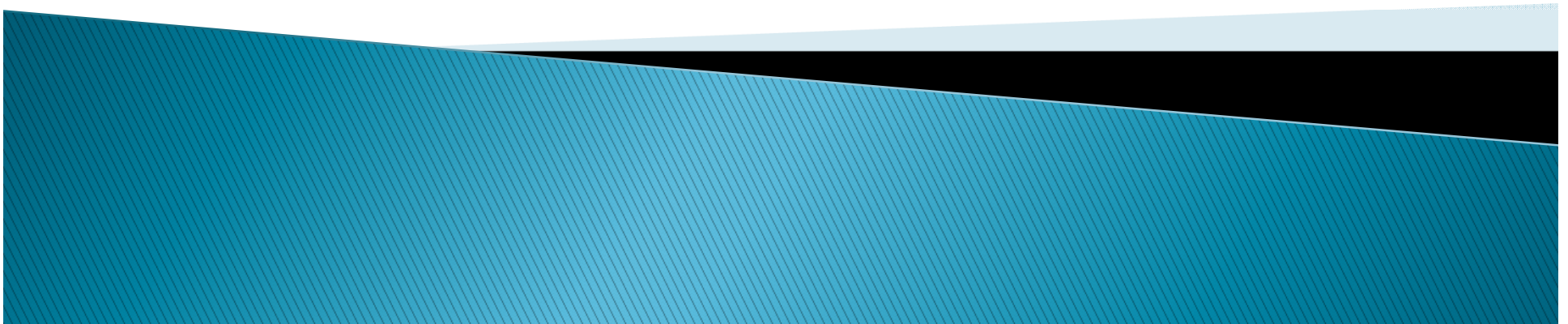


Experiments on ILU(0) Preconditioning

Andrew Rice, Cagri Abis, Kyle Bordeau



Overview

- ▶ Background and Theory
- ▶ Code
- ▶ Results
- ▶ Conclusions



Experiments on ILU(0) Preconditioning

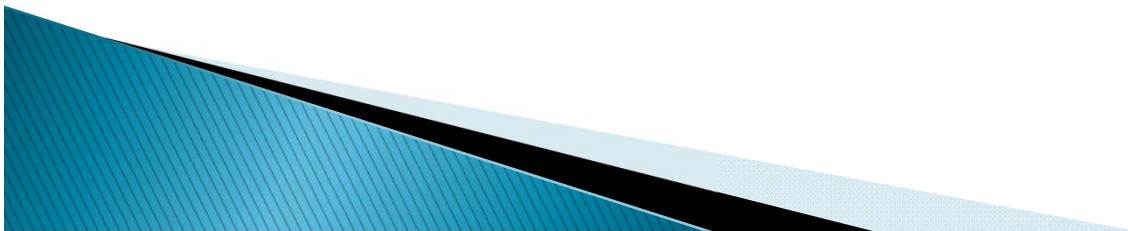
»» Background and Theory

Background and Theory

- ▶ The condition number is defined as:

$$K_P(A) = \|A\|_P \|A^{-1}\|_P$$

- ▶ If K_P is big, then matrix multiplication will result in large precision loss.
- ▶ For iterative solvers, this loss will become cumulative.
- ▶ Preconditioning helps to make K_P small, preserving the precision.



Background and Theory: ILU TYPES

▶ ILU(0)

Specified with:

- ▶ Setup.type='nofill'
- ▶ If desired milu option can be selected
- ▶ Does not have pivoting

▶ ILUC

Specified with:

- ▶ Setup.type='crout'
- ▶ Setup.droptol=...
- ▶ If desired milu option can be selected
- ▶ Does not have pivoting

▶ ILUTP

Specified with:

- ▶ Setup.type='ilutp'
- ▶ Setup.thresh=...
- ▶ Performs pivoting
- ▶ Threshold value between 0 and 1



Background and Theory

- ▶ Modified ILU factorization (milu)
 - Compensates diagonal of U to conserve row or column sums
 - Row

$$Ae = L U e$$

$$e = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

- Column

$$e^T A = e^T L U$$



Background and Theory

▶ DROPTOL

- This setup command sets the drop tolerance of an incomplete LU factorization.
- For MATLAB, the default is 0, producing the complete LU factorization.

▶ For U:

- The nonzero entries of U satisfy:
 $\text{abs}(U(i,j)) \geq \text{droptol} * \text{norm}(A(:,j)),$

▶ For L:

- Entries are tested against local drop tolerance before being scaled by the pivot
- so for nonzeros in L: $\text{abs}(L(i,j)) \geq \text{droptol} * \text{norm}(A(:,j)) / U(j,j)$

▶ Diagonal entries are kept regardless of value.



Experiments on ILU(0) Preconditioning

»» Code

Code

```
for h=1:1.0e4
    C=sprand(h,h,1.0e-5); %Creates a sparse matrix of size h
    S=C*C'+speye(size(C)); %Matrix C symmetric positive definite
    tic %Starts timer
    [L,U]=ilu(S); %Runs code with matrix S and returns L,U
    time(h)=toc; %Stops timer
    count(h)=h; %Counter for plots
    nz_S(h)=nnz(S); %Nonzeros in the ILU of S
    nz_L(h)=nnz(L); %Non zeros in L
    nz_U(h)=nnz(U); %Non zeros in U
end
```

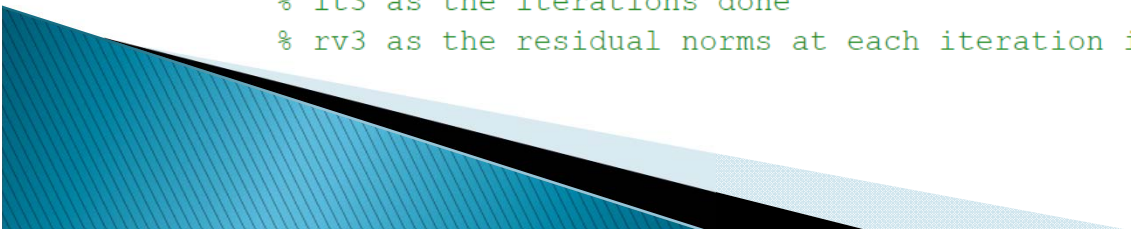
Sample code for solver times

Code

```
%Algorithm
%A is the sparse matrix and b is the left hand side as: A*x=b
setup.type='nofill';
[L,U]=ilu(A,setup);
%% BiConjugate Gradient
%tolerance is defined with tol, and maxit is the maximum iteration desired
[x1,fl1,rr1,it1,rv1] = bicg(A,b,tol,maxit,L,U);
%this returns x1 as the solution, fl1 as convergence flag
% rr1 as relative residual norm(b-A*x)/norm(b)
% it1 as the iterations done
%rv1 as the residual norms at each iteration including norm(b-A*x0)
%% Conjugate Gradient

[x2,fl2,rr2,it2,rv2] = cgs(A,b,tol,maxit,L,U);
%this returns x2 as the solution, fl2 as convergence flag
% rr2 as relative residual norm(b-A*x)/norm(b)
% it2 as the iterations done
% rv2 as the residual norms at each iteration including norm(b-A*x0)

%% Generalized Minimum Residuals
%for the input a restart option can be specified
[x3,fl3,rr3,it3,rv3] = gmres(A,b,[],tol,maxit,L,U);
%this returns x3 as the solution, fl3 as convergence flag
% rr3 as relative residual norm(b-A*x)/norm(b)
% it3 as the iterations done
% rv3 as the residual norms at each iteration including norm(b-A*x0)
```



Code

```
n=logspace(1,3,4);
%defining setups for 3 commands
setc.type='crout'; %crout version
setc.milu='row';
setc.droptol=0.1;
setn.type='nofill'; %nofill version
setn.milu='row';
setp.type='ilutp'; %ilutp version
crout_norm=zeros(length(n),1);
nofill_norm=crout_norm;
ilutp_norm=crout_norm;
for i=1:length(n)
    m=floor(n(i));
    R = sprand(m,m,1e-2); % defining the density and matrix size for a sparse matrix
    S=R*R'+speye(size(R)); % avoiding zero diagonal entries for a symmetric pos. def
matrix

    %solving for crout
    [lc,uc] = ilu(S,setc);
    crout_norm(i)=norm(S-lc*uc,1);

    %solving for nofill
    [ln,un] = ilu(S,setn);
    nofill_norm(i) = norm(S-ln*un,1);

    %solving for ilutp
    [lp,up] = ilu(S,setp);
    ilutp_norm(i)=norm(S-lp*up,1);

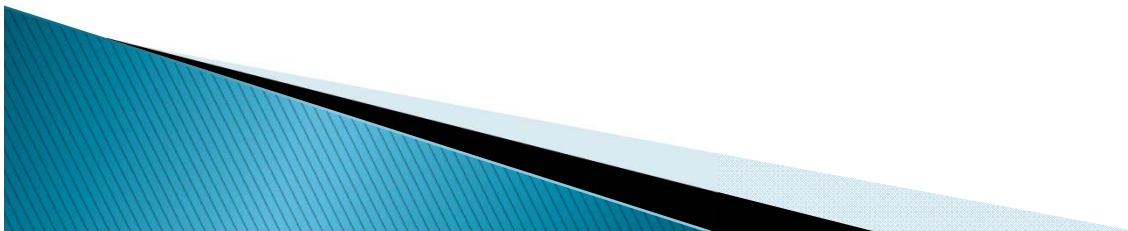
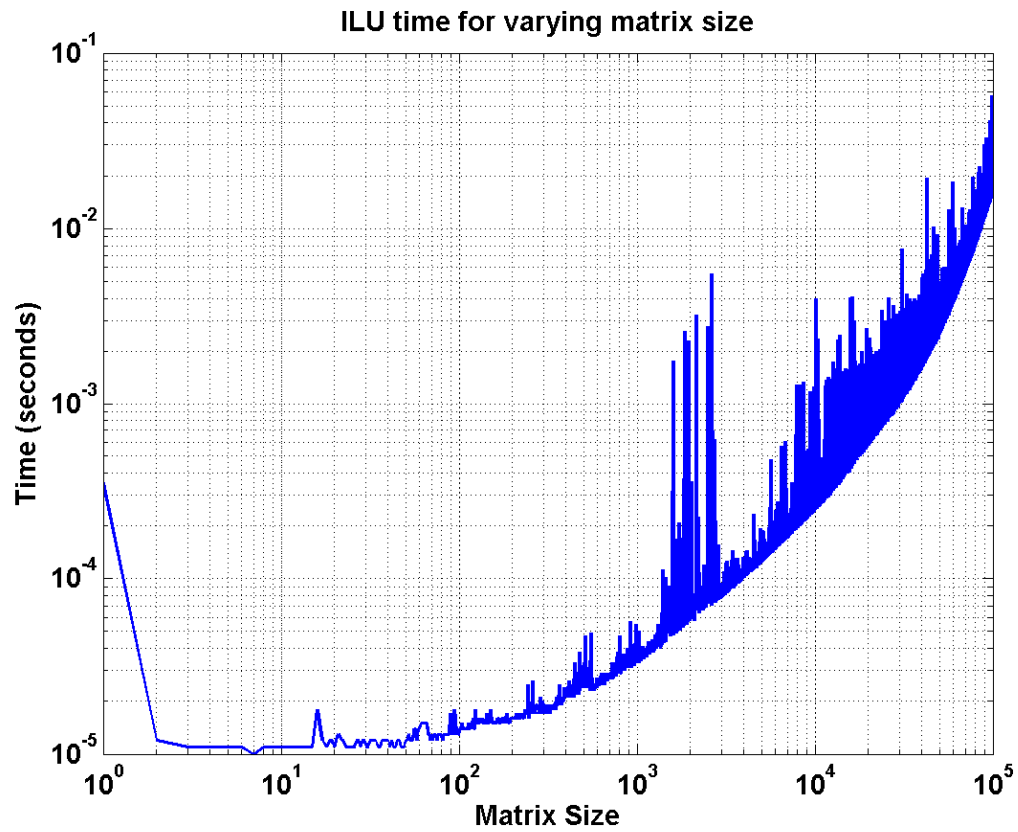
end
```



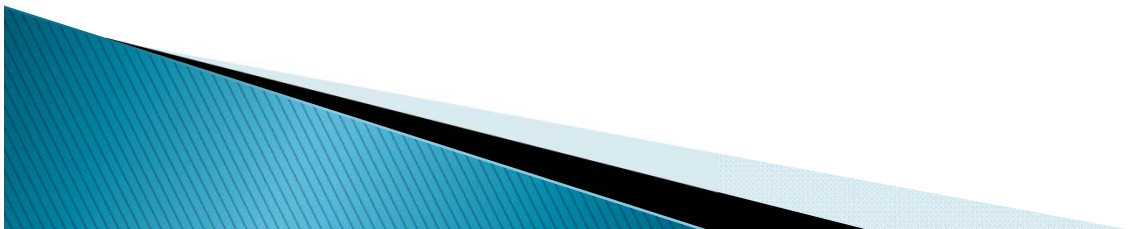
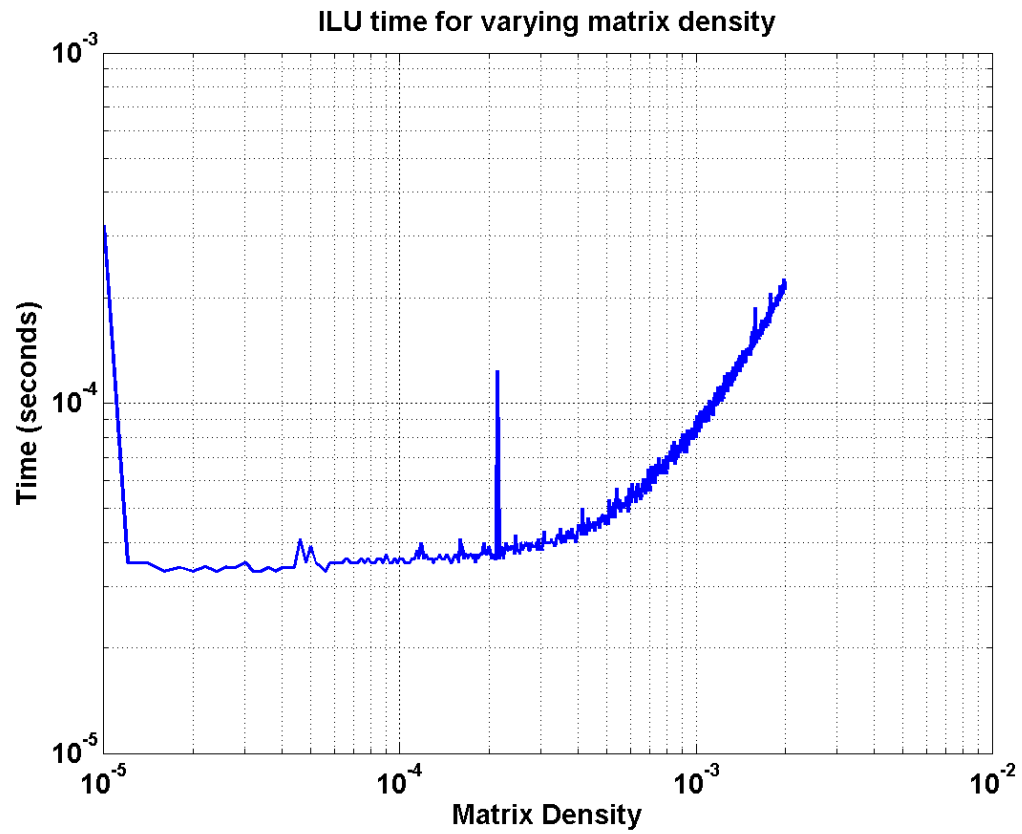
Experiments on ILU(0) Preconditioning

»» Results

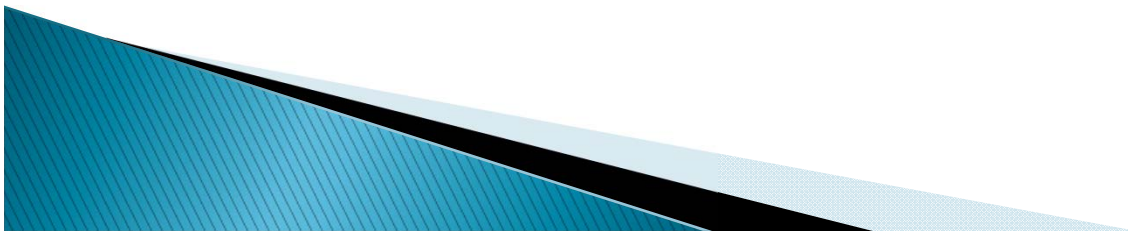
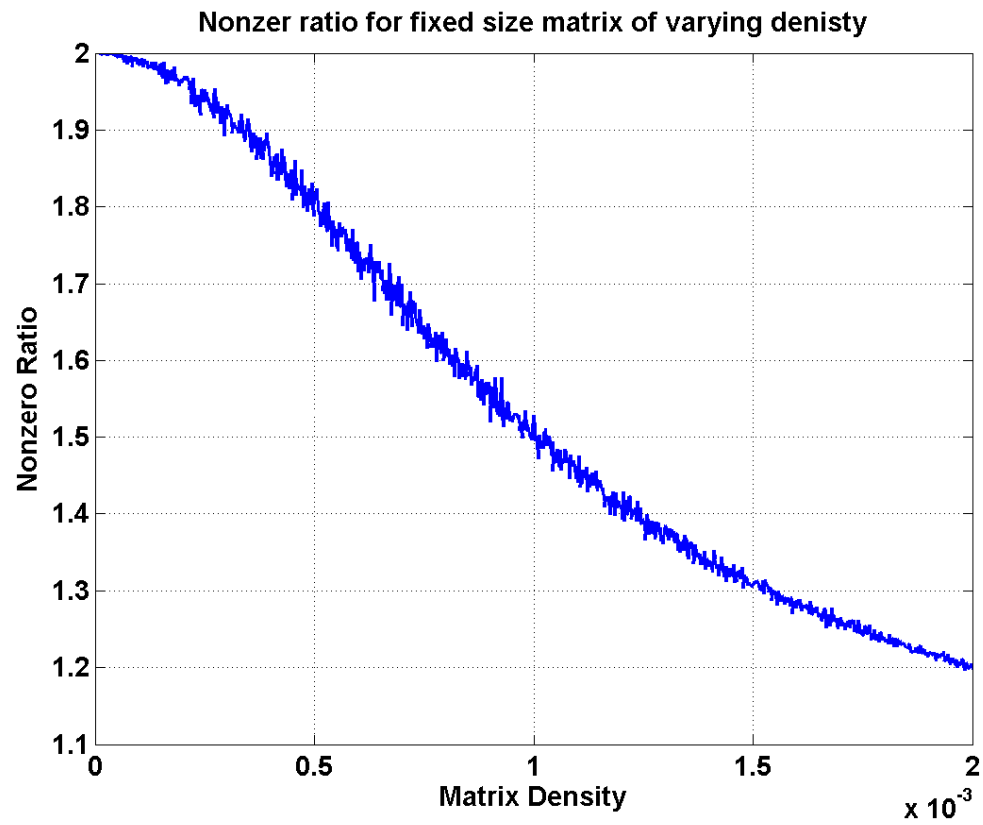
Results



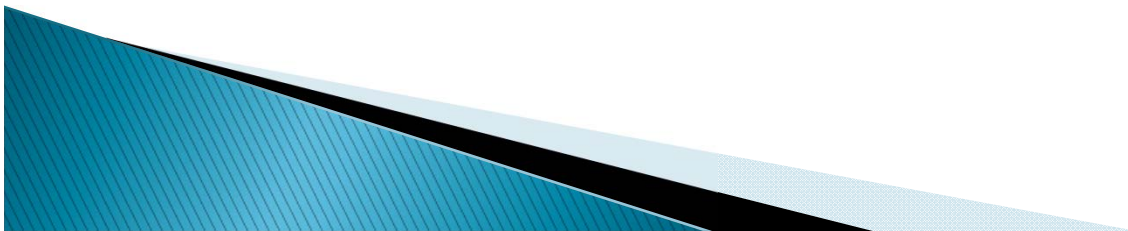
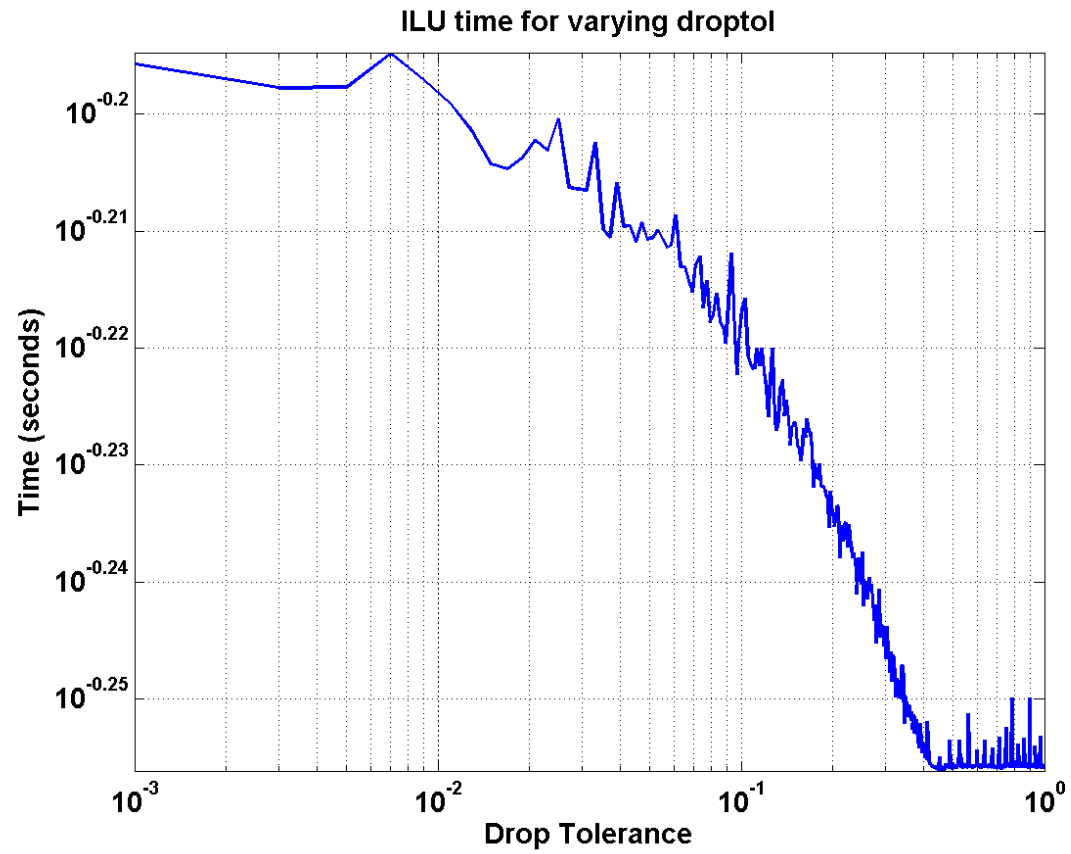
Results



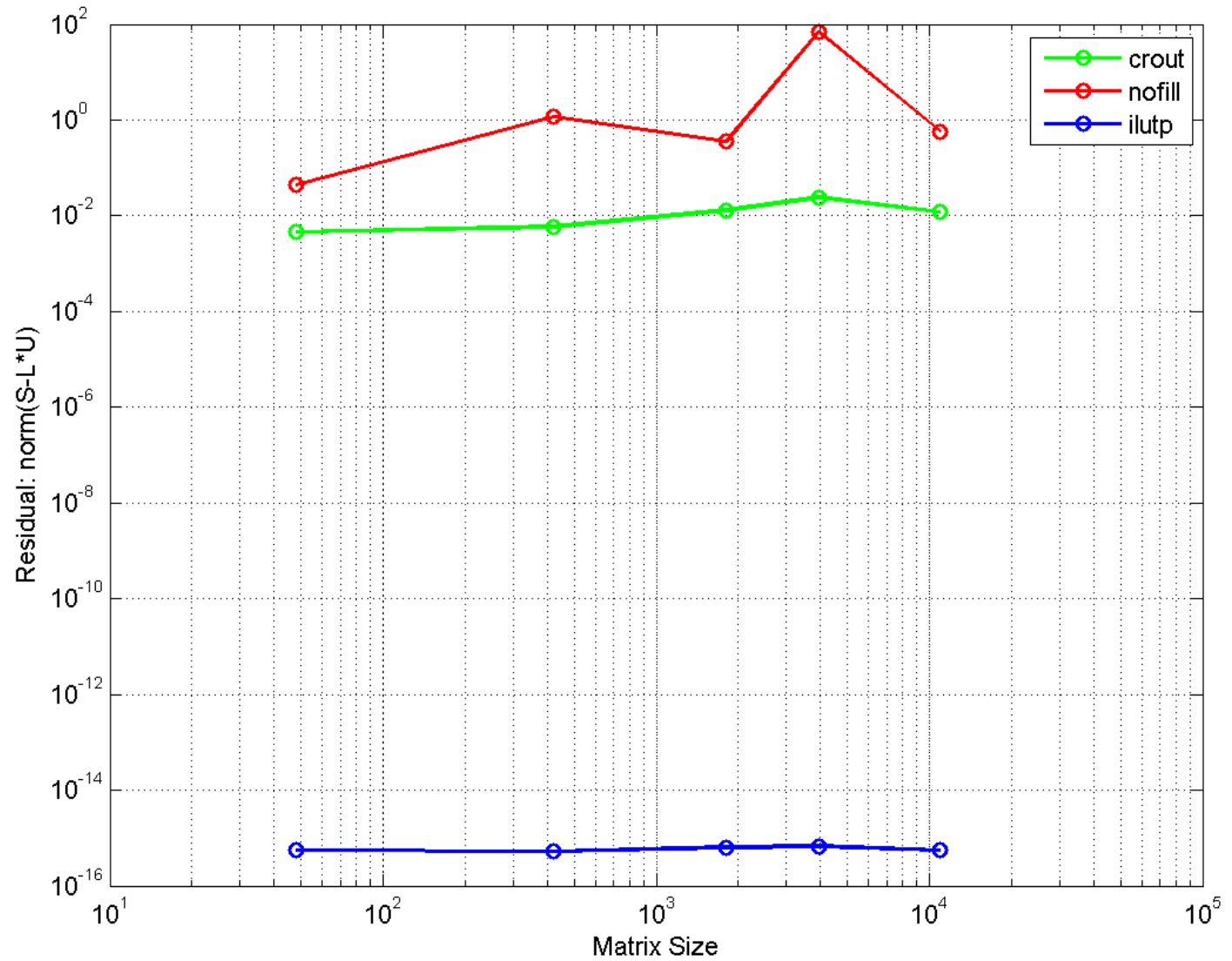
Results



Results



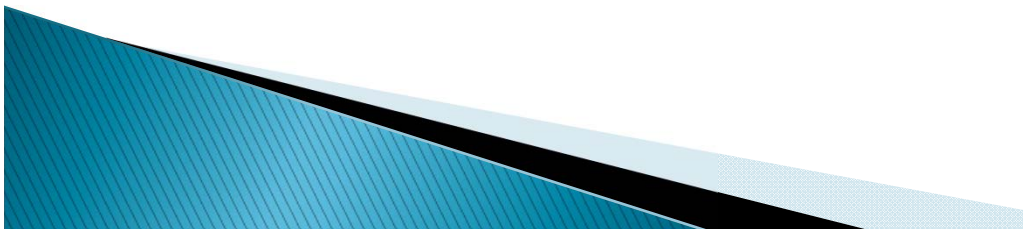
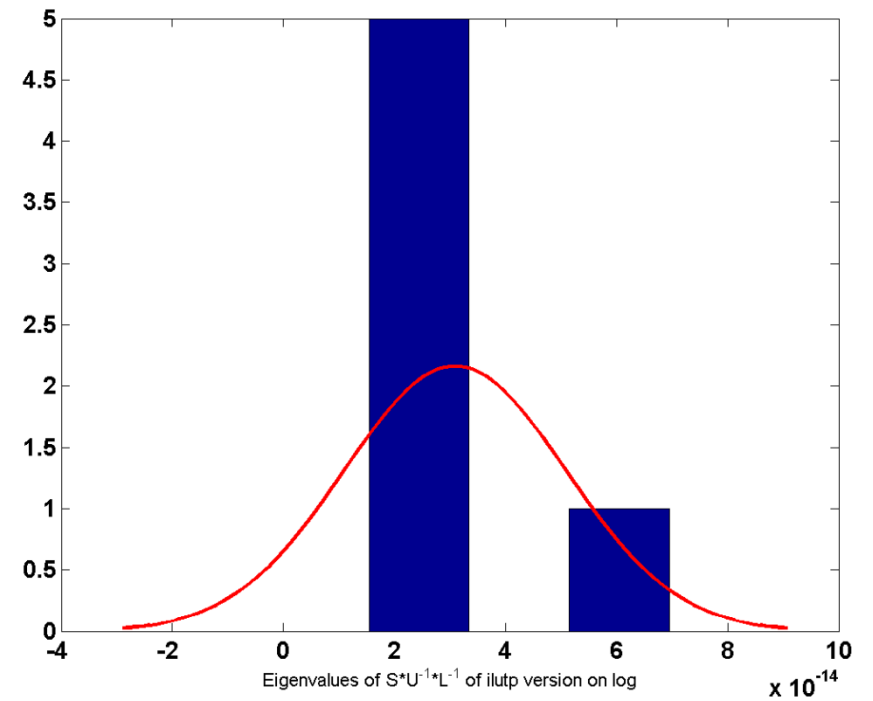
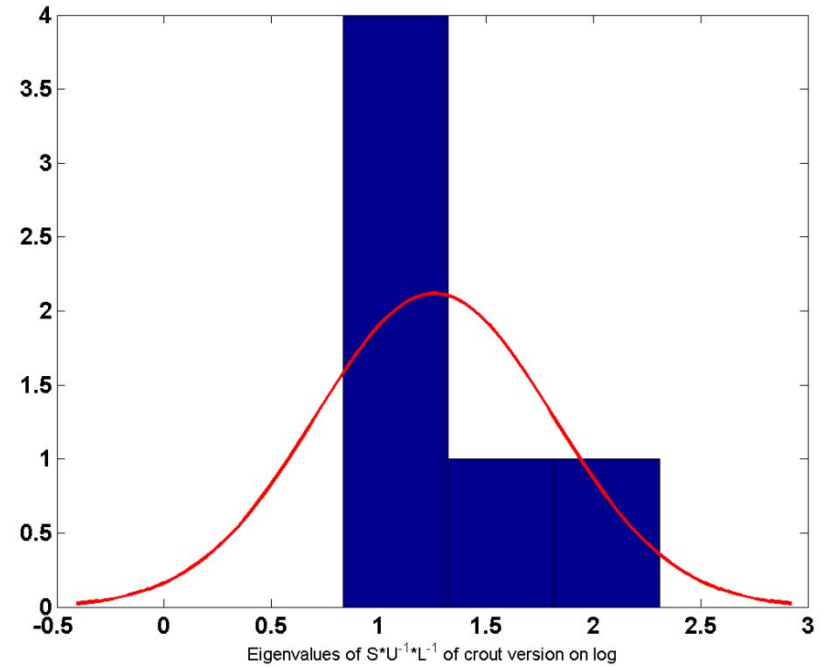
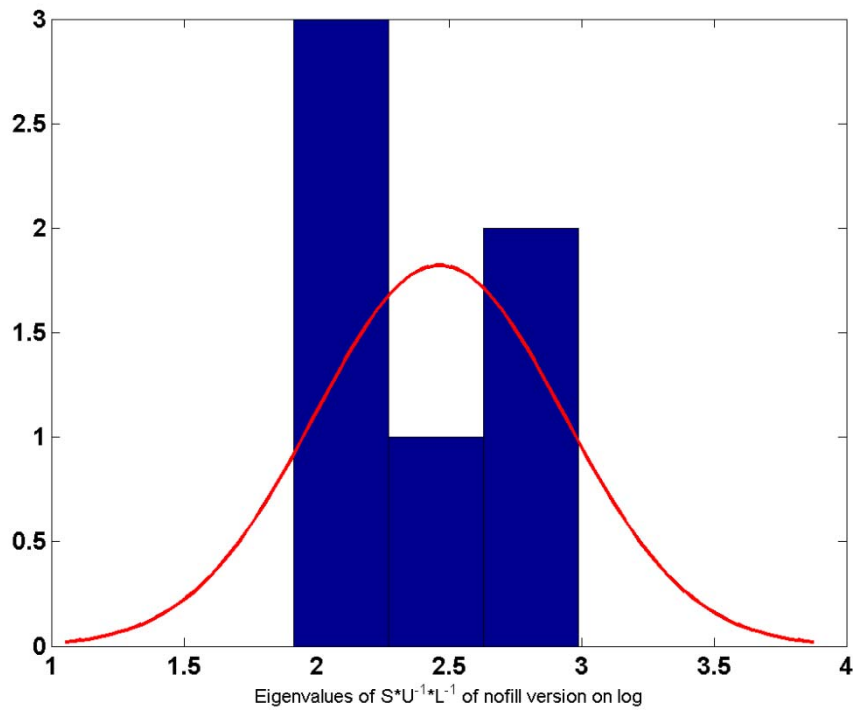
Results: Residuals of S-LU



► Residuals of real life matrices from different types of ILU

Residuals of Eigs

Matrix size: 1000x1000
Density: 1e-2



Results

- ▶ 5 different sized symmetric positive definite matrices
- ▶ Conjugate gradient, biconjugate gradient, and GMRES to solve

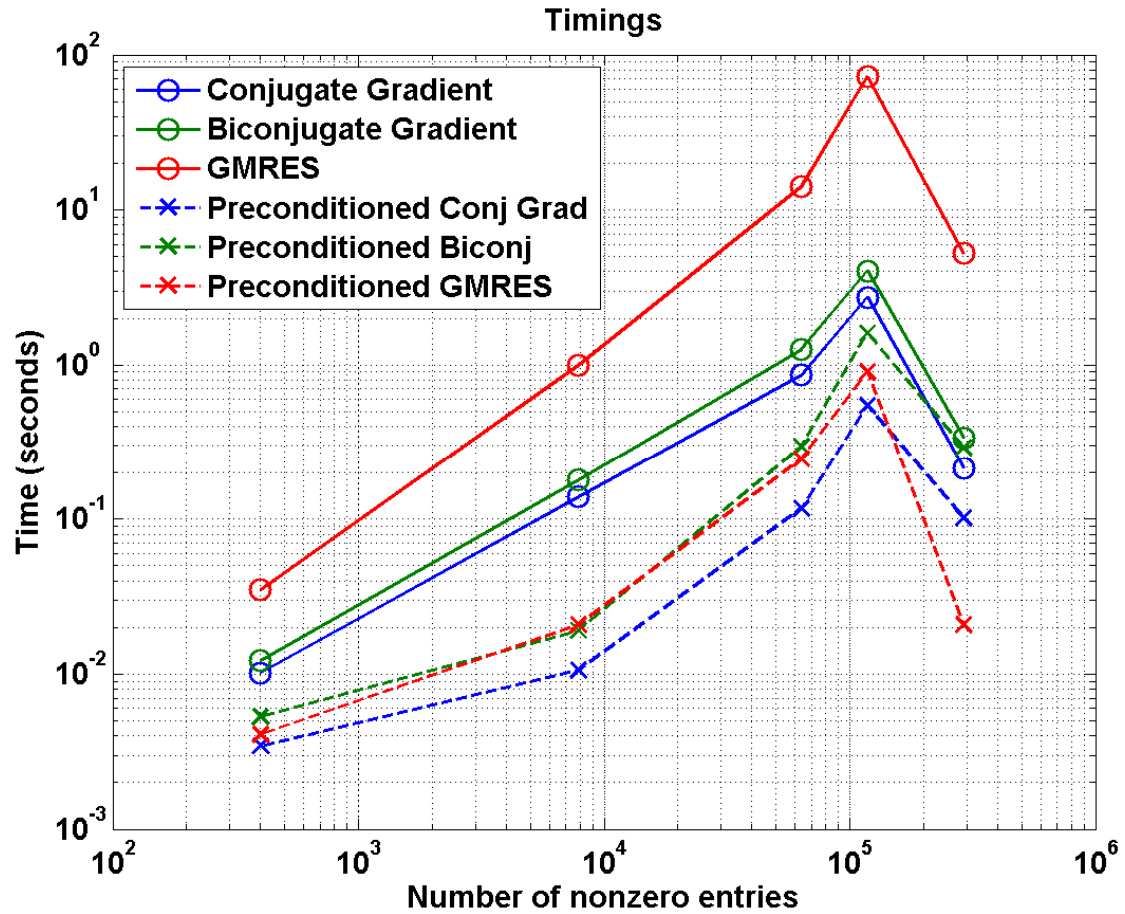
$$Ax = b$$

$$x = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

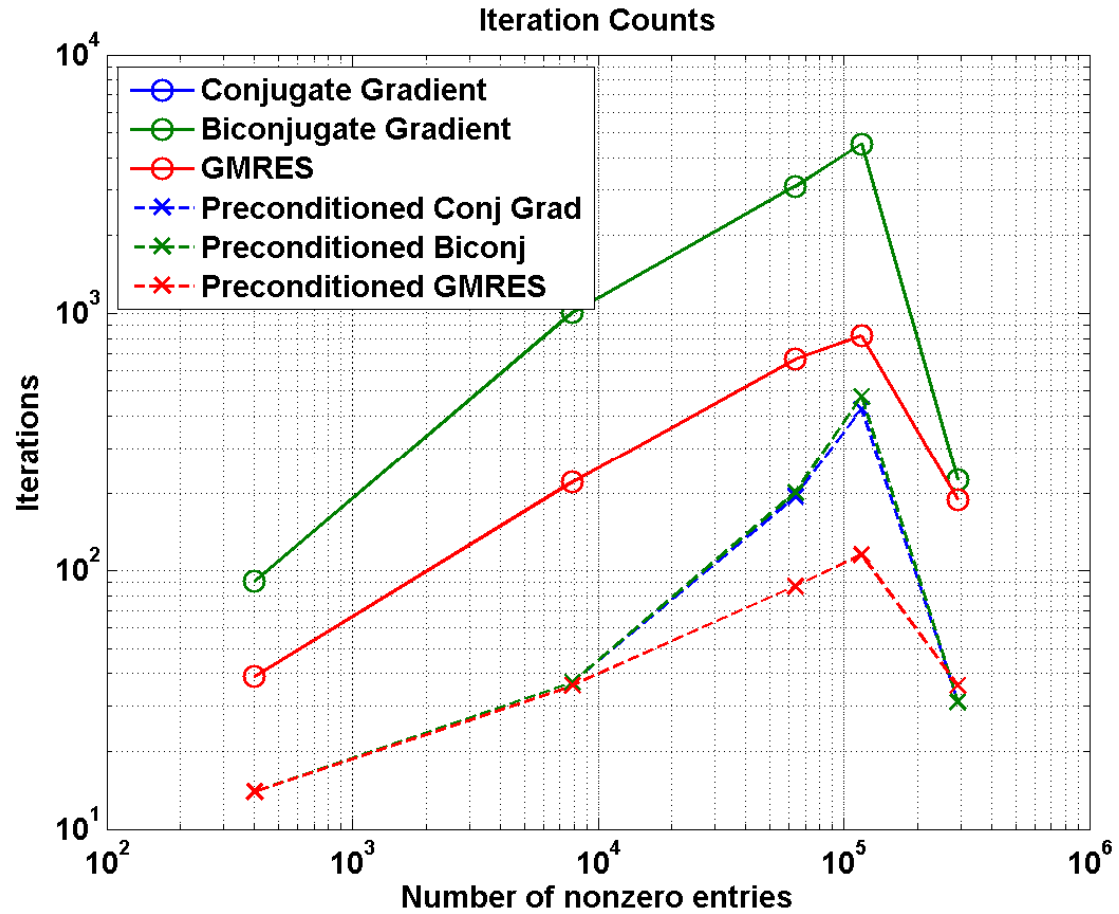
- ▶ Used L and U as preconditioners from ilu(0)



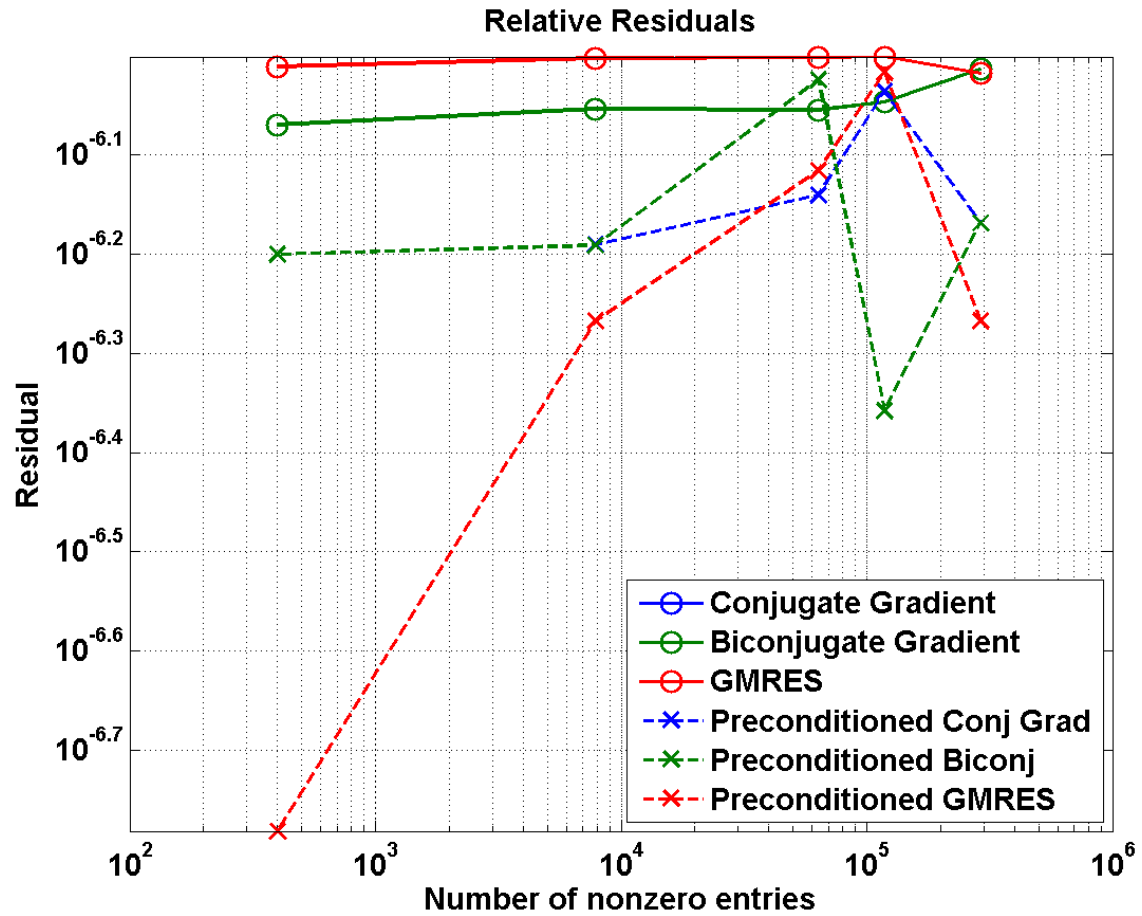
Results



Results

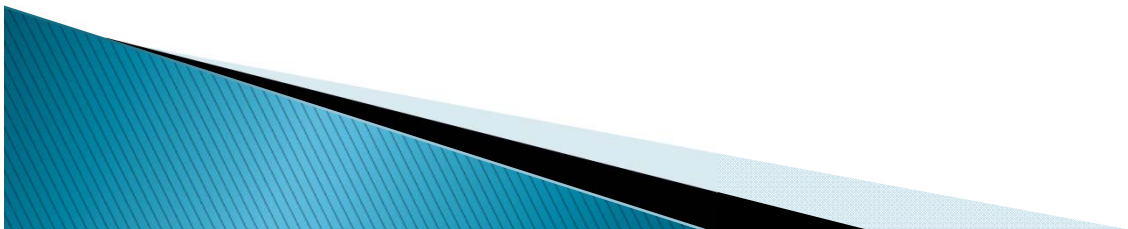


Results



Results

- ▶ Test specifying milu
 - 1806x1806 symmetric positive definite matrix, 63454 non zero entries



Results

		Modified Incomplete LU Option		
		Off	Row	Column
		Time (seconds)		
Solver	Conjugate Gradient	0.0249	0.0015	0.0001
	Biconjugate Gradient	0.0625	0.0014	0.0012
	GMRES	0.0445	0.0071	0.0069
		Relative Residual		
Solver	Conjugate Gradient	9.66E-07	3.95E-16	6.03E-16
	Biconjugate Gradient	9.05E-07	4.12E-16	6.04E-16
	GMRES	7.66E-07	2.54E-10	4.27E-10

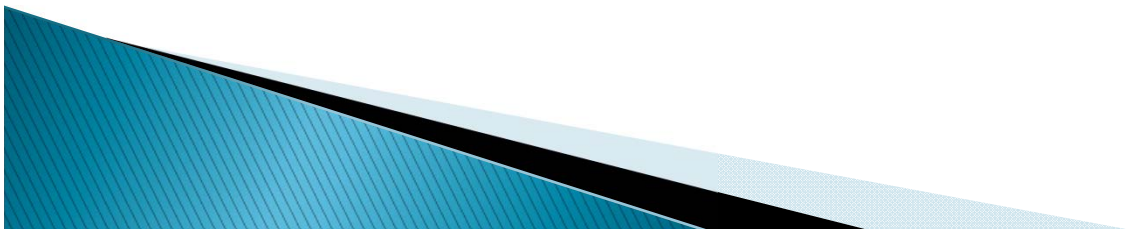


Experiments on ILU(0) Preconditioning

»» Conclusions

Conclusions

- ▶ Timings on ILU are a function of matrix size, density, and ILU drop tolerance
- ▶ Residuals of ILU increases with the matrix size and density
- ▶ Preconditioning using ILU(0)
 - Decreases time
 - Decreases iterations
 - No significant effect on residuals
- ▶ Specifying milu greatly improves performance



Experiments on ILU(0) Preconditioning

»» Questions?