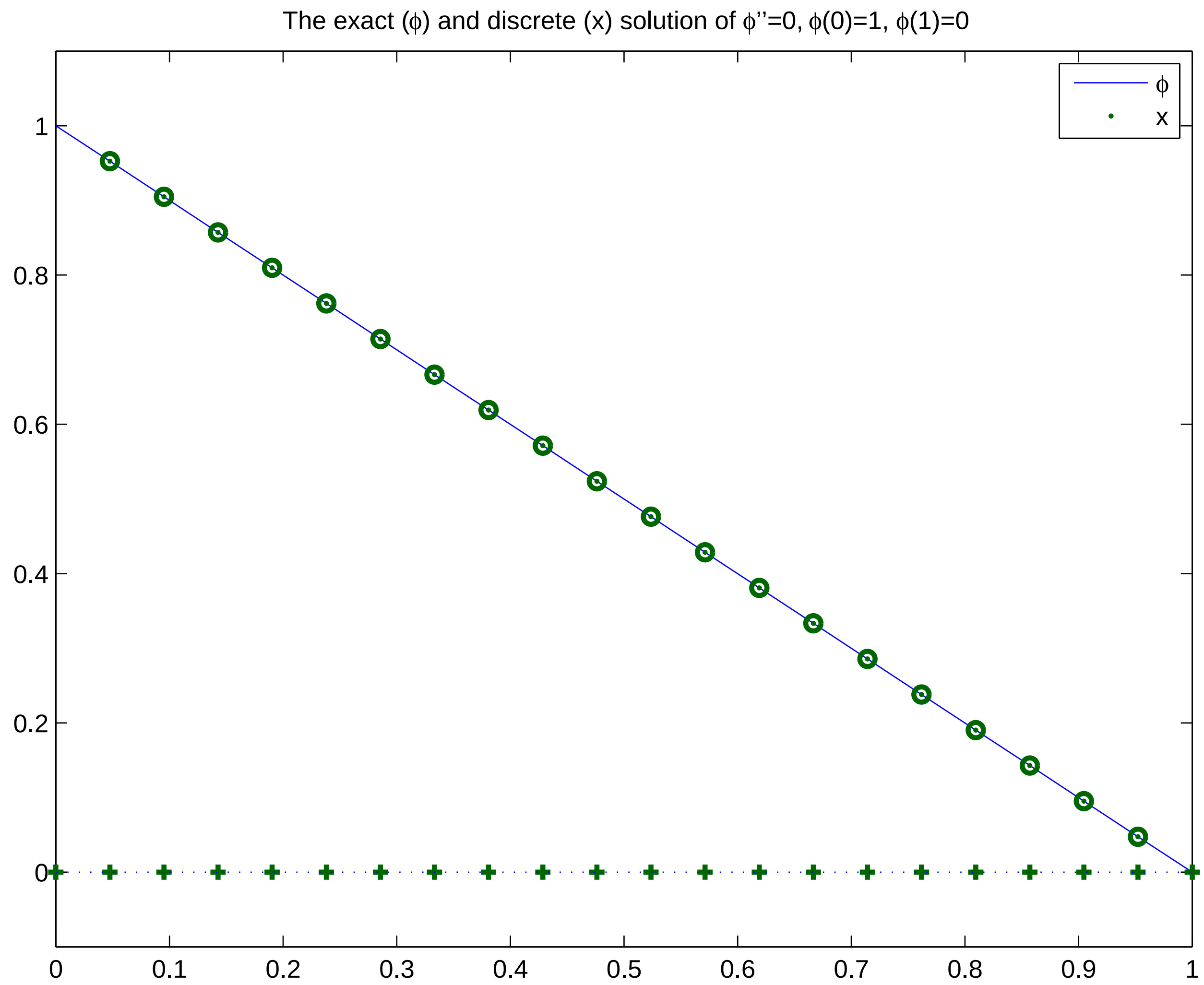
Why preconditioning?

Example.
$$\begin{cases} -\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$$

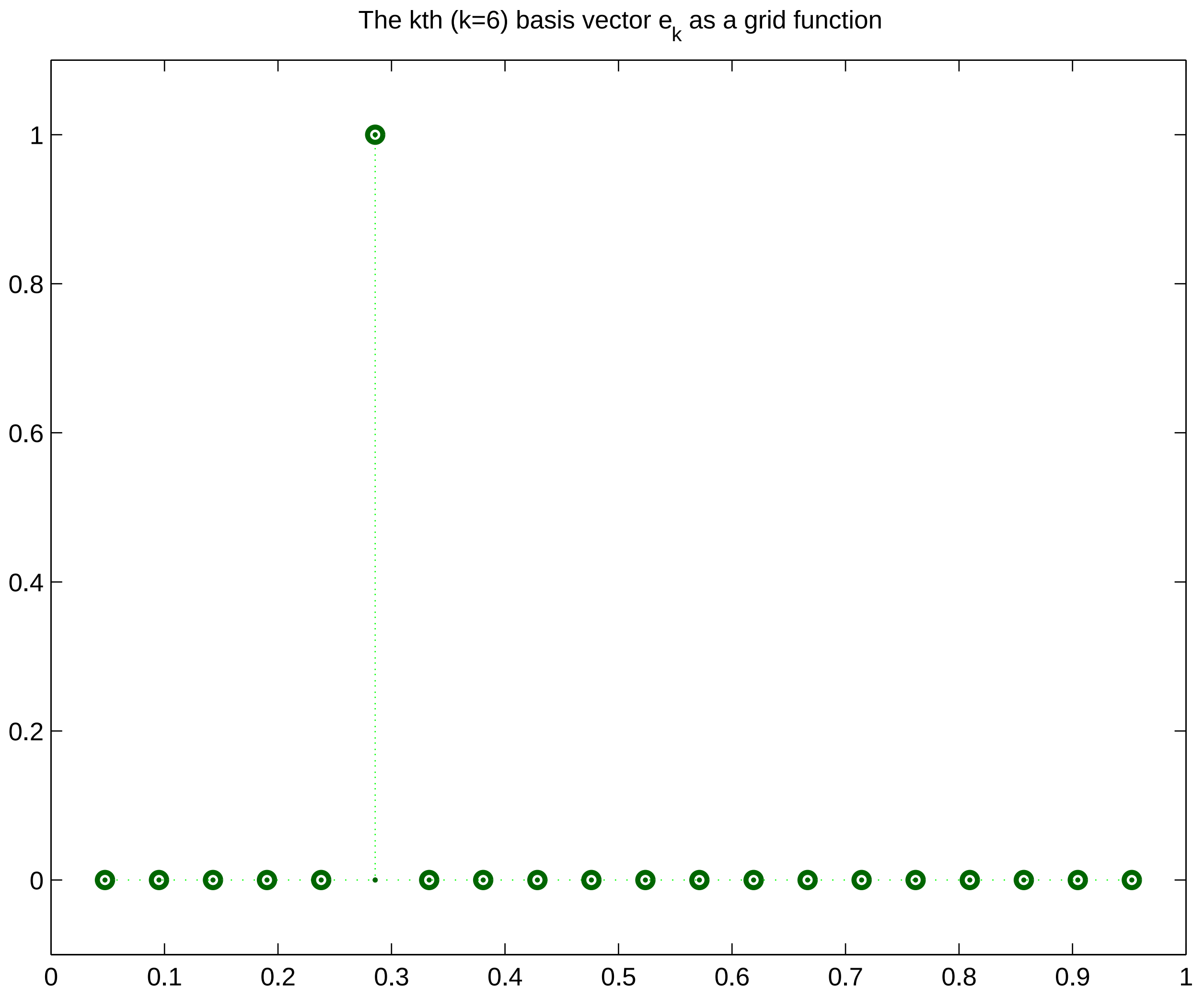
Discretization: symmetric finite differences: $h = \frac{1}{n+1}$

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & \ddots & -1 & 2 & -1 \\ & & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \frac{1}{h^2}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

Solve with GCR (or LMR) with $\mathbf{x}_0 = \mathbf{0}$.



The exact (ϕ) and discrete (x) solution of ϕ''=0, ϕ(0)=1, ϕ(1)=0
ϕ
x
1
0.8
0.6
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



Why preconditioning?

Example.
$$\begin{cases} -\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$$

Discretization: symmetric finite differences: $h = \frac{1}{n+1}$

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & \ddots & -1 & 2 & -1 \\ & & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \frac{1}{h^2}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

With $\mathbf{x}_0 = \mathbf{0}$, we have $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{b} = \tau\mathbf{e}_1$

Why preconditioning?

Example. $$\begin{cases} -\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$$

Discretization: symmetric finite differences: $h = \frac{1}{n+1}$

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & \ddots & -1 & 2 & -1 \\ & & 0 & -1 & 2 \end{bmatrix}, \qquad \mathbf{b} = \frac{1}{h^2}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

With $\mathbf{x}_0 = \mathbf{0}$, we have $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{b} = \tau\mathbf{e}_1$

Observation. For $k = 1, 2, \dots, n-1$, we have that

$$\mathbf{A}(\text{span}(\mathbf{e}_1, \dots, \mathbf{e}_k)) \subset \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_k, \mathbf{e}_{k+1}).$$

Why preconditioning?

Example.
$$\begin{cases} -\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$$

Discretization: symmetric finite differences: $h = \frac{1}{n+1}$

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & \ddots & -1 & 2 & -1 \\ & & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \frac{1}{h^2}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

With $\mathbf{x}_0 = \mathbf{0}$, we have $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{b} = \tau\mathbf{e}_1$

$\mathbf{u}_0 = \mathbf{r}_0$, $\mathbf{c}_1 = \mathbf{A}\mathbf{u}_0 = \mathbf{A}\mathbf{r}_0$, $\mathbf{x}_1 \in \text{span}(\mathbf{e}_1)$, $\mathbf{r}_1 = \mathbf{r}_0 - \alpha_1\mathbf{c}_1$

Why preconditioning?

Example.
$$\begin{cases} -\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$$

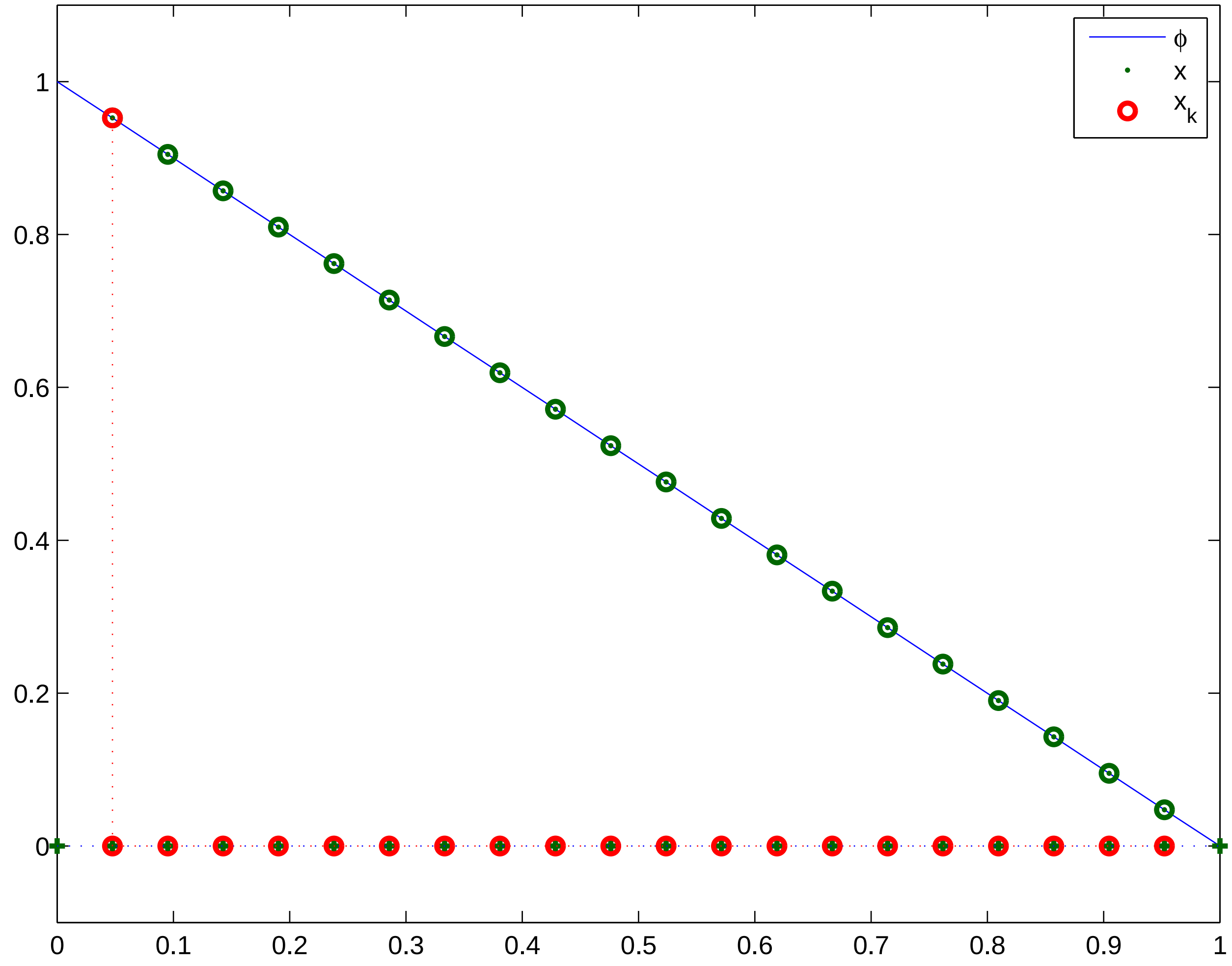
Discretization: symmetric finite differences: $h = \frac{1}{n+1}$

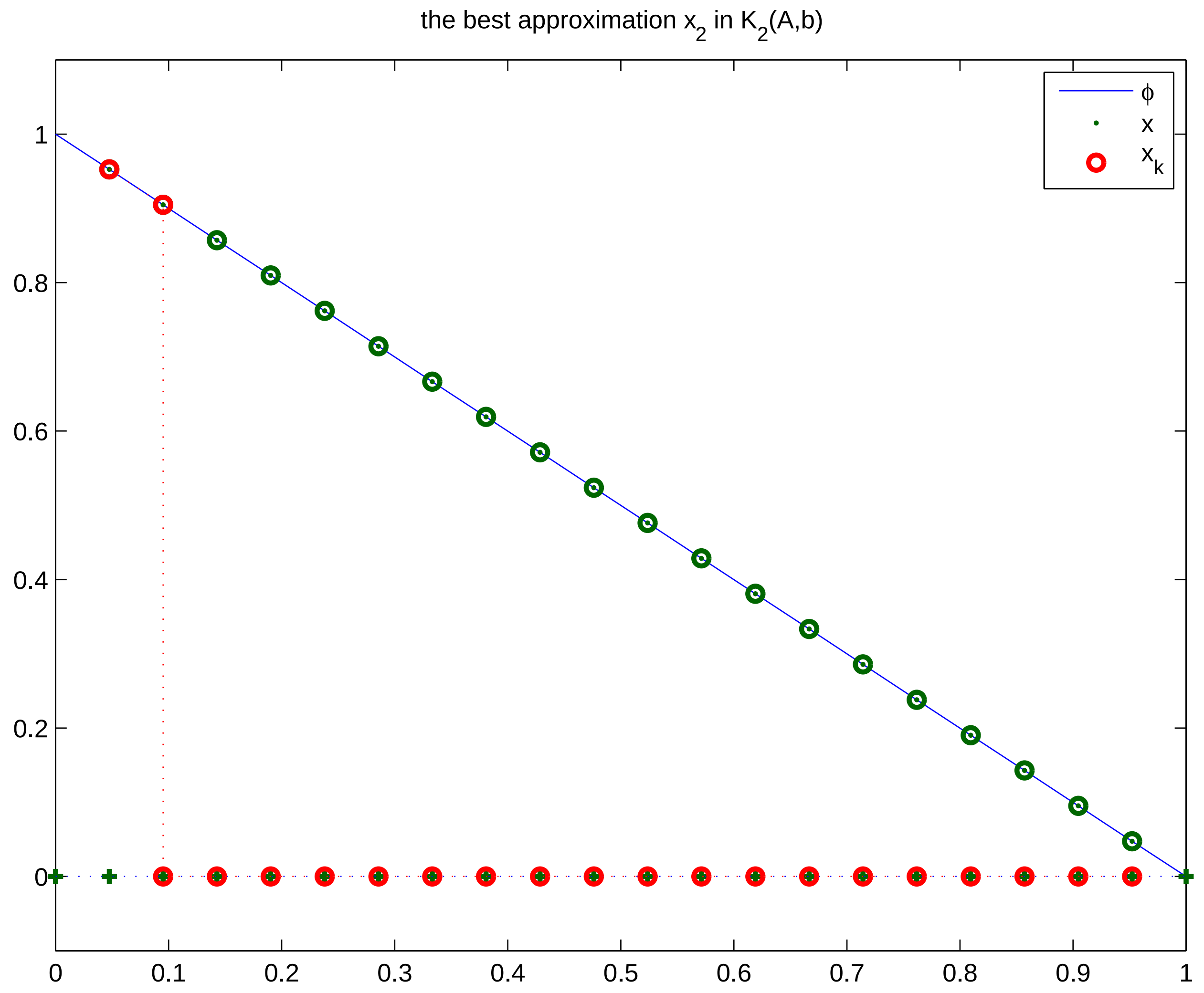
$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & \ddots & -1 & 2 & -1 \\ & & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \frac{1}{h^2}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

With $\mathbf{x}_0 = \mathbf{0}$, we have $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{b} = \tau\mathbf{e}_1$

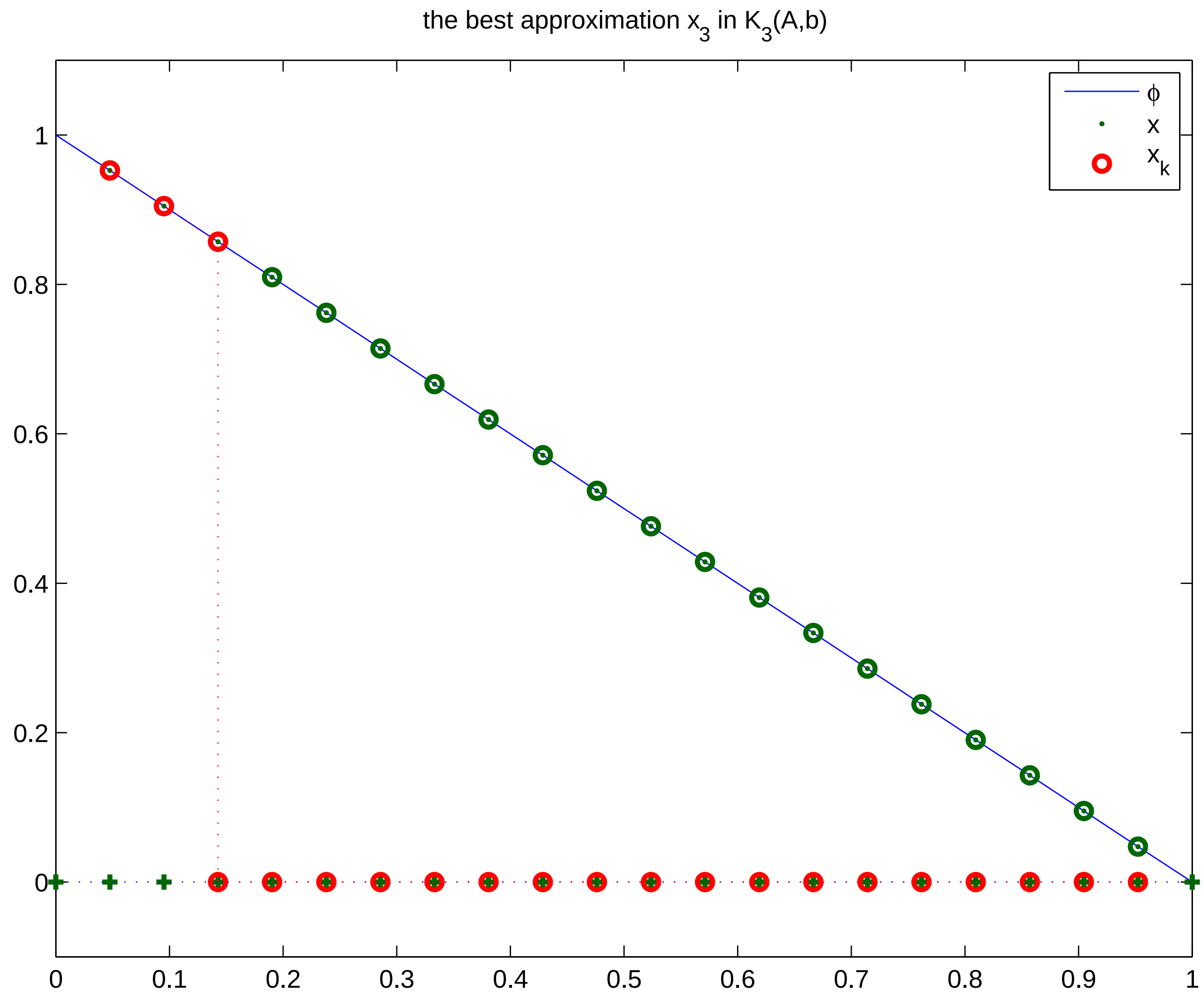
GCR and LMR: $\mathbf{r}_{k-1}, \mathbf{x}_k \in \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_k)$ for $k = 1, \dots, n$.

the best approximation x_1 in $K_1(A,b)$

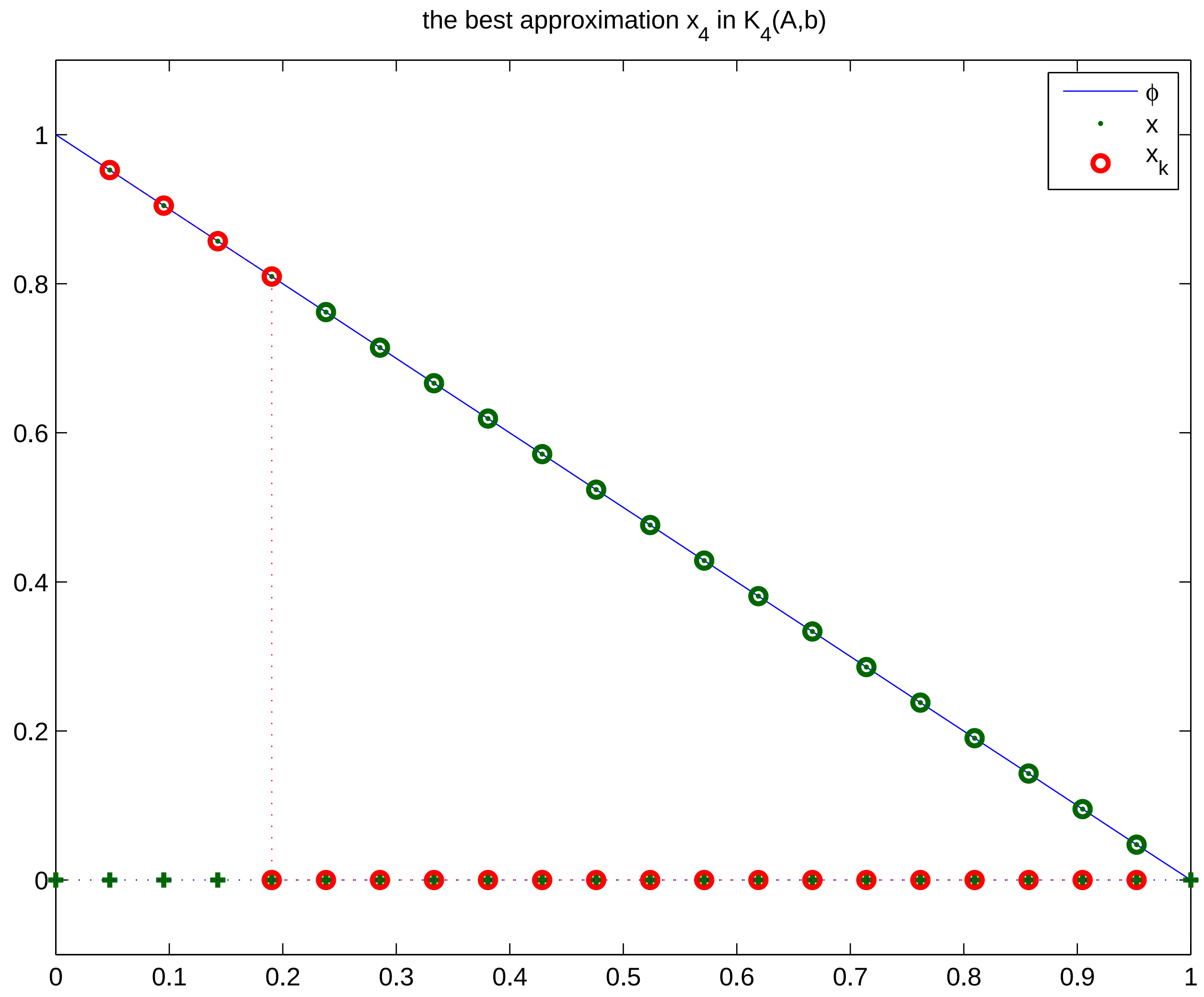




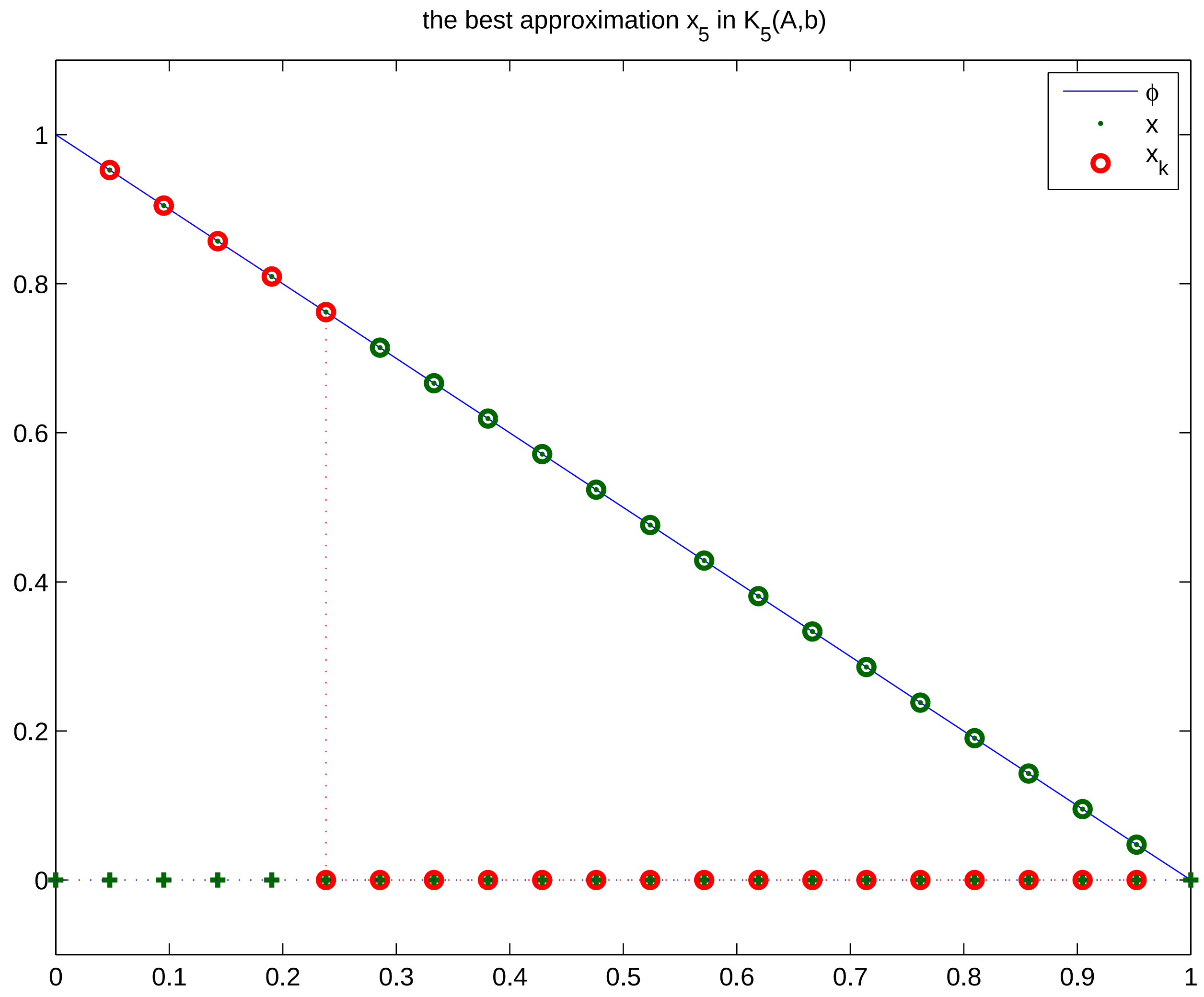
the best approximation x2 in K2(A,b)
ϕ
x
xk
1
0.8
0.6
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

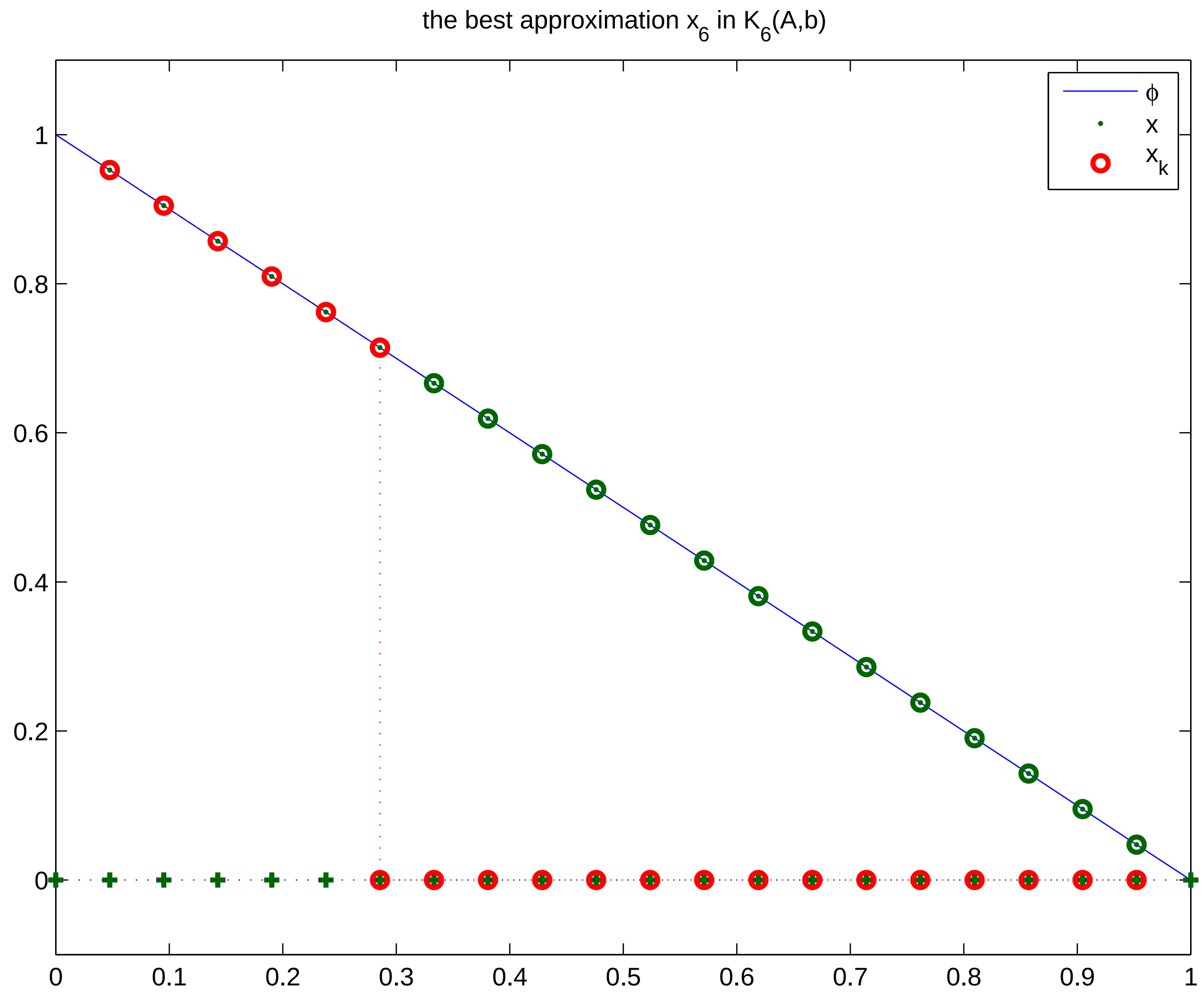


the best approximation x_3 in $K_3(A,b)$
φ
x
x_k
1
0.8
0.6
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

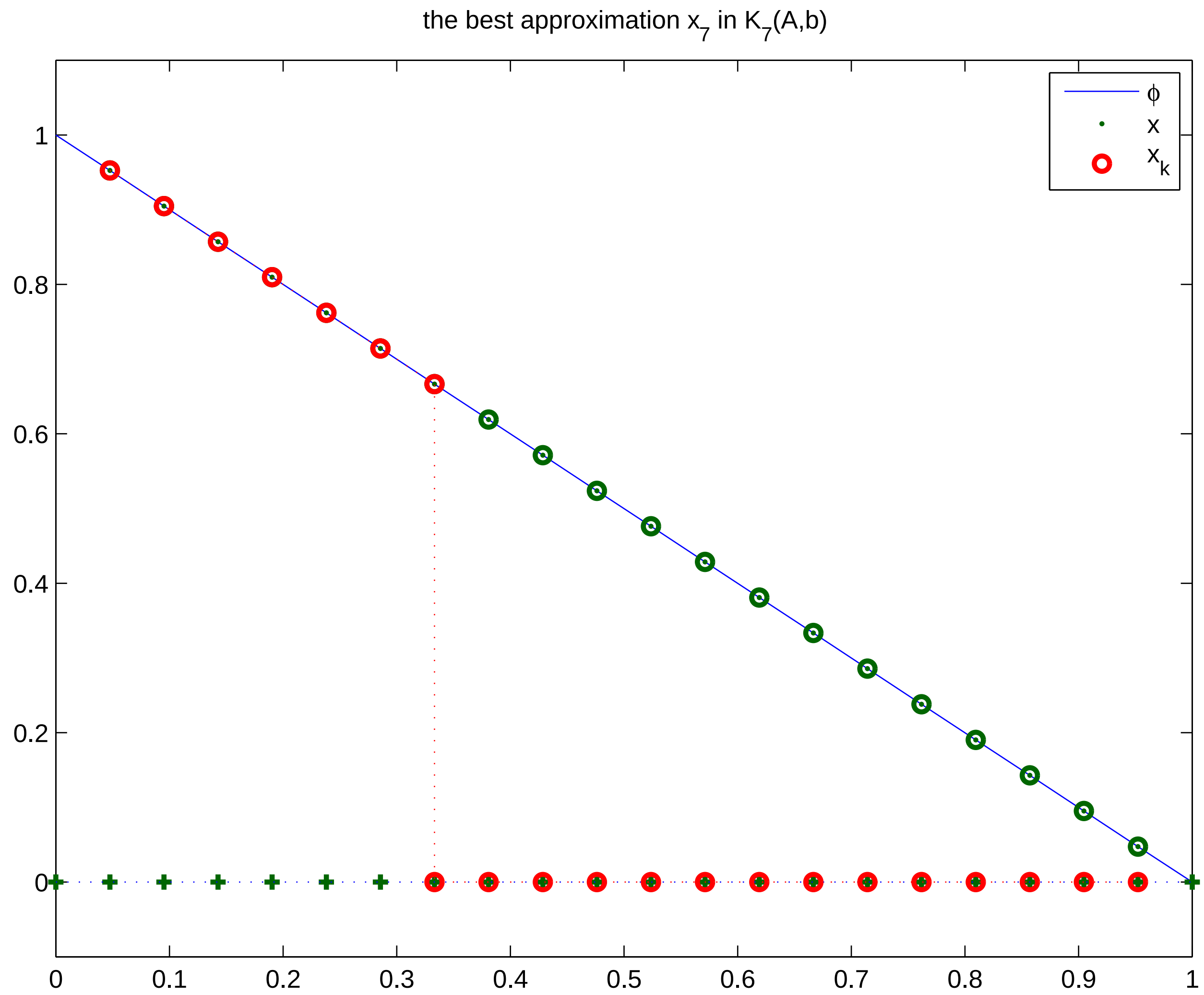


the best approximation x_4 in $K_4(A,b)$
ϕ
x
x_k
0
0.2
0.4
0.6
0.8
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


the best approximation x_5 in $K_5(A,b)$
φ
x
x_k
0
0.2
0.4
0.6
0.8
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



the best approximation x6 in K6(A,b)
ϕ
x
xk
0
0.2
0.4
0.6
0.8
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



the best approximation x_7 in $K_7(A,b)$
ϕ
x
x_k
0
0.2
0.4
0.6
0.8
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Why preconditioning?

Example.
$$\begin{cases} -\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$$

Discretization: symmetric finite differences: $h = \frac{1}{n+1}$

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & \ddots & -1 & 2 & -1 \\ & & 0 & -1 & 2 \end{bmatrix}, \qquad \mathbf{b} = \frac{1}{h^2}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

With $\mathbf{x}_0 = \mathbf{0}$, we have $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{b} = \tau\mathbf{e}_1$

GCR and LMR: $\mathbf{r}_{k-1}, \mathbf{x}_k \in \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_k)$ for $k = 1, \dots, n$.

The best solution in span$(\mathbf{e}_1, \dots, \mathbf{e}_k)$ is (for $k < n$)

$$\sum_{j \le k}(1 - \tfrac{j}{n+1})\mathbf{e}_j, \qquad \text{2-norm error } \tfrac{n-k}{n+1} \ge \tfrac{1}{n+1}.$$

Why preconditioning?

Example. $\begin{cases} -\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$

Discretization: symmetric finite differences: $h = \frac{1}{n+1}$

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & \ddots & -1 & 2 & -1 \\ & & 0 & -1 & 2 \end{bmatrix}, \qquad \mathbf{b} = \frac{1}{h^2}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

With $\mathbf{x}_0 = \mathbf{0}$, we have $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{b} = \tau\mathbf{e}_1$

GCR and LMR: $\mathbf{r}_{k-1}, \mathbf{x}_k \in \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_k)$ for $k = 1, \dots, n$.

The best solution in span$(\mathbf{e}_1, \dots, \mathbf{e}_k)$ is (for $k < n$)

$$\sum_{j \le k}(1 - \tfrac{j}{n+1})\mathbf{e}_j, \qquad \text{2-norm error } \tfrac{n-k}{n+1} \ge \tfrac{1}{n+1}.$$

Conclusion. The error can not drop below h in $< n$ steps.

Why preconditioning?

Example.
$$\begin{cases} -\frac{\partial}{\partial x}\frac{\partial}{\partial x}\phi = 0 \quad \text{on} \quad D \equiv [0,1] \\ \phi(0) = 1, \quad \phi(1) = 0 \end{cases}$$

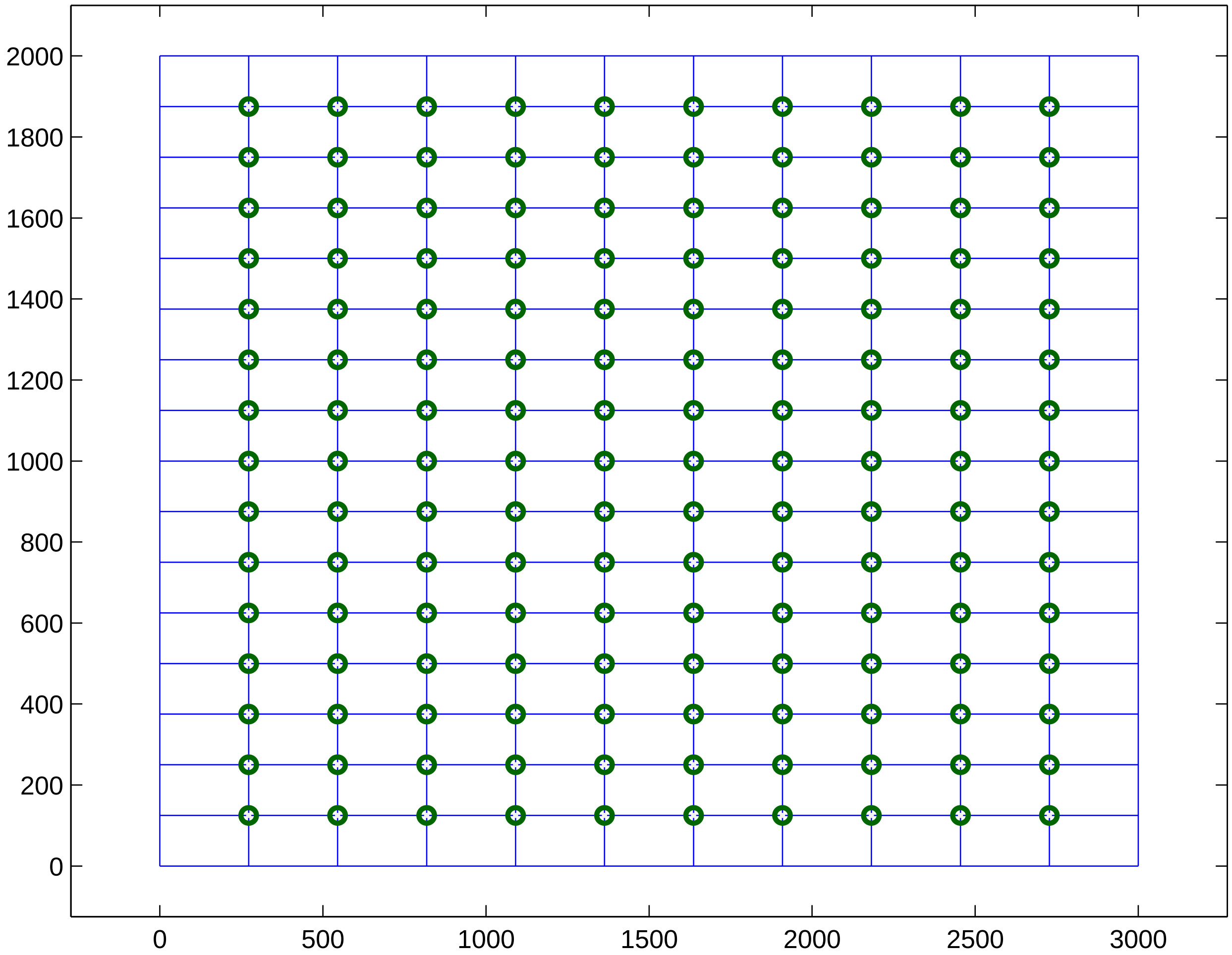
Discretization: symmetric finite differences: $h = \frac{1}{n+1}$

$$\mathbf{A} = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & \ddots & -1 & 2 & -1 \\ & & 0 & -1 & 2 \end{bmatrix}, \qquad \mathbf{b} = \frac{1}{h^2}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}.$$

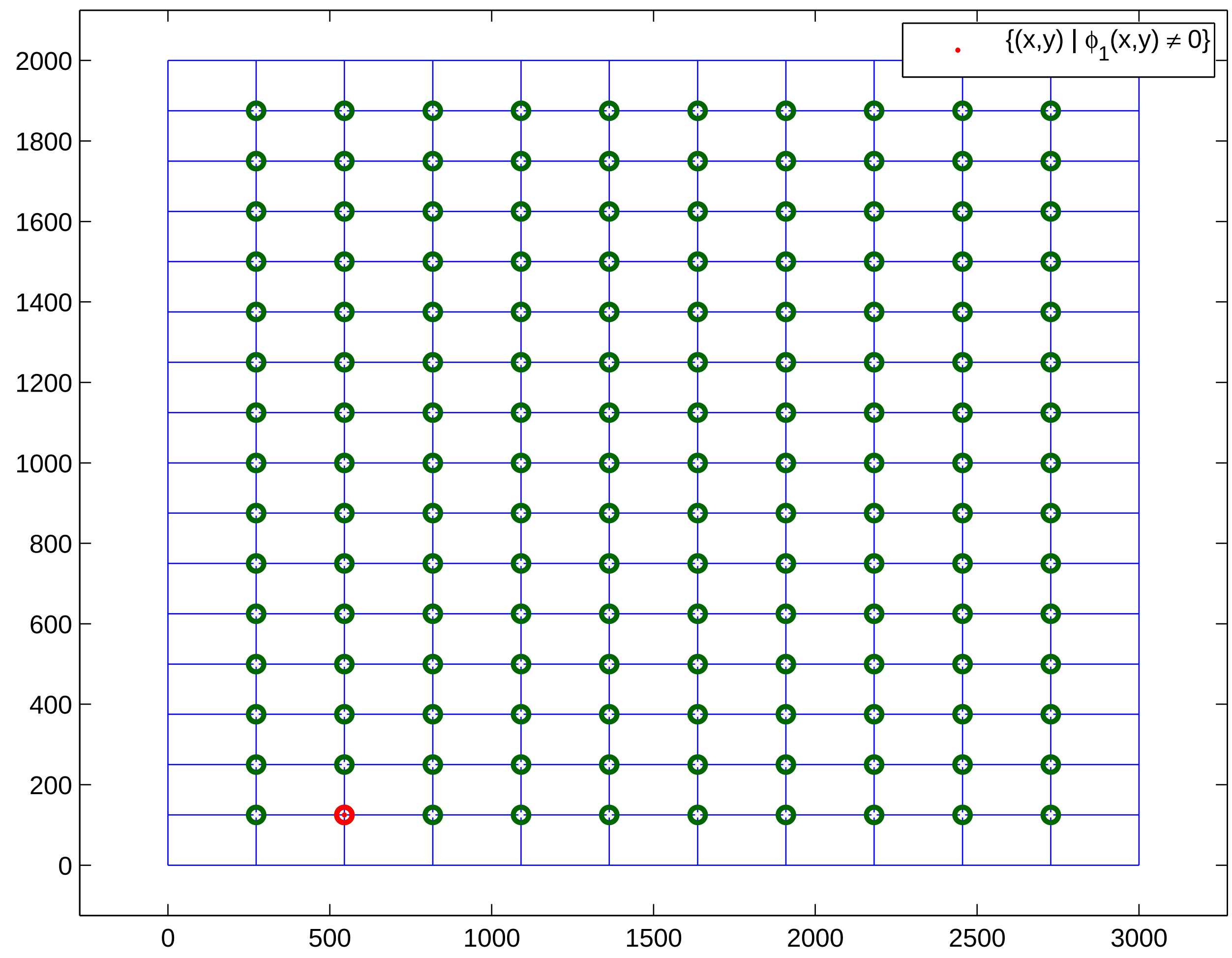
With $\mathbf{x}_0 = \mathbf{0}$, we have $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{b} = \tau\mathbf{e}_1$

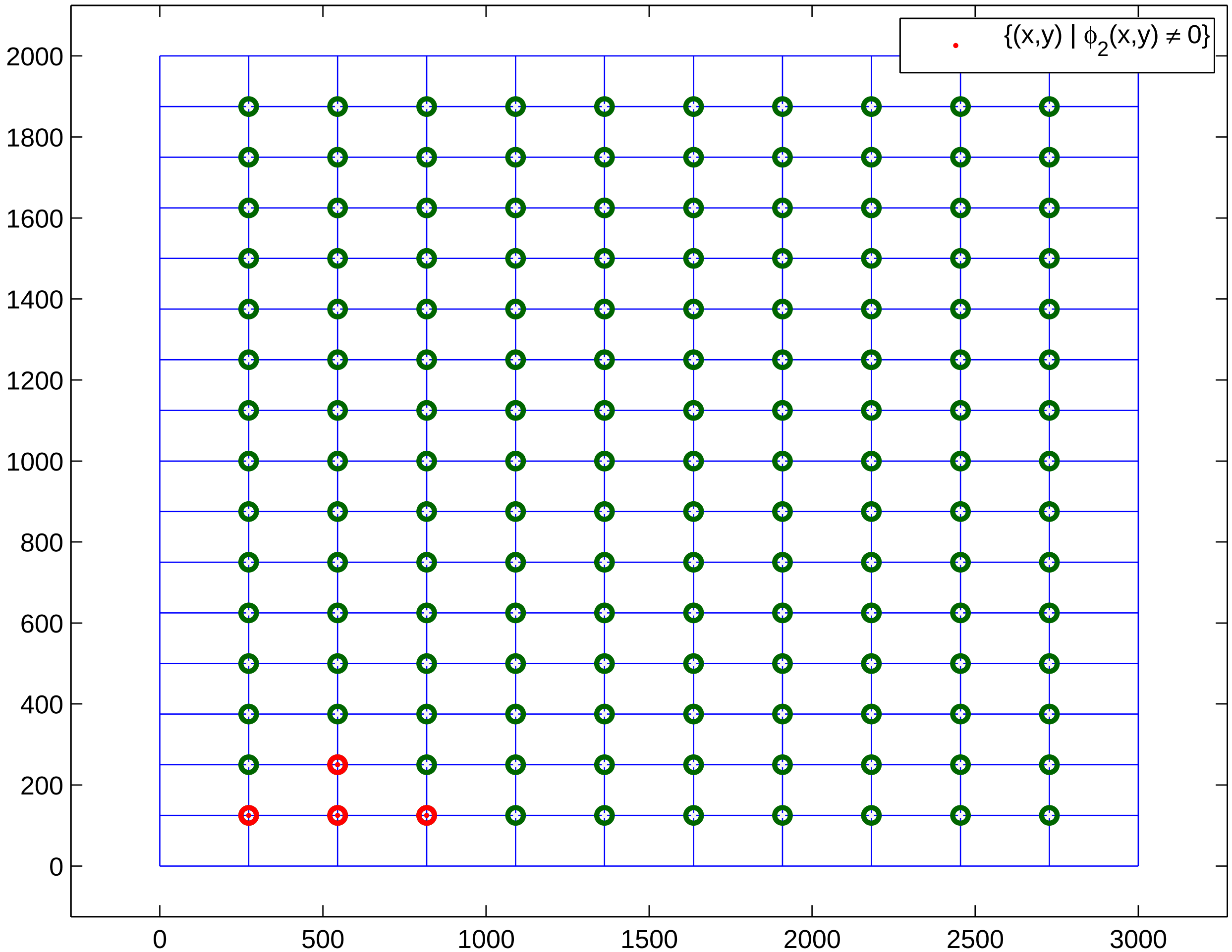
GCR and LMR: $\mathbf{r}_{k-1}, \mathbf{x}_k \in \text{span}(\mathbf{e}_1, \dots, \mathbf{e}_k)$ for $k = 1, \dots, n$.

Interpretation. It takes a Krylov subspace method at least n (=grid size) steps to carry the information in $\mathbf{r}_0$ over the whole grid.

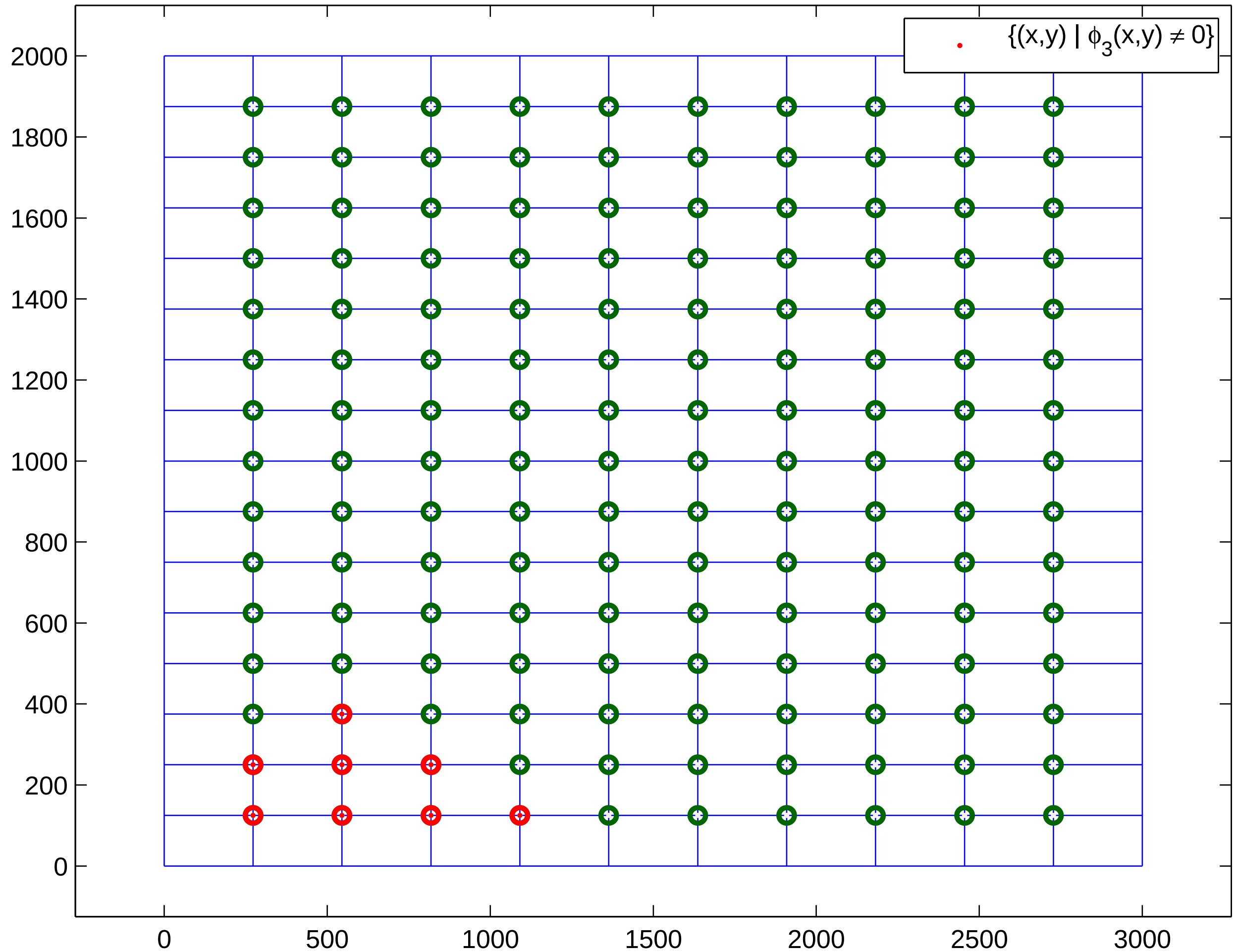


2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000

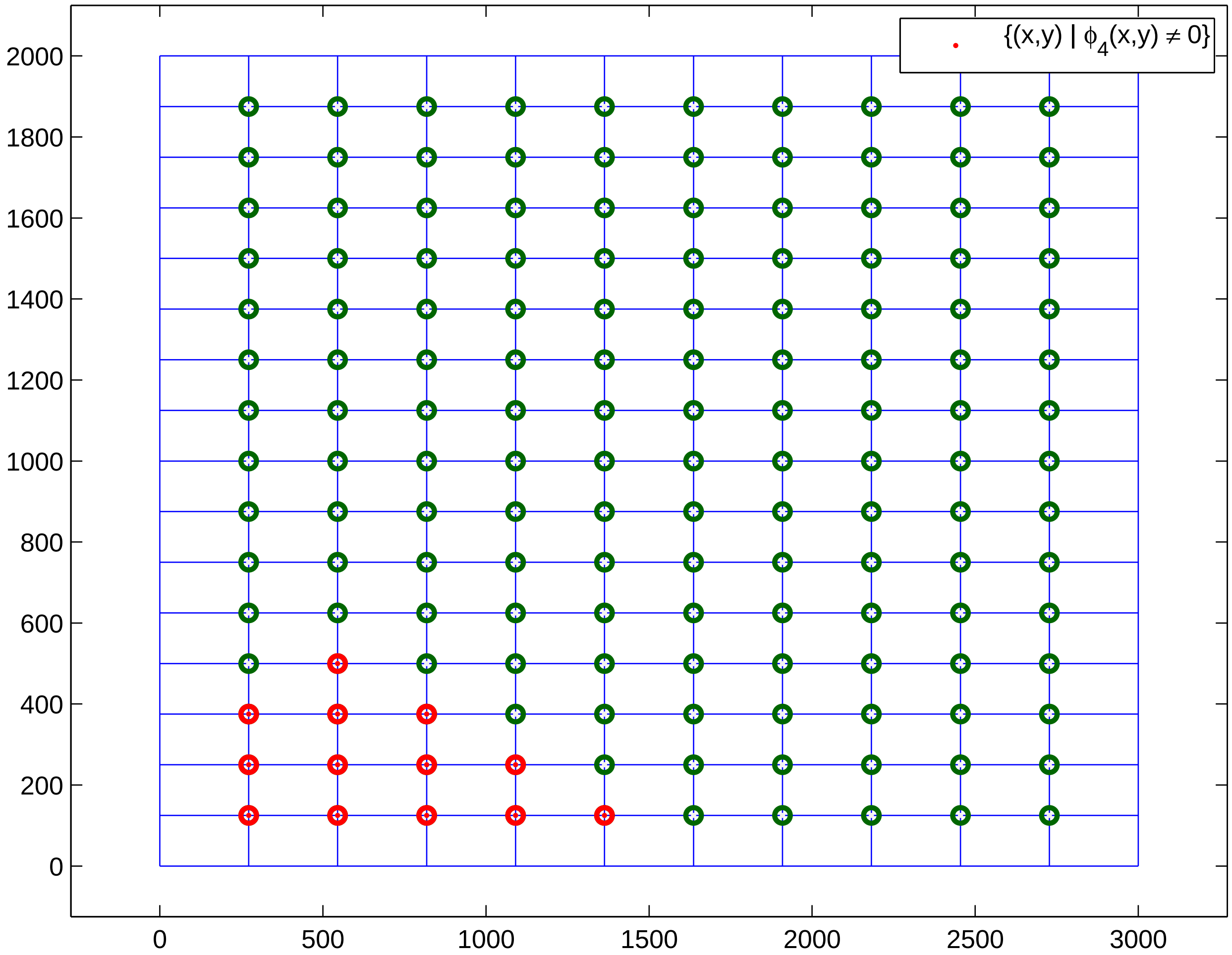

{(x,y) | ϕ_1(x,y) ≠ 0}
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000

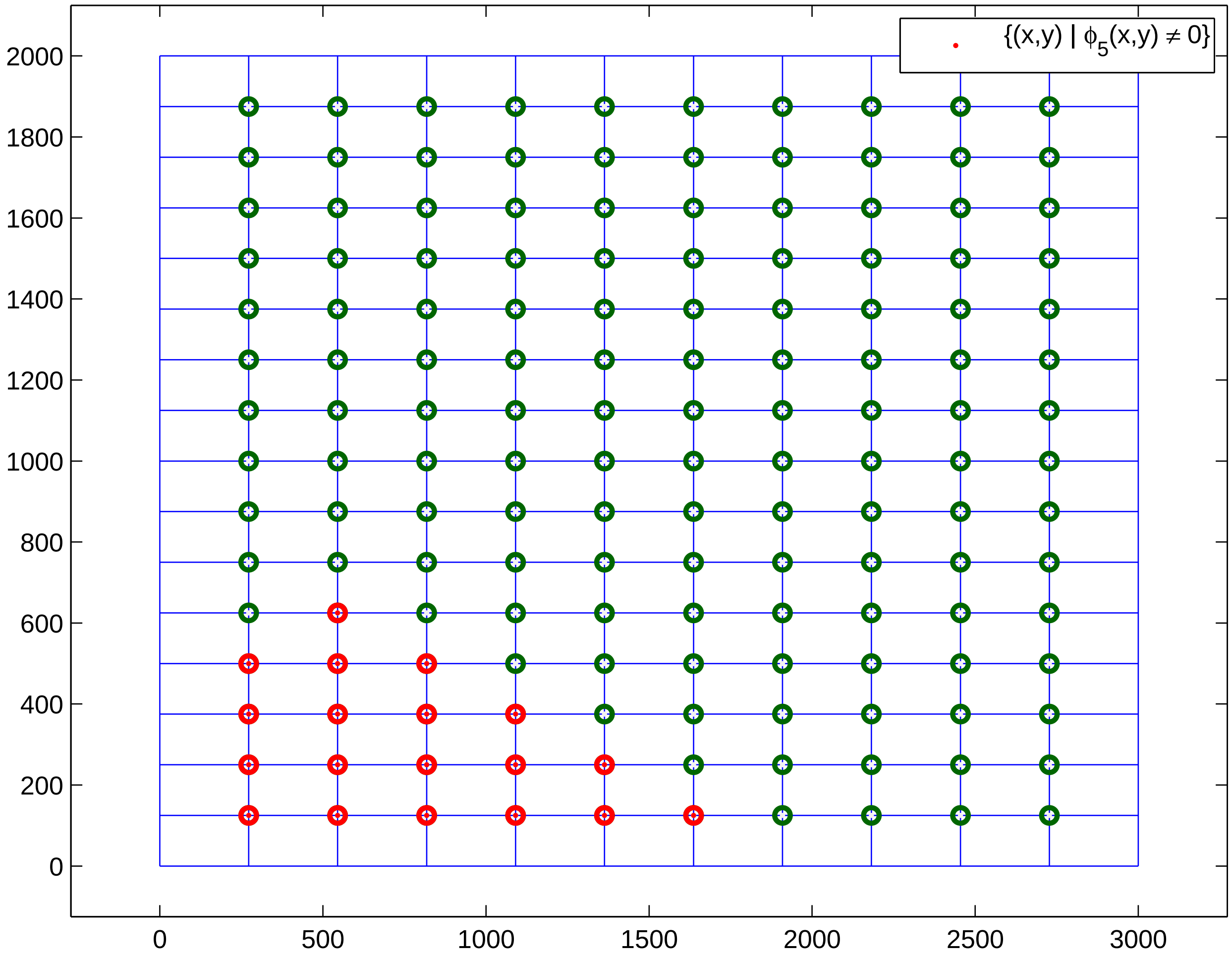


{(x,y) | ϕ_2(x,y) ≠ 0}
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000

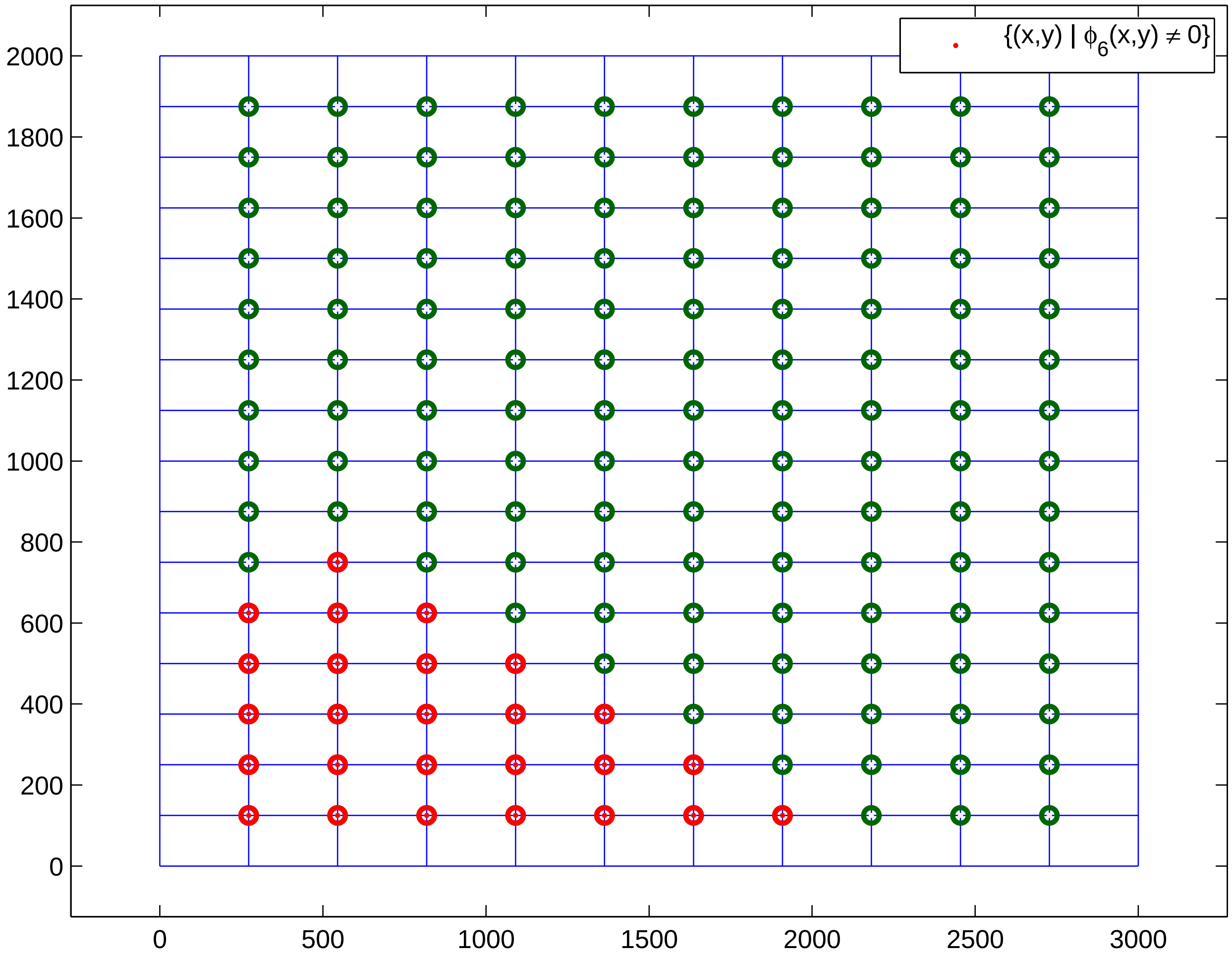


{(x,y) | ϕ_3(x,y) ≠ 0}
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000

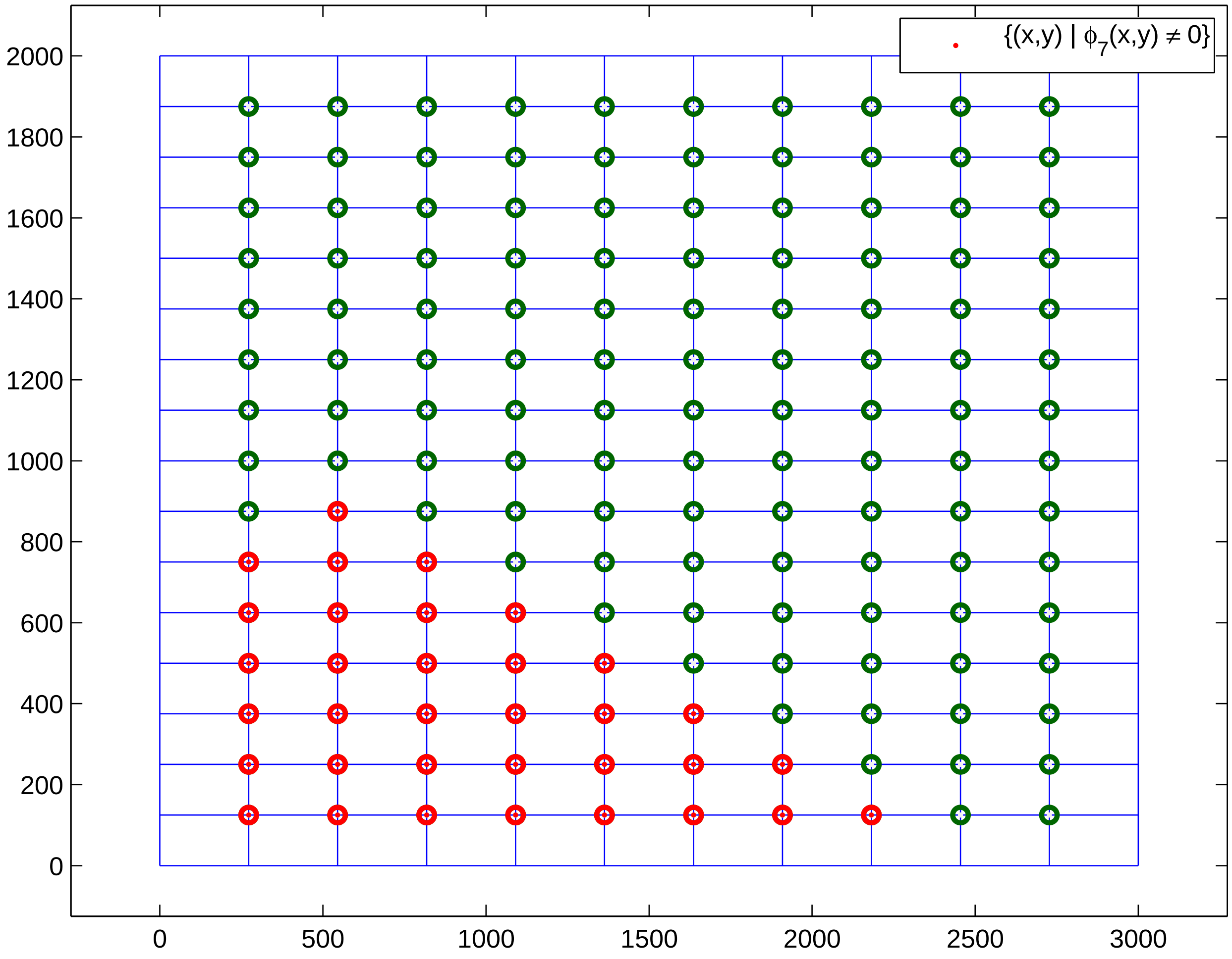

{(x,y) | ϕ4(x,y) ≠ 0}
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000



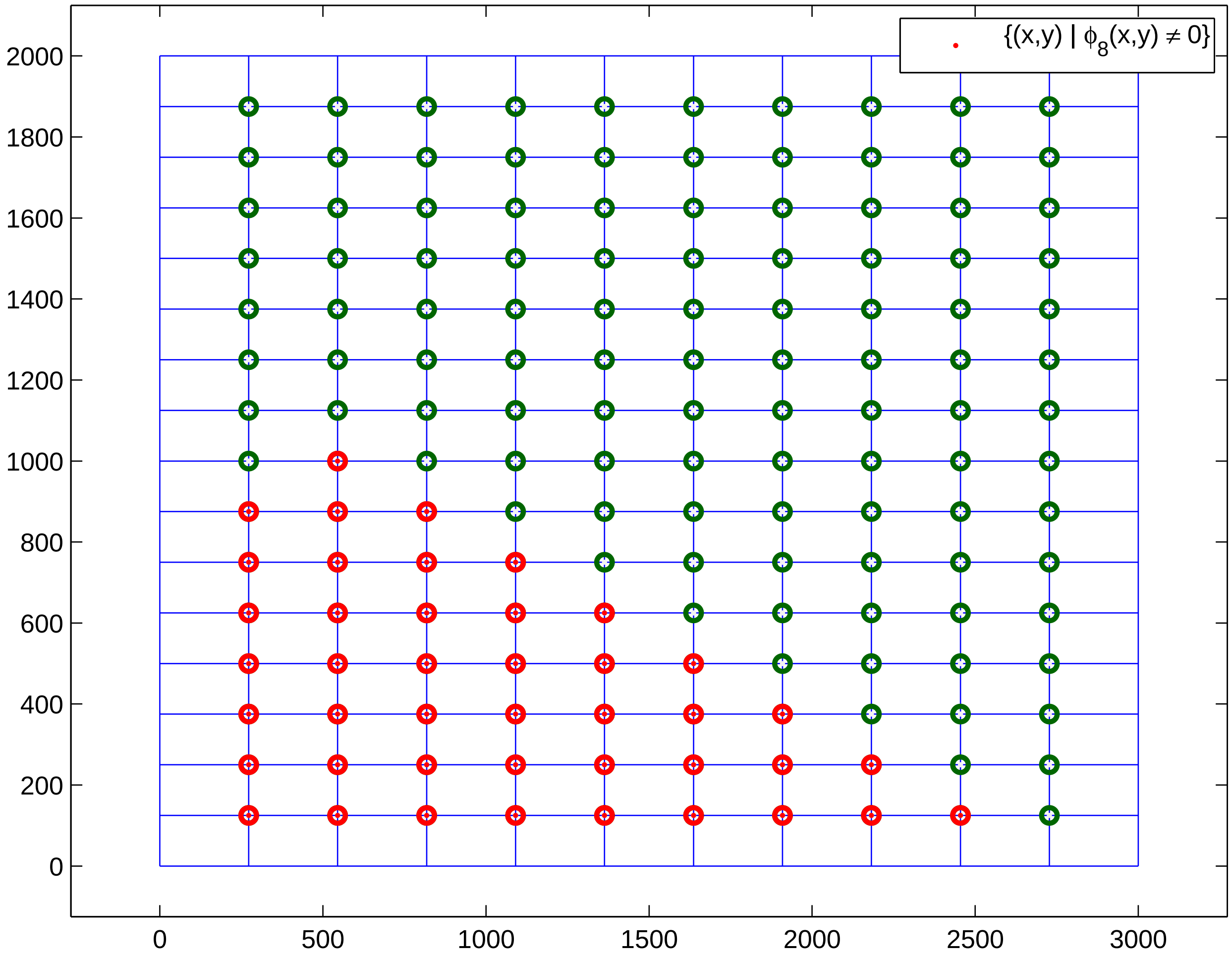
{(x,y) | ϕ_5(x,y) ≠ 0}
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000



{(x,y) | φ6(x,y) ≠ 0}
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000



{(x,y) | ϕ_7(x,y) ≠ 0}
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000



{(x,y) | ϕ_8(x,y) ≠ 0}
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
500
1000
1500
2000
2500
3000

Why preconditioning?

Interpretation. **Without preconditioning.**
In d-dimensional advection diffusion problems, with symmetric finite difference discretization of order 2:

It takes any Krylov subspace method at least max$(n_x, n_y, \ldots)$ (=max. grid size) steps to carry the information in $\mathbf{r}_0$ over the whole grid.

No small error, whence no small residual, can be expected in less than max$(n_x, n_y, \ldots)$ steps with a Krylov subspace method.

Why preconditioning?

Interpretation. Without preconditioning.

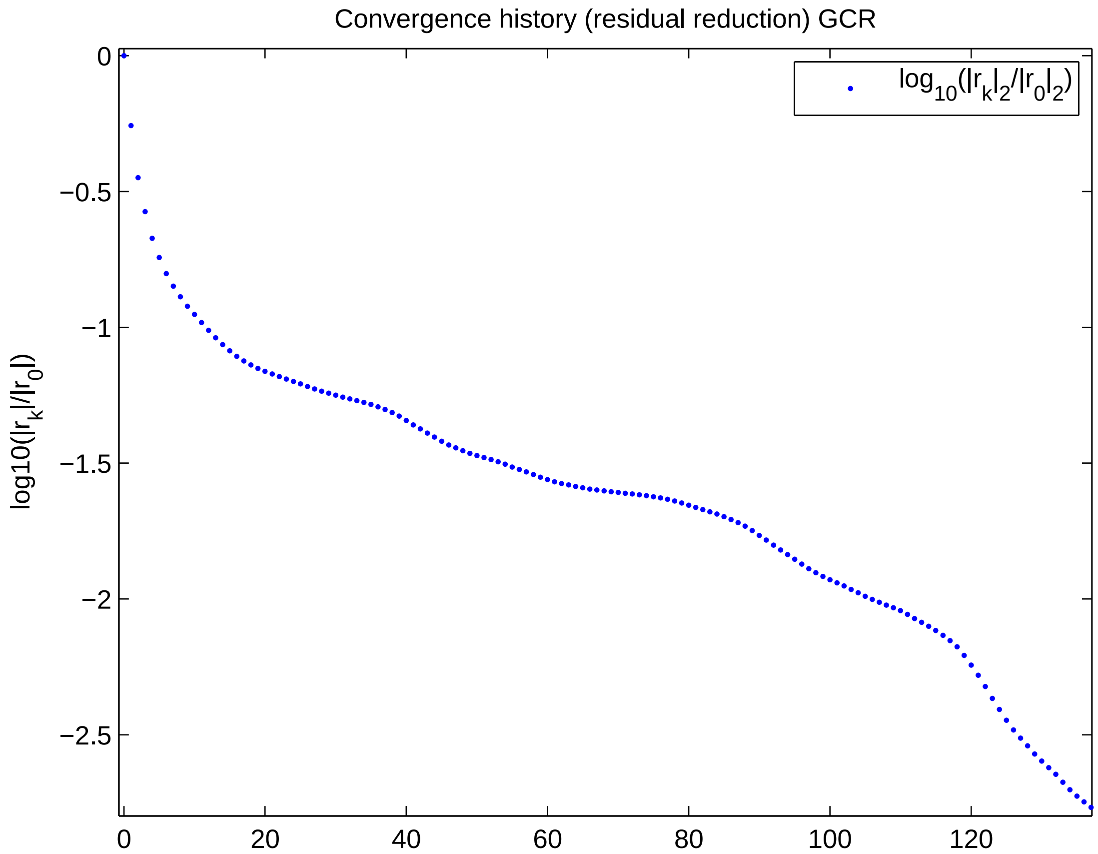
In d-dimensional advection diffusion problems, with symmetric finite difference discretization of order 2:

It takes any Krylov subspace method at least max$(n_x, n_y, \ldots)$ (=max. grid size) steps to carry the information in $\mathbf{r}_0$ over the whole grid.

No small error, whence no small residual, can be expected in less than max$(n_x, n_y, \ldots)$ steps with a Krylov subspace method.

If from 1-d advection diffusion,
one 'sweep' over the grid solves the problem.
In higher dimensions, more sweeps are needed.

GCR without preconditioning on a 40 by 30 grid.



Why preconditioning?

Interpretation. Without preconditioning.
In d-dimensional advection diffusion problems, with symmetric finite difference discretization of order 2:

It takes any Krylov subspace method at least max$(n_x, n_y, \ldots)$ (=max. grid size) steps to carry the information in $\mathbf{r}_0$ over the whole grid.

No small error, whence no small residual, can be expected in less than max$(n_x, n_y, \ldots)$ steps with a Krylov subspace method.

With an ILU-preconditioner, the information is "dragged" all over the grid in each step.

This does not guarantee fast convergence, but it might.

Program

- Flexible GCR
- Preconditioning
- D-ILU
- Incomplete LU-decomposition
- Why preconditioning?
- Costs
- How to include a preconditioner
- Savings

Does preconditioning harm?

$$\mathbf{M} = (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A).$$

Costs. The costs of computing $\mathbf{D}$ are negligible:

$\mathbf{D}$ has to be computed only once (before starting the solution process with GCR (or LMR)).

Does preconditioning harm?

$$\mathbf{M} = (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A).$$

Costs. The costs of computing $\mathbf{D}$ are negligible

$\mathbf{u}$ is solved from $\mathbf{M}\mathbf{u} = \mathbf{r}$ by

- `Solve` $(\mathbf{L}_A + \mathbf{D})\mathbf{u}' = \mathbf{r}$ `for` $\mathbf{u}'$
- `Compute` $\mathbf{u}'' = \mathbf{D}\mathbf{u}'$
- `Solve` $(\mathbf{U}_A + \mathbf{D})\mathbf{u} = \mathbf{u}''$ `for` $\mathbf{u}$

Does preconditioning harm?

$$\mathbf{M} = (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A).$$

Costs. The costs of computing $\mathbf{D}$ are negligible

$\mathbf{u}$ is solved from $\mathbf{M}\mathbf{u} = \mathbf{r}$ by

- `Solve` $(\mathbf{L}_A + \mathbf{D})\mathbf{u}' = \mathbf{r}$ `for` $\mathbf{u}'$ $5n$ `flop`
- `Compute` $\mathbf{u}'' = \mathbf{D}\mathbf{u}'$
- `Solve` $(\mathbf{U}_A + \mathbf{D})\mathbf{u} = \mathbf{u}''$ `for` $\mathbf{u}$

Does preconditioning harm?

$$\mathbf{M} = (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A).$$

Costs. The costs of computing $\mathbf{D}$ are negligible

$\mathbf{u}$ is solved from $\mathbf{Mu} = \mathbf{r}$ by

- `Solve` $(\mathbf{L}_A + \mathbf{D})\mathbf{u}' = \mathbf{r}$ `for` $\mathbf{u}'$ $5n$ `flop`
- `Compute` $\mathbf{u}'' = \mathbf{D}\mathbf{u}'$ n `flop`
- `Solve` $(\mathbf{U}_A + \mathbf{D})\mathbf{u} = \mathbf{u}''$ `for` $\mathbf{u}$ $5n$ `flop`

An $\mathbf{M}$-solve costs $11n$ flop.

Does preconditioning harm?

$$\mathbf{M} = (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A).$$

Costs. The costs of computing $\mathbf{D}$ are negligible

An $\mathbf{M}$-solve costs $11n$ flop.

- The extra costs in k-steps for including $\mathbf{M}$ solves in each step of GCR are $11kn$.
- Reduction costs in GCR when reducing the number if steps from $k+m$ to k is (with no precond. for 2-d) is

$$\geq (19n + 6kn)m \text{ flop.}$$

Conclusion. The total costs in GCR already reduces by including $\mathbf{M}$-solves if this leads to a reduction in the number of required steps by 2 steps (if $m \geq 2$ then $(6kn)m \geq 11kn$).

Does preconditioning harm?

$$\mathbf{M} = (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A).$$

Costs. The costs of computing **D** are negligible

An **M**-solve costs $11n$ flop.

- The extra costs in k-steps for including **M** solves in each step of LMR are $11kn$.
- Reduction costs in LMR when reducing the number if steps from $k+m$ to k is (with no precond. for 2-d) is

$$\geq 19nm \text{ flop.}$$

Conclusion. The total costs in LMR reduces by including **M**-solves if this leads to a reduction in the number of required steps by 40%.

Convergence ILU preconditioning

Groundwaterflow: $\lambda(\mathbf{A}) \in [\lambda_1, \lambda_n] \subset (0, \infty)$, $\mathcal{C} \equiv \frac{\lambda_n}{\lambda_1}$

$1/\mathcal{C} \sim \max(h_x^2, h_y^2, \ldots)$.

Convergence.

$$\rho_k \equiv \frac{\|\mathbf{r}_k\|}{\|\mathbf{r}_0\|} \leq \exp(-2k/\mu)$$

Without preconditioning

LMR: $\mu = \mathcal{C}$, GCR: $\mu = \sqrt{\mathcal{C}}$

With D-MILU preconditioning

LMR: $\mu = \sqrt{\mathcal{C}}$, GCR: $\mu = \mathcal{C}^{\frac{1}{4}}$.

Example. $\mathcal{C} = 2\,10^4$. GCR:

without precond. $\rho_k \leq 10^{-3}$ for $k = 490$,

with D-MILU $\rho_k \leq 10^{-3}$ for $k = 42$.

Convergence ILU preconditioning

Groundwaterflow: $\lambda(\mathbf{A}) \in [\lambda_1, \lambda_n] \subset (0, \infty)$, $\mathcal{C} \equiv \frac{\lambda_n}{\lambda_1}$

$1/\mathcal{C} \sim \max(h_x^2, h_y^2, \ldots)$.

Convergence.

$$\rho_k \equiv \frac{\|\mathbf{r}_k\|}{\|\mathbf{r}_0\|} \leq \exp(-2k/\mu)$$

Without preconditioning

LMR: $\mu = \mathcal{C}$, GCR: $\mu = \sqrt{\mathcal{C}}$

With D-MILU preconditioning

LMR: $\mu = \sqrt{\mathcal{C}}$, GCR: $\mu = \mathcal{C}^{\frac{1}{4}}$.

In case $\mathbf{Ax} = \mathbf{b}$ from an advection diffusion PDE:
ILU or ILU(ω) with ω small can be very effective
if the advection term is large
(and the stepsizes are not very small).

Program

- Flexible GCR
- Preconditioning
- D-ILU
- Incomplete LU-decomposition
- Why preconditioning?
- Costs
- How to include a preconditioner
- Savings

Choose $tol > 0$, $\mathbf{x}$, $k_{\max}$,
Compute $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$
For $k = 0 : k_{\max}$

Stop if $\|\mathbf{r}\|_2 \leq tol \|\mathbf{b}\|_2$
Solve $\mathbf{Mu}_k = \mathbf{r}$ for $\mathbf{u}_k$
$\mathbf{c}_k = \mathbf{Au}_k$
For $j = 0 : k - 1$
$\beta \leftarrow \mathbf{c}_j^* \mathbf{c}_k / \sigma_j$
$\mathbf{u}_k \leftarrow \mathbf{u}_k - \beta \, \mathbf{u}_j$
$\mathbf{c}_k \leftarrow \mathbf{c}_k - \beta \, \mathbf{c}_j$
end for
$\sigma_k = \mathbf{c}_k^* \mathbf{c}_k$, $\alpha \leftarrow \mathbf{c}_k^* \mathbf{r} / \sigma_k$
$\mathbf{x} \leftarrow \mathbf{x} + \alpha \, \mathbf{u}_k$
$\mathbf{r} \leftarrow \mathbf{r} - \alpha \, \mathbf{c}_k$

end for

Choose $tol > 0$, $\mathbf{x}$, $k_{\max}$,
Compute $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$
For $k = 0 : k_{\max}$

Stop if $\|\mathbf{r}\|_2 \leq tol \|\mathbf{b}\|_2$
$\tilde{\mathbf{u}}_k = \mathbf{r}$
$\mathbf{c}_k = \mathbf{AM}^{-1} \tilde{\mathbf{u}}_k$
For $j = 0 : k - 1$
$\beta \leftarrow \mathbf{c}_j^* \mathbf{c}_k / \sigma_j$
$\tilde{\mathbf{u}}_k \leftarrow \tilde{\mathbf{u}}_k - \beta \, \tilde{\mathbf{u}}_j$
$\mathbf{c}_k \leftarrow \mathbf{c}_k - \beta \, \mathbf{c}_j$
end for
$\sigma_k = \mathbf{c}_k^* \mathbf{c}_k$, $\alpha \leftarrow \mathbf{c}_k^* \mathbf{r} / \sigma_k$
$\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} + \alpha \, \tilde{\mathbf{u}}_k$
$\mathbf{r} \leftarrow \mathbf{r} - \alpha \, \mathbf{c}_k$

end for

Including a preconditioner

There are several way to include preconditioner in GCR.

Including a preconditioner

- Modify the GCR algorithm (**implicit** preconditioning): replace "$\mathbf{u}_k = \mathbf{r}_k$" by "`Solve` $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ `for` $\mathbf{u}_k$".
- Modify the problem to $\mathbf{A}\mathbf{M}^{-1}\widetilde{\mathbf{x}} = \mathbf{b}$ (**explicit right** preconditioning): replace $\mathbf{A}$ by $\mathbf{A}\mathbf{M}^{-1}$; $\mathbf{x} = \mathbf{M}^{-1}\widetilde{\mathbf{x}}$.

Theorem. The residuals $\mathbf{r}_k$ and the $\mathbf{c}_j$ in both versions are the same and $\mathbf{x}_k = \mathbf{M}^{-1}\widetilde{\mathbf{x}}_k$.

Including a preconditioner

- Modify the GCR algorithm (**implicit** preconditioning): replace "$\mathbf{u}_k = \mathbf{r}_k$" by "`Solve` $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ `for` $\mathbf{u}_k$".
- Modify the problem to $\mathbf{A}\mathbf{M}^{-1}\widetilde{\mathbf{x}} = \mathbf{b}$ (**explicit right** preconditioning): replace $\mathbf{A}$ by $\mathbf{A}\mathbf{M}^{-1}$; $\mathbf{x} = \mathbf{M}^{-1}\widetilde{\mathbf{x}}$.

Explicit preconditioning requires **pre processing** (that is, before GCR can be applied, a routine has to be formed that computes $\mathbf{c}_k = \mathbf{A}\mathbf{M}^{-1}\widetilde{\mathbf{u}}_k$) and **post processing** (after the applying GCR, $\mathbf{x}_k$ has to be computed from $\widetilde{\mathbf{x}}$).

Including a preconditioner

• Modify the GCR algorithm (**implicit** preconditioning): replace "$\mathbf{u}_k = \mathbf{r}_k$" by "`Solve` $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ `for` $\mathbf{u}_k$".

• Modify the problem to $\mathbf{A}\mathbf{M}^{-1}\widetilde{\mathbf{x}} = \mathbf{b}$ (**explicit right** preconditioning): replace $\mathbf{A}$ by $\mathbf{A}\mathbf{M}^{-1}$; $\mathbf{x} = \mathbf{M}^{-1}\widetilde{\mathbf{x}}$.

Observation. Do not explicitly form the matrix $\mathbf{A}\mathbf{M}^{-1}$, but design an efficient routine to compute $\mathbf{c}_k = \mathbf{A}\mathbf{M}^{-1}\mathbf{u}_k$.

$$\mathbf{c} = \mathtt{MV}(\mathbf{A}, \mathbf{M}, \mathbf{r})$$

Including a preconditioner

- Modify the GCR algorithm (**implicit** preconditioning): replace "$\mathbf{u}_k = \mathbf{r}_k$" by "`Solve` $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ `for` $\mathbf{u}_k$".
- Modify the problem to $\mathbf{A}\mathbf{M}^{-1}\tilde{\mathbf{x}} = \mathbf{b}$ (**explicit right** preconditioning): replace $\mathbf{A}$ by $\mathbf{A}\mathbf{M}^{-1}$; $\mathbf{x} = \mathbf{M}^{-1}\tilde{\mathbf{x}}$.
- Modify the problem to $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \tilde{\mathbf{b}} \equiv \mathbf{M}^{-1}\mathbf{b}$ (**explicit left** preconditioning): replace $\mathbf{A}$ by $\mathbf{M}^{-1}\mathbf{A}$ and $\mathbf{b}$ by $\tilde{\mathbf{b}}$.

Explicit left preconditioning requires **pre processing** (to form a routine that computes $\mathbf{c}_k = \mathbf{M}^{-1}\mathbf{A}\mathbf{r}_k$) and to solve $\tilde{\mathbf{b}}$ from $\mathbf{M}\tilde{\mathbf{b}} = \mathbf{b}$. But no post processing

Including a preconditioner

- Modify the GCR algorithm (**implicit** preconditioning): replace "$\mathbf{u}_k = \mathbf{r}_k$" by "`Solve` $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ `for` $\mathbf{u}_k$".
- Modify the problem to $\mathbf{A}\mathbf{M}^{-1}\widetilde{\mathbf{x}} = \mathbf{b}$ (**explicit right** preconditioning): replace $\mathbf{A}$ by $\mathbf{A}\mathbf{M}^{-1}$; $\mathbf{x} = \mathbf{M}^{-1}\widetilde{\mathbf{x}}$.
- Modify the problem to $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \widetilde{\mathbf{b}} \equiv \mathbf{M}^{-1}\mathbf{b}$ (**explicit left** preconditioning): replace $\mathbf{A}$ by $\mathbf{M}^{-1}\mathbf{A}$ and $\mathbf{b}$ by $\widetilde{\mathbf{b}}$.

Assignment. Write a routine that perform explicit left preconditioning.

Compare the performance of this routine with the one of GCR with implicit preconditioning (use the same preconditioner and the same $\mathbf{x}_0$). Make sure that you obtain residuals of comparable quality.

Including a preconditioner

- Modify the GCR algorithm (**implicit** preconditioning): replace "$\mathbf{u}_k = \mathbf{r}_k$" by "`Solve` $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ `for` $\mathbf{u}_k$".
- Modify the problem to $\mathbf{A}\mathbf{M}^{-1}\tilde{\mathbf{x}} = \mathbf{b}$ (**explicit right** preconditioning): replace $\mathbf{A}$ by $\mathbf{A}\mathbf{M}^{-1}$; $\mathbf{x} = \mathbf{M}^{-1}\tilde{\mathbf{x}}$.
- Modify the problem to $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \tilde{\mathbf{b}} \equiv \mathbf{M}^{-1}\mathbf{b}$ (**explicit left** preconditioning): replace $\mathbf{A}$ by $\mathbf{M}^{-1}\mathbf{A}$ and $\mathbf{b}$ by $\tilde{\mathbf{b}}$.

Observations.
– The $\mathbf{x}_k$ and $\mathbf{r}_k$ in GCR with implicit preconditioning are approximations and residuals of the original problem.
– GCR with right preconditioning computes residuals of the original problem, but the approximates are preconditioned ($\mathbf{M}\mathbf{x}_k = \tilde{\mathbf{x}}_k$).
– GCR with left preconditioning computes approximate solutions of the original problem, but the residuals are preconditioned.

Program

- Flexible GCR
- Preconditioning
- D-ILU
- Incomplete LU-decomposition
- Why preconditioning?
- Costs
- How to include a preconditioner
- Savings

GCR

```
Choose tol > 0, x, k_max,
Compute r = b − Ax
For k = 0 : k_max
    Stop if ||r||_2 ≤ tol||b||_2
    u_k = r
    c_k = A u_k
    For j = 0 : k − 1
        β ← c_j^* c_k / σ_j
        u_k = u_k − β u_j
        c_k = c_k − β c_j
    end for
    σ_k = c_k^* c_k,  α ← c_k^* r / σ_k
    x ← x + α u_k
    r ← r − α c_k
end for
```

More efficients steps

The most expensive part of GCR is the orthogonalization loop: the costs of this part at step k for step k are

$6kn$,

while the costs (for 2-d) for the other part are

$19n$ flop (without precond.) or

$30n$ flop (with D-ILU).

For $k > 10$, the orthogonalization dominates the costs.

More efficients steps

The most expensive part of GCR is the orthogonalization loop: the costs of this part at step k for step k are

$6kn$,

while the costs (for 2-d) for the other part are

$19n$ flop (without precond.) or

$30n$ flop (with D-ILU).

Idea. Restart every ℓ steps with the most recent approximate solution as initial guess for the next ℓ steps: **restarted** GCR.

Notation. $\lfloor \mu \rfloor = k$ if $\mu \in [k, k+1)$ and $k \in \mathbb{N}_0$.

Restarted GCR

```
Choose tol > 0, x, k_max, ℓ ∈ ℕ
Compute r = b − Ax
For k = 0 : k_max
    Stop if ||r||_2 ≤ tol||b||_2
    u_k = r
    c_k = A u_k
    For j = ℓ⌊k/ℓ⌋ : k − 1
        β ← c_j^* c_k / σ_j
        u_k = u_k − β u_j
        c_k = c_k − β c_j
    end for
    σ_k = c_k^* c_k,  α ← c_k^* r / σ_k
    x ← x + α u_k
    r ← r − α c_k
end for
```

$$\begin{array}{l}
\texttt{Choose}\ \ \mathit{tol} > 0,\ \mathbf{x},\ k_{\mathrm{max}},\ \ell \in \mathbb{N} \\
\texttt{Compute}\ \ \mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x} \\
\texttt{For}\ \ k = 0 : k_{\mathrm{max}} \\
\quad \texttt{Stop if}\ \ \|\mathbf{r}\|_2 \leq \mathit{tol}\|\mathbf{b}\|_2 \\
\quad \mathbf{u}_k = \mathbf{r} \\
\quad \mathbf{c}_k = \mathbf{A}\mathbf{u}_k \\
\quad \texttt{For}\ \ j = \ell\lfloor \frac{k}{\ell} \rfloor : k-1 \\
\quad\quad \beta \leftarrow \mathbf{c}_j^* \mathbf{c}_k / \sigma_j \\
\quad\quad \mathbf{u}_k = \mathbf{u}_k - \beta\, \mathbf{u}_j \\
\quad\quad \mathbf{c}_k = \mathbf{c}_k - \beta\, \mathbf{c}_j \\
\quad \texttt{end for} \\
\quad \sigma_k = \mathbf{c}_k^* \mathbf{c}_k,\quad \alpha \leftarrow \mathbf{c}_k^* \mathbf{r} / \sigma_k \\
\quad \mathbf{x} \leftarrow \mathbf{x} + \alpha\, \mathbf{u}_k \\
\quad \mathbf{r} \leftarrow \mathbf{r} - \alpha\, \mathbf{c}_k \\
\texttt{end for}
\end{array}$$

More efficients steps

The most expensive part of GCR is the orthogonalization loop: the costs of this part at step k for step k are

$6kn$,

while the costs (for 2-d) for the other part are

$19n$ flop (without precond.) or

$30n$ flop (with D-ILU).

Idea. Keep only the last ℓ vectors $\mathbf{c}_{k-\ell}, \ldots, \mathbf{c}_{k-1}$ (and the associated $\mathbf{u}_j$) in the orthogonalization process: **truncate** GCR.

Truncated GCR

```
Choose tol > 0, x, k_max, ℓ ∈ ℕ
Compute r = b − Ax
For k = 0 : k_max
    Stop if ||r||_2 ≤ tol||b||_2
    u_k = r
    c_k = Au_k
    For j = max(k − ℓ, 0) : k − 1
        β ← c_j^* c_k / σ_j
        u_k = u_k − β u_j
        c_k = c_k − β c_j
    end for
    σ_k = c_k^* c_k,  α ← c_k^* r / σ_k
    x ← x + α u_k
    r ← r − α c_k
end for
```

Choose $tol > 0$, $\mathbf{x}$, $k_{\max}$, $\ell \in \mathbb{N}$

Compute $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$

For $k = 0 : k_{\max}$

- Stop if $\|\mathbf{r}\|_2 \le tol\|\mathbf{b}\|_2$
- $\mathbf{u}_k = \mathbf{r}$
- $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$
- For $j = \max(k - \ell, 0) : k - 1$
  - $\beta \leftarrow \mathbf{c}_j^* \mathbf{c}_k / \sigma_j$
  - $\mathbf{u}_k = \mathbf{u}_k - \beta\, \mathbf{u}_j$
  - $\mathbf{c}_k = \mathbf{c}_k - \beta\, \mathbf{c}_j$
- end for
- $\sigma_k = \mathbf{c}_k^* \mathbf{c}_k$, $\alpha \leftarrow \mathbf{c}_k^* \mathbf{r} / \sigma_k$
- $\mathbf{x} \leftarrow \mathbf{x} + \alpha\, \mathbf{u}_k$
- $\mathbf{r} \leftarrow \mathbf{r} - \alpha\, \mathbf{c}_k$

end for

Assignment. Write a function subroutine GCR

$$\mathbf{x} = \mathtt{GCR}(\mathbf{A}, \mathbf{b}, \mathbf{x}_0, \mathit{tol}, k_{\max}, \ell)$$

that performs

- restart if $\ell > 0$,
- standard GCR if $\ell = 0$,
- truncates if $\ell < 0$ with truncation length $|\ell|$.

More efficients steps

The most expensive part of GCR is the orthogonalization loop: the costs of this part at step k for step k are

$6kn$,

while the costs (for 2-d) for the other part are

$19n$ flop (without precond.) or

$30n$ flop (with D-ILU).

Idea. Keep only the last ℓ vectors $\mathbf{c}_{k-\ell}, \ldots, \mathbf{c}_{k-1}$ (and the associated $\mathbf{u}_j$) in the orthogonalization process: **truncate** GCR.

Example. $\ell = 1$: **Conjugate Residuals**

Conjugate Residuals

```
Choose tol > 0, x, k_max
Compute r = b - Ax
For k = 0 : k_max
    Stop if ||r||_2 <= tol ||b||_2
    u_k = r
    c_k = A u_k
    if k > 0
        β ← c*_{k-1} c_k / σ_{k-1}
        u_k ← u_k - β u_{k-1}
        c_k ← c_k - β c_{k-1}
    end if
    σ_k = c*_k c_k,  α ← c*_k r / σ_k
    x ← x + α u_k
    r ← r - α c_k
end for
```

Choose $tol > 0$, $\mathbf{x}$, $k_{\max}$

Compute $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$

For $k = 0 : k_{\max}$

- Stop if $\|\mathbf{r}\|_2 \le tol\|\mathbf{b}\|_2$
- $\mathbf{u}_k = \mathbf{r}$
- $\mathbf{c}_k = \mathbf{A}\mathbf{u}_k$
- if $k > 0$
  - $\beta \leftarrow \mathbf{c}_{k-1}^* \mathbf{c}_k / \sigma_{k-1}$
  - $\mathbf{u}_k \leftarrow \mathbf{u}_k - \beta\, \mathbf{u}_{k-1}$
  - $\mathbf{c}_k \leftarrow \mathbf{c}_k - \beta\, \mathbf{c}_{k-1}$
- end if
- $\sigma_k = \mathbf{c}_k^* \mathbf{c}_k$, $\alpha \leftarrow \mathbf{c}_k^* \mathbf{r} / \sigma_k$
- $\mathbf{x} \leftarrow \mathbf{x} + \alpha\, \mathbf{u}_k$
- $\mathbf{r} \leftarrow \mathbf{r} - \alpha\, \mathbf{c}_k$

end for

Conjugate Residuals

Choose $\mathit{tol} > 0$, $\mathbf{x}$, $k_{\max}$

$\mathbf{u}_1 = \mathbf{c}_1 = \mathbf{0}$, $\sigma = 1$

Compute $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$

For $k = 0 : k_{\max}$

    Stop if $\|\mathbf{r}\|_2 \le \mathit{tol}\|\mathbf{b}\|_2$

    $\mathbf{u}_0 \leftarrow \mathbf{u}_1$, $\mathbf{u}_1 \leftarrow \mathbf{r}$

    $\mathbf{c}_0 \leftarrow \mathbf{c}_1$, $\mathbf{c}_1 \leftarrow \mathbf{A}\mathbf{u}_k$

    $\beta \leftarrow \mathbf{c}_0^* \mathbf{c}_1 / \sigma$

    $\mathbf{u}_1 \leftarrow \mathbf{u}_1 - \beta\, \mathbf{u}_0$

    $\mathbf{c}_1 \leftarrow \mathbf{c}_1 - \beta\, \mathbf{c}_0$

    $\sigma \leftarrow \mathbf{c}_1^* \mathbf{c}_1$, $\alpha \leftarrow \mathbf{c}_1^* \mathbf{r} / \sigma$

    $\mathbf{x} \leftarrow \mathbf{x} + \alpha\, \mathbf{u}_1$

    $\mathbf{r} \leftarrow \mathbf{r} - \alpha\, \mathbf{c}_1$

end for

Theorem. If $\mathbf{A}^* = \mathbf{A}$ then GCR = CR.

that is, in exact arithmetic CR and GCR have the same residual $\mathbf{r}_k$ and the same approximate solution $\mathbf{x}_k$ (when started withe the same initial guess $\mathbf{x}_0$).

Assignment. Program CR. Check that CG = GCR in case the matrix is symmetric (and real).

Include also preconditioning in CR. What is the reduction in number of steps that is required by including preconditioning in order to have a more efficient CR algorithm?