Computing the rank of big sparse matrices modulo p using gaussian elimination

Charles Bouillaguet [1] Claire Delaplace [2]

[12]CRIStAL, Université de Lille

[2]IRISA, Université de Rennes 1

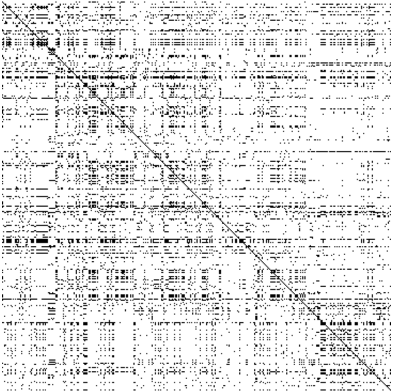
JNCF, 16 janvier 2017

Background

Sparse Linear Algebra
Modulo p (coefficients : int)

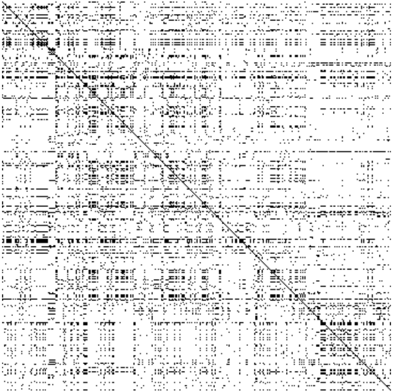
Operations

- Rank
- Linear systems
- Kernel
- etc...



Background

Sparse Linear Algebra
Modulo p (coefficients : int)



Operations

- Rank
- Linear systems
- Kernel
- etc...

Two families of Algorithms

- **Direct methods** (Gaussian Elimination, LU, ...)
- **Iterative methods** (Wiedemann, Lanczos...)

Related Work

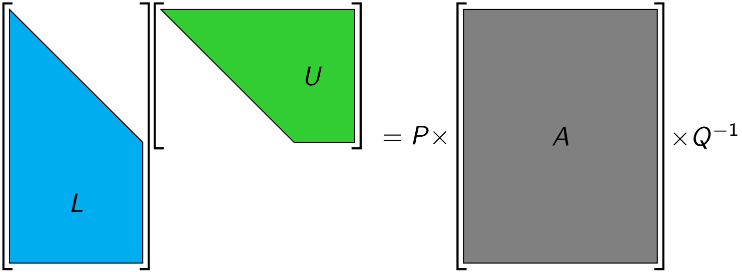
Algorithms

- Comparison between a sparse gaussian elimination and the Wiedmann algorithm: [Dumas & Villard 02]
- Direct methods in the numerical world (e.g. [Davis 06])
- Pivots selection heuristic for Gröbner Basis Matrices [Faugère & Lacharte 10]

Software

- Exact: *not much* (LinBox, GBLA (*Gröbner basis*), MAGMA)
- Numeric: *many* (SuperLU, UMFPACK, ...)

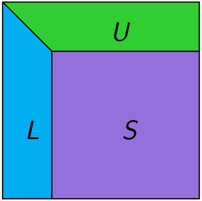
PLUQ Factorization



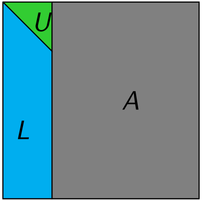
- L has non zero diagonal
- U has unit diagonal
- A can be rectangular
- A can be rank deficient

Right-looking and Left-looking LU

Usual right-looking Algorithm:

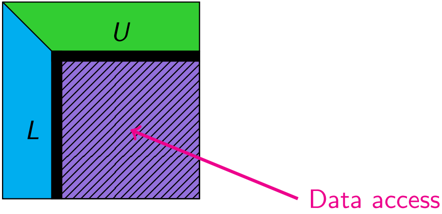


Left-looking GPLU Algorithm
[Gilbert & Peierls 88]

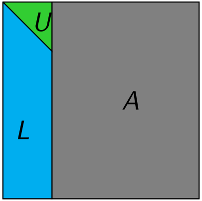


Right-looking and Left-looking LU

Usual right-looking Algorithm:

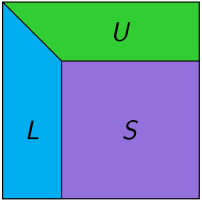


Left-looking GPLU Algorithm [Gilbert & Peierls 88]

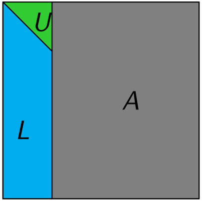


Right-looking and Left-looking LU

Usual right-looking Algorithm:

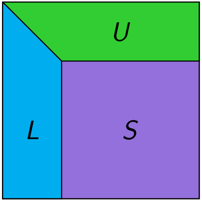


Left-looking GPLU Algorithm [Gilbert & Peierls 88]



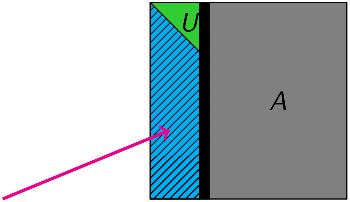
Right-looking and Left-looking LU

Usual right-looking Algorithm:



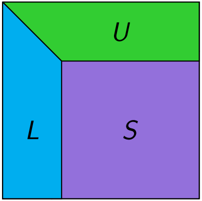
Left-looking GPLU Algorithm [Gilbert & Peierls 88]

Data access

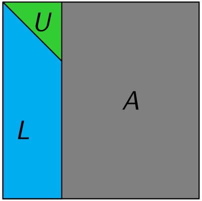


Right-looking and Left-looking LU

Usual right-looking Algorithm:

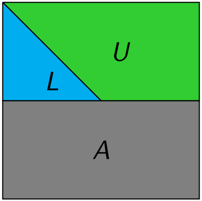


Left-looking GPLU Algorithm [Gilbert & Peierls 88]



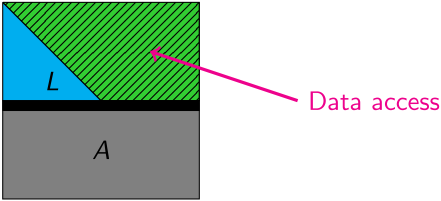
An up-looking variant of the GPLU

- Row-by-row version
- Adapted to Computer Algebra



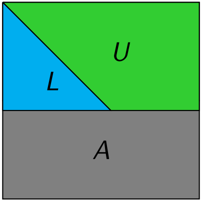
An up-looking variant of the GPLU

- Row-by-row version
- Adapted to Computer Algebra



An up-looking variant of the GPLU

- Row-by-row version
- Adapted to Computer Algebra



GPLU Algorithm: Application to Exact Linear Algebra

- Never been used before for exact computations
- We implemented it
- We benchmarked it against LinBox (sparse right-looking)

Our Benchmarks show:

- GPLU work best when U is very sparse
- Sometimes GPLU outperform the right-looking algorithm (often)
- Sometimes the right-looking algorithm outperform GPLU (less often)

$\Longrightarrow$ Can we take advantage of both methods?

Our Work: A New Hybrid Algorithm (CASC 2016)

Description

- Find many pivots without performing any arithmetical operations.
- Compute the Schur complement S, using an up-looking algorithm.
- Compute the rank of S.

Our Work: A New Hybrid Algorithm (CASC 2016)

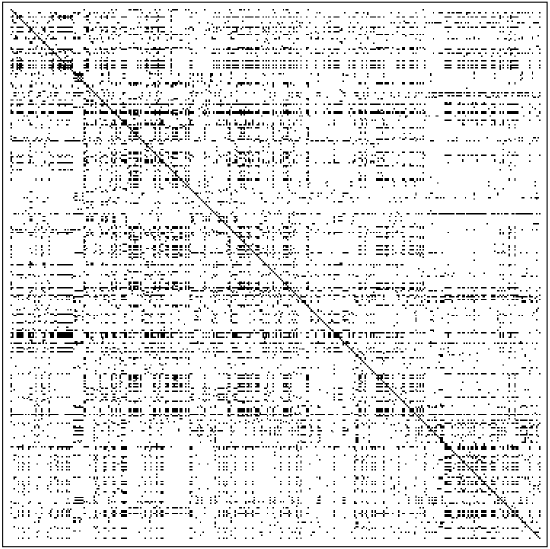
Description

- Find many pivots without performing any arithmetical operations.
- Compute the Schur complement S, using an up-looking algorithm.
- Compute the rank of S.

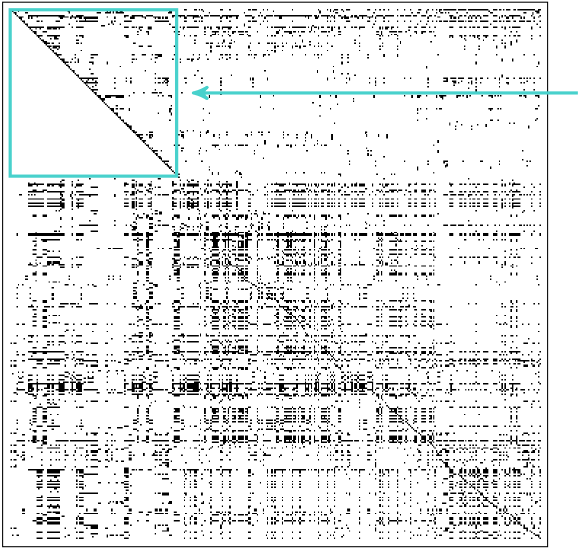
Rank of S

- Recurse
- Dense rank computation
- Wiedemann Algorithm
- ...

Example:

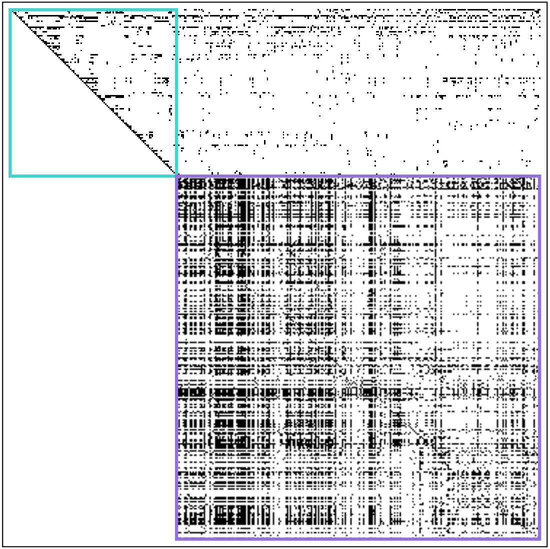


Example:



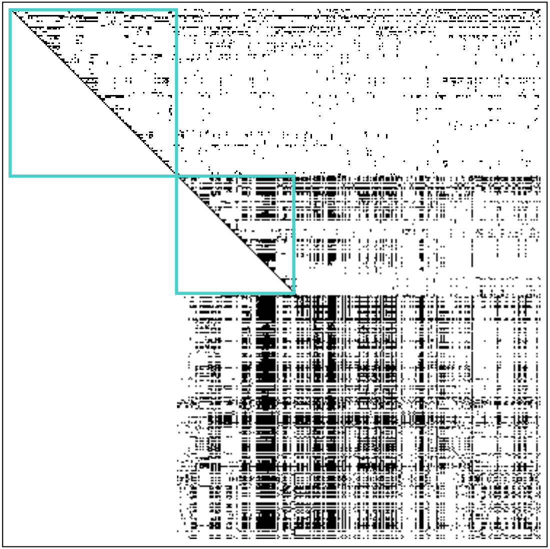
Set of pivots found without any arithmetical operations

Example:

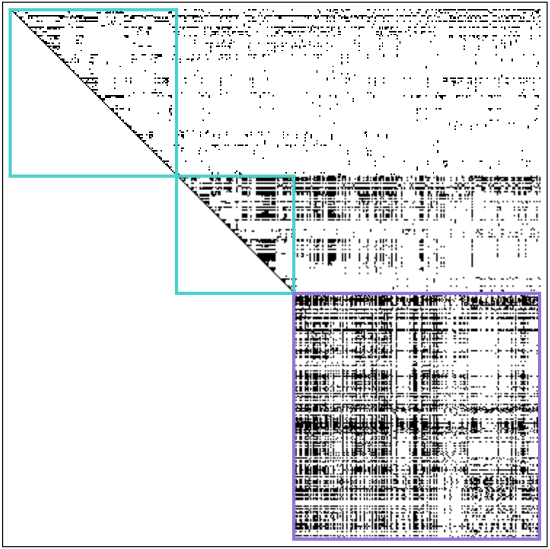


← Schur Complement

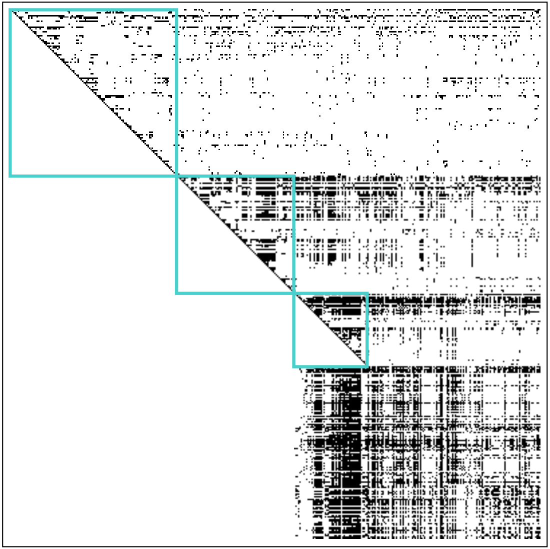
Example:



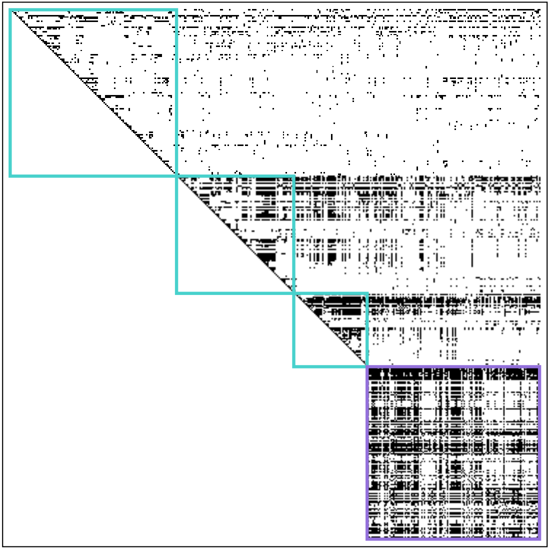
Example:



Example:

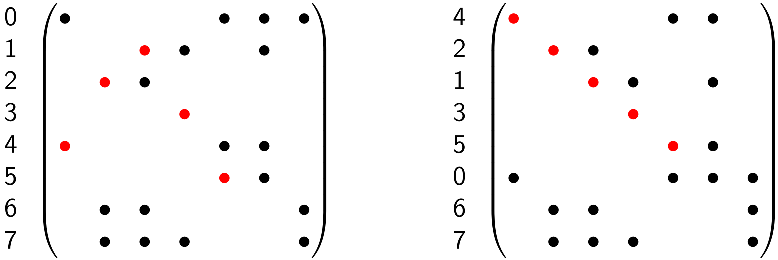


Example:



← Parallelizable

Initial Pivots Selection Heuristic [Faugère & Lachartre 10]



Description

- Each row is mapped to the column of its leftmost coefficient.
- When several rows have the same leftmost coefficient, select the sparsest.
- Move the selected rows before the others and sort them by increasing position of the leftmost coefficient.

Schur Complement Computation

P denotes the permutation that pushes the pre-computed pivots in the top of A. Ignoring permutation over the columns of A:

$$PA = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix}$$

Schur Complement Computation

P denotes the permutation that pushes the pre-computed pivots in the top of A. Ignoring permutation over the columns of A:

$$PA = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix}$$

Definition

The **Schur Complement** S of PA with respect to U_{00} is given by :

$$S = A_{11} - A_{10} U_{00}^{-1} U_{01}$$

Schur Complement Computation

P denotes the permutation that pushes the pre-computed pivots in the top of A. Ignoring permutation over the columns of A:

$$PA = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix}$$

Definition

The **Schur Complement** S of PA with respect to U_{00} is given by :

$$S = A_{11} - A_{10} U_{00}^{-1} U_{01}$$

Denote by $(\mathbf{a}_{i0}\ \mathbf{a}_{i1})$ the i-th row of $(A_{10}\ A_{11})$, and consider the following system :

$$(\mathbf{x}_0\ \mathbf{x}_1) \cdot \begin{pmatrix} U_{00} & U_{01} \\ & Id \end{pmatrix} = (\mathbf{a}_{i0}\ \mathbf{a}_{i1})$$

Schur Complement Computation

P denotes the permutation that pushes the pre-computed pivots in the top of A. Ignoring permutation over the columns of A:

$$PA = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix}$$

Definition

The **Schur Complement** S of PA with respect to U_{00} is given by :

$$S = A_{11} - A_{10} U_{00}^{-1} U_{01}$$

Denote by $(\mathbf{a}_{i0}\ \mathbf{a}_{i1})$ the i-th row of $(A_{10}\ A_{11})$, and consider the following system :

$$(\mathbf{x}_0\ \mathbf{x}_1) \cdot \begin{pmatrix} U_{00} & U_{01} \\ & Id \end{pmatrix} = (\mathbf{a}_{i0}\ \mathbf{a}_{i1})$$

We obtain $\mathbf{x}_1 = \mathbf{a}_{i1} - \mathbf{a}_{i0} U_{00}^{-1} U_{01}$. $\mathbf{x}_1$ **is the i-th row of** S

Schur Complement Computation

In a Nutshell:

- Each row of S can be computed independently
- Guess if S is sparse or dense by computing some random rows
- If S sparse : compute S using the previous method
- The Schur complement computation is parallelizable

Performance of the Hybrid Algorithm (CASC 2016)

Experiments carried on an Intel Core i7-3770 with 8 GB of RAM
Experiments carried on all matrices from SIMC with integer coeffcients
Only one core used

Hybrid versus Right-looking (Linbox) and GPLU (time in s)

Matrix	Right-looking	GPLU	Hybrid
GL7d/GL7d24	34	276	11.6
Margulies/cat_ears_4_4	3	184	0.1
Homology/ch7-8.b4	173	0.2	0.2
Homology/ch7-8.b5	611	45	10.7

Hybrid versus Wiedmann (time in s)

Matrix	Wiedmann	Hybrid
M0,6-D7	20397	0.8
relat8	244	2
relat9	176694	2024

New: Improvement of the Pivots Selection Heuristic

Why

- Enable us to reduce the number of the elimination steps
- Keep U sparse $\Rightarrow$ fast elimination steps

New: Improvement of the Pivots Selection Heuristic

Why

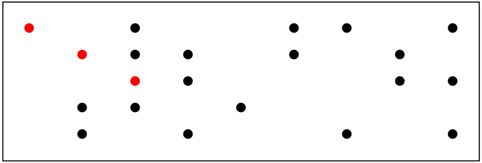
- Enable us to reduce the number of the elimination steps
- Keep U sparse $\Rightarrow$ fast elimination steps

How

- Find the largest set: NP-Complete
- Find a large set using heuristics

Our idea: First find the F.-L. pivots, then search for more.

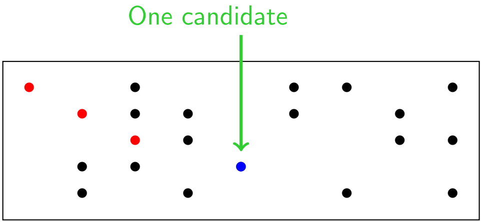
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

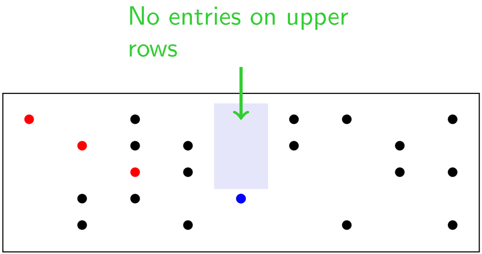
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG ⇒ If there is a cycle, the entry can't be selected.

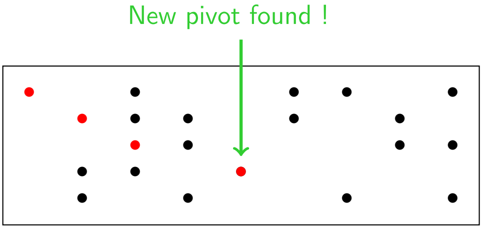
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

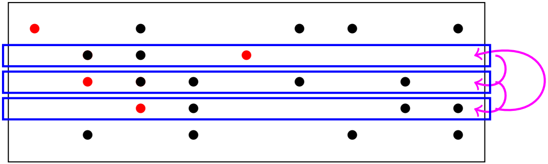
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

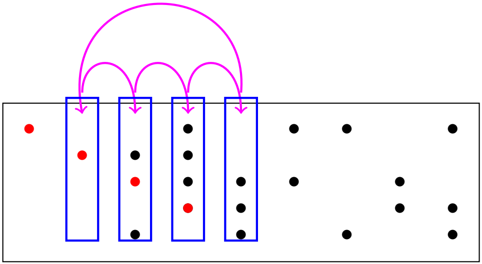
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG ⇒ If there is a cycle, the entry can't be selected.

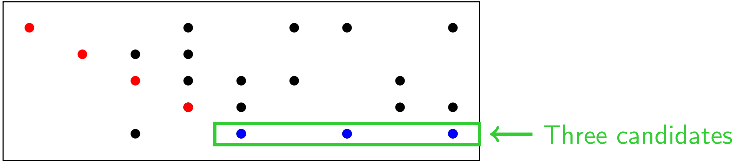
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

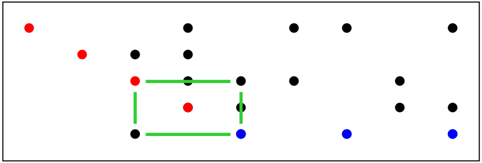
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

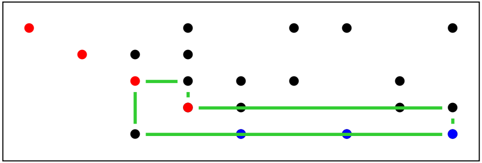
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

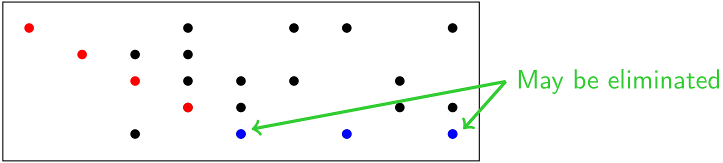
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

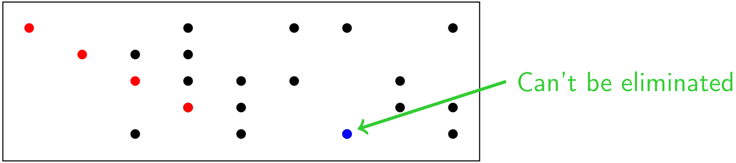
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG ⇒ If there is a cycle, the entry can't be selected.

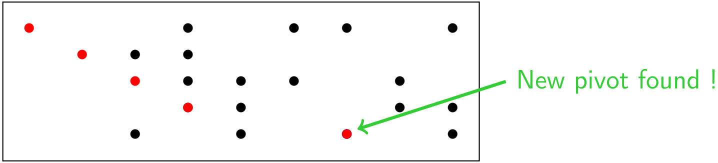
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

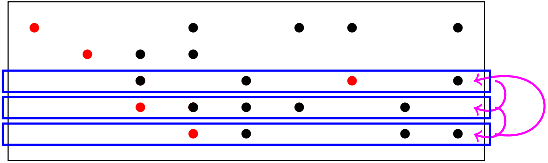
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG ⇒ If there is a cycle, the entry can't be selected.

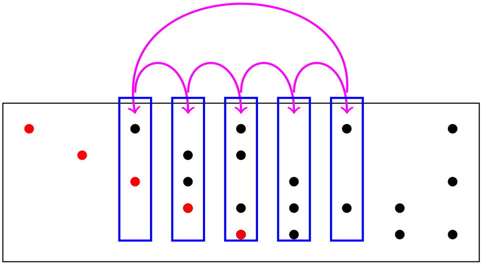
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

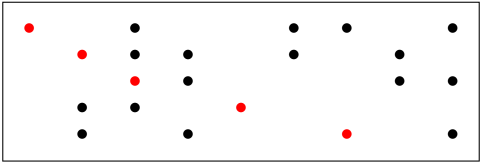
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

Our new pivot selection algorithm



In a Nutshell:

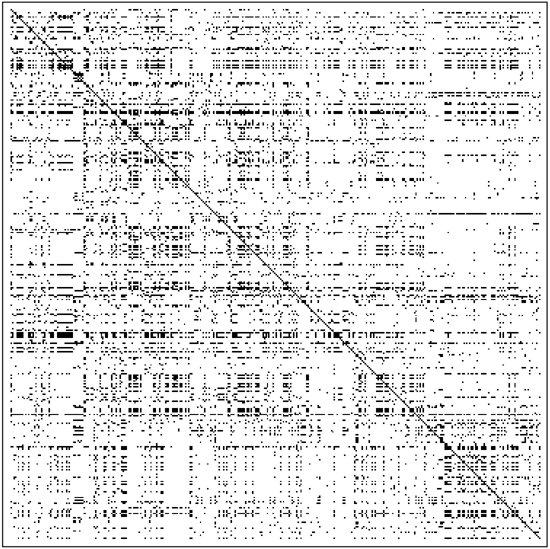
- Find the F.-L. pivots
- Push the F.-L. pivots on the top of A.
- For all non-pivotal column j, if the upmost coefficient a_{ij} is on a non-pivotal row, select a_{ij}
- Perform a graph search to find more pivots

About Graph Search

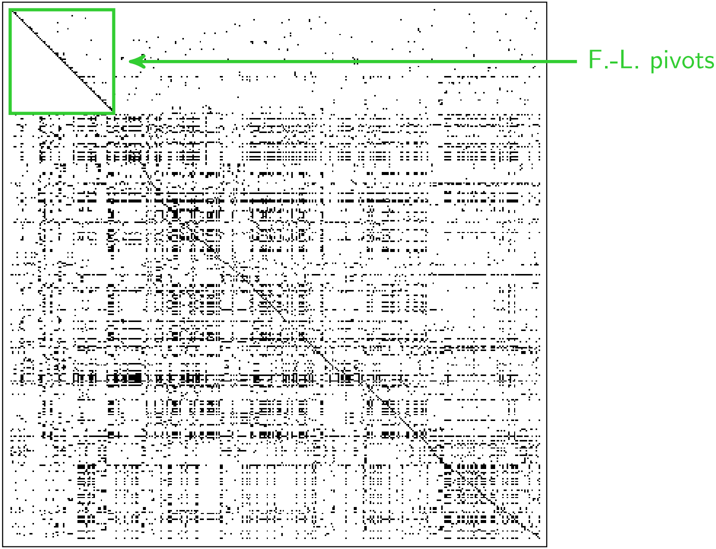
Remark

- This method is parallelizable.
- The worst case complexity of the search is $\mathcal{O}(n|A|)$ (same as Wiedemann)
- In practice : much faster than Wiedmann

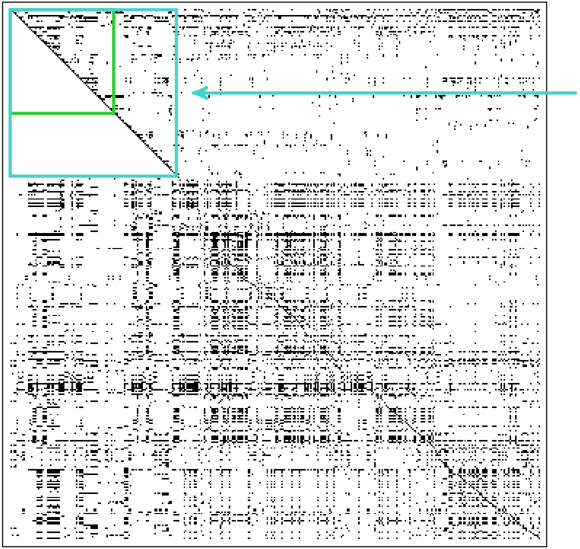
Example



Example



Example



More pivots after graph search

Conclusion

- Implemented in C and C++ in the SpaSM (SPArse direct Solver Modulo p) library and publicy available at:

  `https://github.com/cbouilla/spasm`

- Will be implemented in LinBox

Conclusion

- Implemented in C and C++ in the SpaSM (SPArse direct Solver Modulo p) library and publicy available at:

  `https://github.com/cbouilla/spasm`

- Will be implemented in LinBox

Open questions:

- Can we still improve the pivots selection?
- Is it possible to adapt this method on matrices from CADO-NFS?

Thank you for your time !