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**
  - Visualization
  - 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 \textit{ILUPACK} system path (e.g. \texttt{addpath 'ilupack'}) a large sparse system $Ax = b$ could be solved as follows:

\begin{itemize}
  \item \textbf{Approximate Factorization}
    \begin{verbatim}
    >> [PREC,options] = AMGfactor(A);
    \end{verbatim}
    \Rightarrow \text{preconditioner is built using the default options.}
  \item \textbf{Iterative Solution}
    \begin{verbatim}
    >> [x, options] = AMGsolver(A, PREC, options, b);
    \end{verbatim}
    \Rightarrow \text{system is solved.}
  \item \textbf{Release Memory}
    \begin{verbatim}
    >> PREC = AMGdelete(PREC);
    \end{verbatim}
\end{itemize}
Getting started

After adding \textit{ILUPACK} system path (e.g. >> addpath 'ilupack') a large sparse system $Ax = b$ could be solved as follows:

\textbf{Approximate Factorization}

\begin{verbatim}
>> [PREC,options]=AMGfactor(A);
\end{verbatim}

⇒ preconditioner is built using the default options.

\textbf{Iterative Solution}

\begin{verbatim}
>> [x, options] = AMGsolver(A, PREC, options, b);
\end{verbatim}

⇒ system is solved.

\textbf{Release Memory}

\begin{verbatim}
>> PREC = AMGdelete(PREC);
\end{verbatim}
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**

\[
\text{≫ [PREC,options]=AMGfactor(A);
}
\]

⇒ preconditioner is built using the default options.

**Iterative Solution**

\[
\text{≫ [x, options] = AMGsolver(A, PREC, options, b);
}
\]

⇒ system is solved.

**Release Memory**

\[
\text{≫ 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**

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

⇒ preconditioner is built using the default options.

**Iterative Solution**

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

⇒ system is solved.

**Release Memory**

```matlab
≫ PREC = AMGdelete(PREC);
```
**Parameter setting**

*ILUPACK* offers many, many parameters

### Some Default Parameters

```matlab
≫ 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'
npostsmoothing:1
ncourse: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
```

M. Bollhöfer  |  ILUPACK toolbox for MATLAB  |  Page 7
Parameter setting — don’t try to read all that!

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

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

droptolc threshold to drop small entries in the constraint part (saddle point problems)
elevator recommended memory space to keep the preconditioner (relative to the number of nonzeros of A)
complement
type tv type of test vector ('static' or 'dynamic')
tv keep the ILU exact for this test vector
amg type of multilevel algorithm ('ilu', 'aml', 'amg')
npre smoothing number of pre-smoothing steps
npost smoothing number of post-smoothing steps
nc coarse 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')
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCpart</td>
<td>a priori separation of unknowns into fine and coarse grid nodes ('yes', 'no')</td>
</tr>
<tr>
<td>typecoarse</td>
<td>type of coarse grid system (approximate Schur complement, 'ilu', 'amg')</td>
</tr>
<tr>
<td>damping</td>
<td>damping factor for Jacobi smoothing</td>
</tr>
<tr>
<td>contraction</td>
<td>contraction factor for the local residual on the coarse grid system (when flexible solvers are used)</td>
</tr>
<tr>
<td>ind</td>
<td>index indicator to identify a saddle point structure</td>
</tr>
<tr>
<td>mixedprecision</td>
<td>single precision preconditioner, double precision solver</td>
</tr>
<tr>
<td>coarsereduce</td>
<td>discard (1,2) block and (2,1) in the multilevel ILU</td>
</tr>
<tr>
<td>decoupleconstraints</td>
<td>decouple constraint in a saddle point problem</td>
</tr>
</tbody>
</table>
How to set up your own parameters

- Your iterative solver does not converge
  ⇒ reduce options.droptol, options.droptolS

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

- You would accept more iteration steps
  ⇒ increase options.maxit

- You would like to have a more accurate solution
  ⇒ decrease options.restol
How to set up your own parameters

- Your iterative solver does not converge
  ⇒ reduce \texttt{options.droptol}, \texttt{options.droptolS}

- You want to provide your own initial guess
  ⇒ call \texttt{[x, options] = AMGsolver(A, PREC, options, b, x0)};

- You would accept more iteration steps
  ⇒ increase \texttt{options.maxit}

- You would like to have a more accurate solution
  ⇒ decrease \texttt{options.restol}
How to set up your own parameters

- Your iterative solver does not converge
  ⇒ reduce `options.droptol`, `options.droptolS`

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

- You would accept more iteration steps
  ⇒ increase `options.maxit`

- You would like to have a more accurate solution
  ⇒ decrease `options.restol`
How to set up your own parameters

- Your iterative solver does not converge
  ⇒ reduce `options.droptol`, `options.droptolS`

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

- You would accept more iteration steps
  ⇒ increase `options.maxit`

- You would like to have a more accurate solution
  ⇒ 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 $A \rightarrow \Pi^T D_r AD_c$ satisfies $|a_{ij}^{new}| \leq 1$, $|a_{ii}^{new}| = 1$.

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

![Graphs showing matchings and symmetrization process]
Outline

- Introduction — using the ILUPACK toolbox
  - Preconditioning Systems
  - Getting started

- What’s behind the toolbox
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting

- Tools
  - Visualization
  - 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
  - Visualization
  - Further tools
  - Automatic structure detection

- Closing Remarks
Inverse-based pivoting

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

- Postponed updates become the coarse grid system
- Algebraic Multilevel Method
Inverse-based pivoting

\[ \kappa = 3 \rightarrow 4 \text{ levels} \quad \kappa = 10 \rightarrow 2 \text{ levels} \]

\[ \kappa = 100 \rightarrow 1 \text{ level} \]
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
\[ \text{[PREC, options]} = \text{AMGfactor(A, options)}; \]
Display ILU
\[ \text{AMGspy(PREC)}; \]
Display reordered & rescaled original system
\[ \text{AMGspy(A, PREC)}; \]
Outline

- Introduction — using the ILUPACK toolbox
  - Preconditioning Systems
  - Getting started

- What’s behind the toolbox
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting

- Tools
  - Visualization
  - Further tools
  - Automatic structure detection

- Closing Remarks
### Further tools

#### Number of nonzeros

```matlab
≫ nz=AMGnnz(PREC);
```

#### Solve a single preconditioned system

```matlab
≫ x = AMGsol(PREC,b);
```

#### Load/save matrix in Harwell-Boeing format

```matlab
≫ [A,rhs,rhstyp]=loadhbo(filename);
≫ savehbo(filename, A);
(also available for right side, initial guess, . . . )
```

#### Simplified QMR

```matlab
≫ [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] = mwrmrcm(A);  [pl,pr,Dl,Dr] = mwrmamd(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] = symmwmmd(A);
Outline

- Introduction — using the ILUPACK toolbox
  - Preconditioning Systems
  - Getting started

- What’s behind the toolbox
  - Matchings
  - Symmetric reorderings
  - Inverse-based pivoting

- Tools
  - Visualization
  - 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**

\[ \text{options.isdefinite}=1; \]
\[ \text{options}=\text{AMGinit}(A,\text{options}); \]

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**

\[ \text{PREC} = \text{AMGconvert}(\text{PREC}); \]

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)