Technische Universität Braunschweig

Institute for Computational Mathematics

ILUPACK toolbox for MATLAB

http://ilupack.tu-bs.de

Matthias Bollhöfer, October 1, 2012

Outline

- **Introduction — using the ILUPACK toolbox**
  - Preconditioning Systems
  - Getting started
- **What's behind the toolbox**
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting
- **Tools**
  - Vizualization
  - Further tools
  - Automatic structure detection
- **Closing Remarks**

Outline

- **Introduction — using the ILUPACK toolbox**
  - Preconditioning Systems
  - Getting started
- **What's behind the toolbox**
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting
- **Tools**
  - Vizualization
  - Further tools
  - Automatic structure detection
- **Closing Remarks**

Preconditioning Systems

Objective

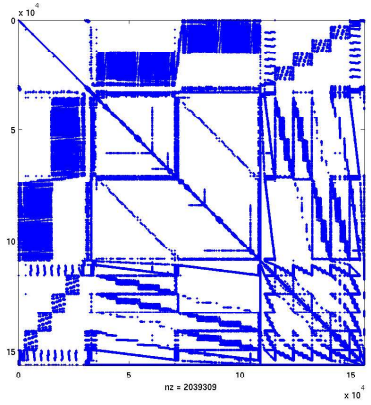
Given a large sparse nonsingular matrix A and a linear system

$$Ax = b,$$

1. construct approximate factorization $A \approx \tilde{A} = LU$
2. solve $Ax = b$ using a preconditioned Krylov subspace iteration method

How large, how sparse, and why using an approximate factorization?

- system size $n = 10^5 \rightarrow 10^9$, number of nonzeros typically less $100n$
- memory requirements, computation time



Outline

- **Introduction — using the ILUPACK toolbox**
  - Preconditioning Systems
  - Getting started
- **What's behind the toolbox**
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting
- **Tools**
  - Vizualization
  - Further tools
  - Automatic structure detection
- **Closing Remarks**

Getting started

After adding *ILUPACK* system path (e.g. ≫ `addpath 'ilupack'`) a large sparse system Ax = b could be solved as follows:

Approximate Factorization

```
≫ [PREC,options]=AMGfactor(A);
```

⇒ preconditioner is built using the default options.

Iterative Solution

```
≫ [x, options] = AMGsolver(A, PREC, options, b);
```

⇒ system is solved.

Release Memory

```
≫ PREC = AMGdelete(PREC);
```

Getting started

After adding *ILUPACK* system path (e.g. ≫ `addpath 'ilupack'`) a large sparse system Ax = b could be solved as follows:

Approximate Factorization

```
≫ [PREC,options]=AMGfactor(A);
```

⇒ preconditioner is built using the default options.

Iterative Solution

```
≫ [x, options] = AMGsolver(A, PREC, options, b);
```

⇒ system is solved.

Release Memory

```
≫ PREC = AMGdelete(PREC);
```

Getting started

After adding *ILUPACK* system path (e.g. ≫ `addpath 'ilupack'`) a large sparse system Ax = b could be solved as follows:

Approximate Factorization

```
≫ [PREC,options]=AMGfactor(A);
```

⇒ preconditioner is built using the default options.

Iterative Solution

```
≫ [x, options] = AMGsolver(A, PREC, options, b);
```

⇒ system is solved.

Release Memory

```
≫ PREC = AMGdelete(PREC);
```

Getting started

After adding *ILUPACK* system path (e.g. $\gg$ `addpath 'ilupack'`) a large sparse system Ax = b could be solved as follows:

Approximate Factorization

```
>> [PREC,options]=AMGfactor(A);
```

$\Rightarrow$ preconditioner is built using the default options.

Iterative Solution

```
>> [x, options] = AMGsolver(A, PREC, options, b);
```

$\Rightarrow$ system is solved.

Release Memory

```
>> PREC = AMGdelete(PREC);
```

Getting started

After adding *ILUPACK* system path (e.g. ≫ `addpath 'ilupack'`) a large sparse system Ax = b could be solved as follows:

Approximate Factorization

```
≫ [PREC,options]=AMGfactor(A);
```

⇒ preconditioner is built using the default options.

Iterative Solution

```
≫ [x, options] = AMGsolver(A, PREC, options, b);
```

⇒ system is solved.

Release Memory

```
≫ PREC = AMGdelete(PREC);
```

Parameter setting

ILUPACK offers many, many parameters

Some Default Parameters

```
≫ options=AMGinit(A);
options =
     matching:1
     ordering:'amd'
      droptol:1.0000e-02
     droptolS:1.0000e-03
     droptolc:2.2204e-12
      condest:5
       restol:1.4901e-08
        maxit:500
        elbow:10
         lfil:156116
        lfilS:156116
       typetv:'none'
           tv:[156115x1 double]
          amg:'ilu'
npresmoothing:1
npostsmoothing:1
       ncoarse:1
    presmoother:'gsf'
   postsmoother:'gsb'
         FCpart:'none'
     typecoarse:'ilu'
         solver:'gmres'
        damping:6.6667e-01
    contraction:5.0000e-01
       nrestart:30
            ind:[156115x1 double]
mixedprecision:0
  coarsereduce:1
decoupleconstraints:0
```

Parameter setting — don't try to read all that!

Some parameters you will find familiar, others may confuse you . . .

`matching`	improve diagonal dominance using maximum weighted matchings
`ordering`	preprocess the system by a symbolic reordering (e.g. 'Approximate Minimum Degree')
`droptol`	threshold to drop small entries during the factorization
`droptolS`	threshold to drop small entries in the intermediate Schur complements
`condest`	bound for the inverse triangular factors
`restol`	stopping criterion for the iterative process (backwar error)
`maxit`	maximum number of iteration steps
`lfil`	number of nonzeros per row in the approximate factorization
`lfilS`	number of nonzeros per row in the approximate Schur complement
`solver`	Krylov subspace method ('cg', 'sqmr', 'gmres',. . .)
`nrestart`	number of restarts for GMRES (if used)

Parameter setting — continued . . .

`droptolc`	treshold to drop small entries in the constraint part (saddle point problems)
`elbow`	recommended memory space to keep the preconditioner (relative to the number of nonzeros of A)
complement	
`typetv`	type of test vector (’static’ or ’dynamic’)
`tv`	keep the ILU exact for this test vector
`amg`	type of multilevel algorithm (’ilu’, ’amli’, ’amg’)
`npresmoothing`	number of pre-smoothing steps
`npostsmoothing`	number of post-smoothing steps
`ncoarse`	number of coarse grid correction steps
`presmoother`	type of pre-smoother (Jacobi, Gauss-Seidel, ’j’, ’gsf’, ’gsb’)
`postsmoother`	type of post-smoother (Jacobi, Gauss-Seidel, ’j’, ’gsf’, ’gsb’)

Parameter setting — continued . . .

`FCpart`	a priori separation of unknowns into fine and coarse grid nodes ('yes', 'no')
`typecoarse`	type of coarse grid system (approximate Schur complement, 'ilu', 'amg'))
`damping`	damping factor for Jacobi smoothing
`contraction`	contraction factor for the local residual on the coarse grid system (when flexible solvers are used)
`ind`	index indicator to identify a saddle poinbt structure
`mixedprecision`	single precision preconditioner, double precision solver
`coarsereduce`	discard (1,2) block and (2,1) in the multilevel ILU
`decoupleconstraints`	decouple constraint in a saddle point problem

How to set up your own parameters

- Your iterative solver does not converge
  $\Rightarrow$ reduce `options.droptol`, `options.droptolS`
- You want to provide your own initial guess
  $\Rightarrow$ call `[x, options] = AMGsolver(A, PREC, options, b, x0);`
- You would accept more iteration steps
  $\Rightarrow$ increase `options.maxit`
- You would like to have a more accurate solution
  $\Rightarrow$ decrease `options.restol`

How to set up your own parameters

- Your iterative solver does not converge
  $\Rightarrow$ reduce `options.droptol`, `options.droptolS`

- You want to provide your own initial guess
  $\Rightarrow$ call `[x, options] = AMGsolver(A, PREC, options, b, x0);`

- You would accept more iteration steps
  $\Rightarrow$ increase `options.maxit`

- You would like to have a more accurate solution
  $\Rightarrow$ decrease `options.restol`

How to set up your own parameters

- Your iterative solver does not converge
  $\Rightarrow$ reduce `options.droptol`, `options.droptolS`
- You want to provide your own initial guess
  $\Rightarrow$ call `[x, options] = AMGsolver(A, PREC, options, b, x0);`
- You would accept more iteration steps
  $\Rightarrow$ increase `options.maxit`
- You would like to have a more accurate solution
  $\Rightarrow$ decrease `options.restol`

How to set up your own parameters

- Your iterative solver does not converge
  $\Rightarrow$ reduce `options.droptol`, `options.droptolS`
- You want to provide your own initial guess
  $\Rightarrow$ call `[x, options] = AMGsolver(A, PREC, options, b, x0);`
- You would accept more iteration steps
  $\Rightarrow$ increase `options.maxit`
- You would like to have a more accurate solution
  $\Rightarrow$ decrease `options.restol`

Outline

- Introduction — using the ILUPACK toolbox
  - Preconditioning Systems
  - Getting started
- **What's behind the toolbox**
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting
- Tools
  - Vizualization
  - Further tools
  - Automatic structure detection
- Closing Remarks

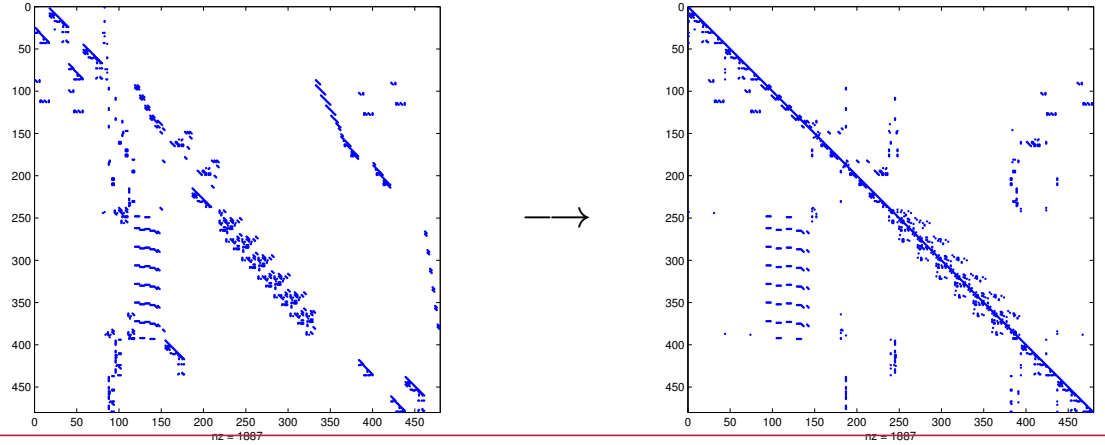
Matchings

- Maximum weighted matchings are a combinatorial graph theoretical approach to improve diagonal dominance

- Using matchings a general system A is
  1. rescaled
  2. permuted

  such that $A \to \Pi^T D_r A D_c$ satisfies $|a_{ij}^{new}| \leqslant 1$, $|a_{ii}^{new}| = 1$.

- Matchings can be symmetrized for systems satisfying $|A| = |A^T|$ with similar properties

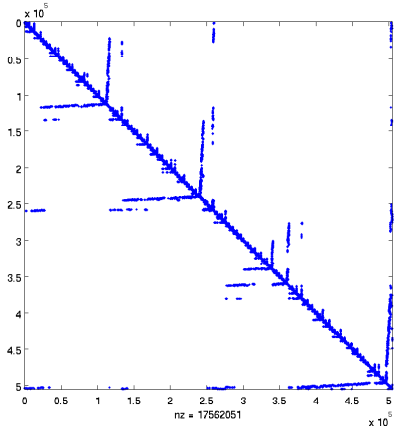


Outline

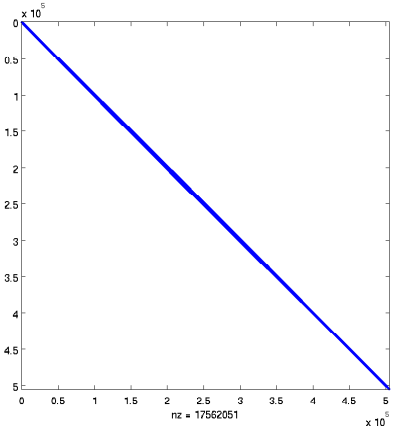
- Introduction — using the ILUPACK toolbox
  - Preconditioning Systems
  - Getting started
- **What's behind the toolbox**
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting
- Tools
  - Vizualization
  - Further tools
  - Automatic structure detection
- Closing Remarks

Symmetric reorderings

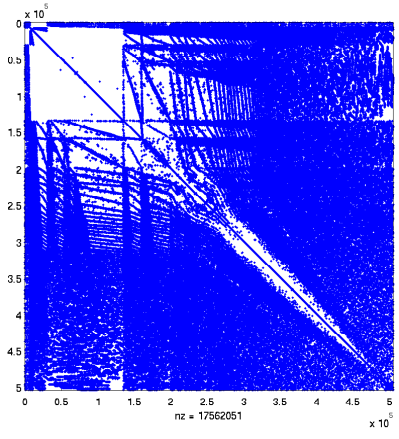
'amd' Approximate Minimum Degree



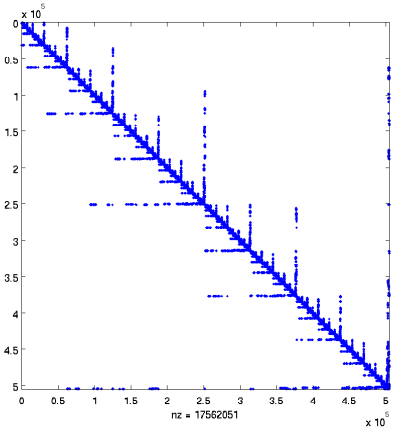
'rcm' Reverse Cuthill-McKee



'mmd' Multiple Minimum Degree



'metisn', 'metise' Metis (by nodes/edges)

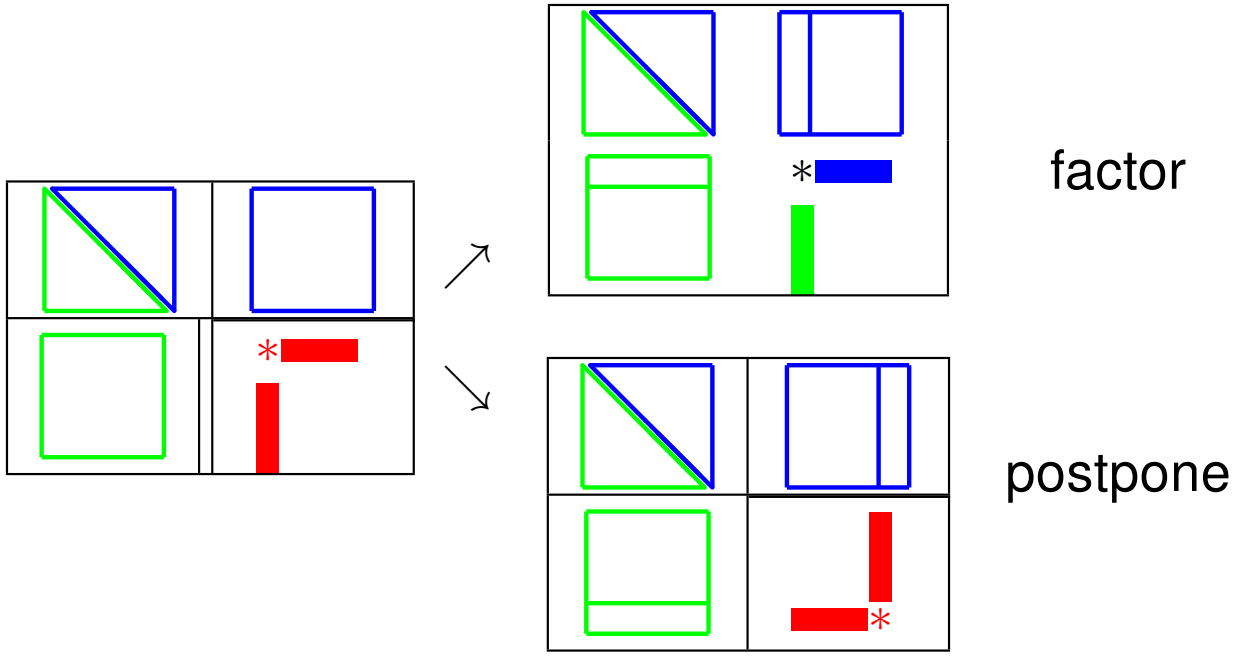


Outline

- Introduction — using the ILUPACK toolbox
  - Preconditioning Systems
  - Getting started
- **What's behind the toolbox**
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting
- Tools
  - Vizualization
  - Further tools
  - Automatic structure detection
- Closing Remarks

Inverse-based pivoting

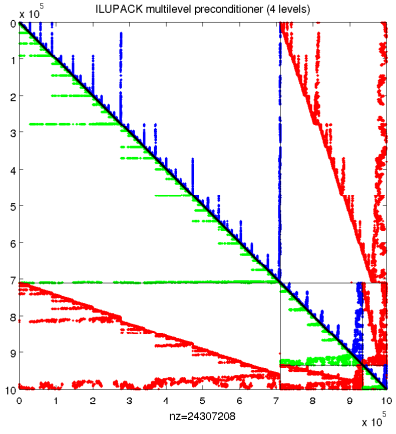
- Factorize $A \approx LDU$ such that $\|L^{-1}\| \leqslant \kappa$, $\|U^{-1}\| \leqslant \kappa$
- estimate $\|L^{-1}\|$, $\|U^{-1}\|$ efficiently [Cline,Moler,Stewart,Wilkinson'77]
- Enforce $\|L^{-1}\| \leqslant \kappa$, $\|U^{-1}\| \leqslant \kappa$ by inverse-based pivoting



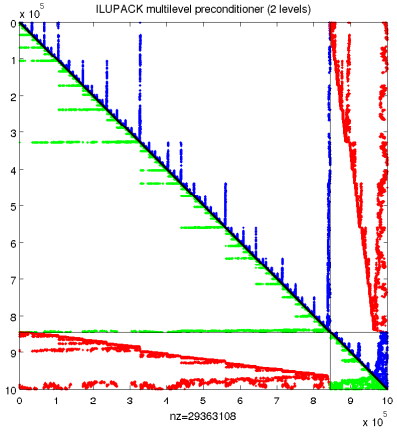
- postponed updates become the coarse grid system
- Algebraic Multilevel Method

Inverse-based pivoting

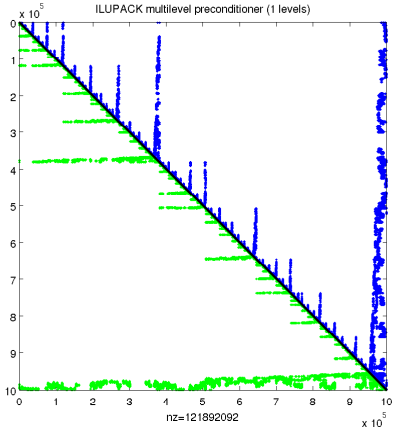
$\kappa = 3 \longrightarrow$ 4 levels



$\kappa = 10 \longrightarrow$ 2 levels



$\kappa = 100 \longrightarrow$ 1 level



Outline

- Introduction — using the ILUPACK toolbox
  - Preconditioning Systems
  - Getting started
- What's behind the toolbox
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting
- **Tools**
  - Vizualization
  - Further tools
  - Automatic structure detection
- Closing Remarks

Vizualization

Display Multilevel ILU

Factorization

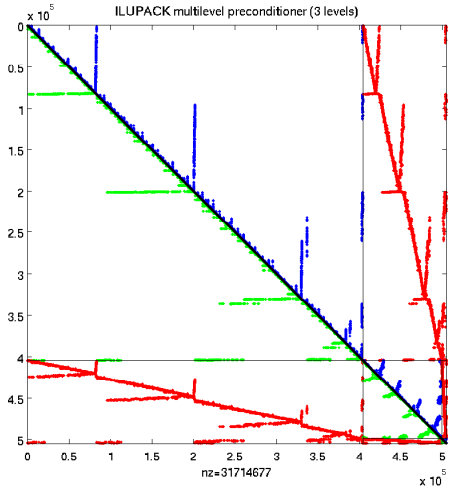
```
≫ [PREC,options]=AMGfactor(A,options);
```

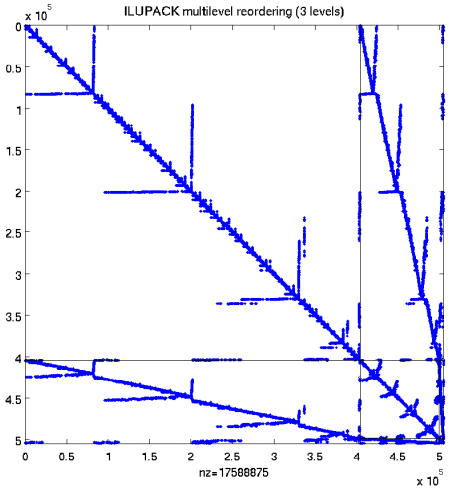
Display ILU

```
≫ AMGspy(PREC);
```

Display reordered & rescaled original system

```
≫ AMGspy(A, PREC);
```





Outline

- Introduction — using the ILUPACK toolbox
  - Preconditioning Systems
  - Getting started
- What's behind the toolbox
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting
- **Tools**
  - Vizualization
  - Further tools
  - Automatic structure detection
- Closing Remarks

Further tools

Number of nonzeros

```
>> nz=AMGnnz(PREC);
```

Solve a single preconditioned system

```
>> x = AMGsol(PREC,b);
```

Load/save matrix in Harwell-Boeing format

```
>> [A,rhs,rhstyp]=loadhbo(filename);
>> savehbo(filename, A);
```

(also available for right side, initial guess, . . .)

simplified QMR

```
>> [x,flag,iter,resvec]=sqmr(A,b,tol,maxit,M1,M2,x0);
```

Further tools

Metis reorderings

```
≫ [pl,pr,Dl,Dr] = metisn(A);   [pl,pr,Dl,Dr] = metise(A);
```

Reorderings including Maximum Weight Matching

```
≫ [pl,pr,Dl,Dr] = mwmmetisn(A); [pl,pr,Dl,Dr] = mwmmetisn(A);
≫ [pl,pr,Dl,Dr] = mwmrcm(A);    [pl,pr,Dl,Dr] = mwmamd(A);
≫ [pl,pr,Dl,Dr] = mwmmmd(A);
```

Reorderings including Symmetric Maximum Weight Matching

```
≫ [p,D] = symmwmmetisn(A);   [p,D] = symmwmmetisn(A);
≫ [p,D] = symmwmrcm(A);      [p,D] = symmwmamd(A);
≫ [p,D] = symmwmmmd(A);
```

Outline

- Introduction — using the ILUPACK toolbox
  - Preconditioning Systems
  - Getting started
- What's behind the toolbox
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting
- **Tools**
  - Vizualization
  - Further tools
  - Automatic structure detection
- Closing Remarks

Automatic structure detection

ILUPACK offers many special purpose drivers for

- complex systems
  - general sparse (with GMRES)
  - complex symmetric (with SQMR)
  - complex Hermitian (with SQMR)
  - complex Hermitian positive definite (with CG)
- real systems
  - general sparse (with GMRES)
  - real symmetric (with SQMR)
  - real symmetric positive definite (with CG)

ILUPACK toolbox for MATLAB automatically detects

- real/complex systems
- symmetry structures
- mixed real/complex systems (e.g. real preconditioner/complex matrix)

Automatic structure detection

ILUPACK asks YOU to specify if the system is positive definite

SPD Case

```
≫ options.isdefinite=1;
≫ options=AMGinit(A,options);
```

default options for the symmetric (Hermitian) positive definite case are set

ILUPACK offers a tool to convert a symmetric preconditioner into a positive definite one

SPD Case

```
≫ PREC = AMGconvert(PREC);
```

symmetric (Hermitian) indefinite factorization is changed to a definite factorization

Closing Remarks

Watch the *ILUPACK* website at

http://ilupack.tu-bs.de

Current release is *ILUPACK* V2.4 (including MATLAB toolbox)