Comparison of Methods for Solving Sparse Linear Systems

About the Solving of Systems of Linear Equations

Willi Braun, Kaja Balzereit, Bernhard Bachmann

FH Bielefeld
University of
Applied Sciences

University of Applied Sciences Bielefeld
Bielefeld, Germany

February 2, 2015

Motivation

I love deadlines. I like the whooshing sound they make as they fly by.

*Douglas Adams, *1952 †2001*

- Discussion on this topic raised after the last OpenModelica Workshop.
- Status at that time:
  - Linear torn system solved with a non-linear solver.
  - All non torn systems were solved always just by lapack.
- Question: Why you don't use sparse linear solvers?

Motivation

I love deadlines. I like the whooshing sound they make as they fly by.

*Douglas Adams, *1952 †2001*

- Discussion on this topic raised after the last OpenModelica Workshop.
- Status at that time:
  - Linear torn system solved with a non-linear solver.
  - All non torn systems were solved always just by lapack.
- Question: Why you don't use sparse linear solvers?

Outline

FH Bielefeld
University of
Applied Sciences

Transformation steps for simulation

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{\dot{x}}(t), \underline{y}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{z}(t), \underline{u}(t), \underline{p}, t),\ \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$$\Downarrow$$

$$\underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \underline{g}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\underline{\dot{x}}(t) = \underline{h}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

$$\underline{y}(t) = \underline{k}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

Transformation example

$$\begin{aligned} f_2(z_2) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_3(z_2, z_3, z_5) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \\ f_1(z_3, z_4) &= 0 \end{aligned}$$

Algebraic loop (SCC)

$$\begin{aligned} f_3(z_2, z_3, z_5) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \end{aligned}$$

General form

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 \;\; &+ \ldots + a_{1n}x_n = b_1 \\
a_{21}x_1 + a_{22}x_2 \;\; &+ \ldots + a_{2n}x_n = b_2 \\
\vdots \qquad\qquad\quad &\qquad\qquad\;\; \vdots \;\; = \;\; \vdots \\
a_{m1}x_1 + a_{m2}x_2 &+ \ldots + a_{mn}x_n = b_m
\end{aligned}
$$

in compact matrix form:

$$Ax = b$$

Example

$$
\begin{aligned}
2x - 2y &= 4 \\
-2x + 4y &= 0
\end{aligned}
$$

$\Rightarrow$ solution $(4, 2)$

Systems of Linear Equations

Solution set

A linear system may behave in any one of three possible ways:

- The system has no solution.
- The system has infinitely many solutions.
- The system has a single unique solution.

Solution set

A linear system may behave in any one of three possible ways:

- The system has no solution.
- The system has infinitely many solutions.
- The system has a single unique solution.

No Solution

- System is inconsistent.
- System is overdetermined.

Example

$$4x + 2y = 4$$
$$4x + 2y = 2$$

$$x + y = 4$$
$$x + 4y = 2$$
$$4x + 2y = 0$$

Solution set

A linear system may behave in any one of three possible ways:

- The system has no solution.
- The system has infinitely many solutions.
- The system has a single unique solution.

Infinitely many solutions

- System is known as an under-determined system.
- System as linear dependent equations.

Example

$$
\begin{aligned}
x + 2y + 2z &= 2 \\
2x + 4y + 4z &= 4 \\
4x + 2y + z &= 0
\end{aligned}
$$

Systems of Linear Equations

Solution set

A linear system may behave in any one of three possible ways:

- The system has no solution.
- The system has infinitely many solutions.
- The system has a single unique solution.

Single unique solution

- System is regular.
- If the $rg(A) = rg(A, b)$.

Example

$$\begin{aligned} 2x - 2y &= 4 \\ -2x + 4y &= 0 \end{aligned}$$

Systems of Linear Equations

Solving a linear system

- Elimination of variables
- Cramer's rule
- Matrix solution (Inverse A)
- LU decomposition
- Iterative methods
- [...]

LU decomposition

Factorization is performed by replacing any row in A by a linear combination of itself and any other row.

$$Ax = b = LUx$$

$$\Leftrightarrow$$

$$L(Ux) = b$$

Solution by substitution:

$$Ly = b$$

$$Ux = y$$

Solving a linear system

- Elimination of variables
- Cramer's rule
- Matrix solution (Inverse A)
- LU decomposition
- Iterative methods
- [...]

LU decomposition

Factorization is performed by replacing any row in A by a linear combination of itself and any other row.

$$Ax = b = LUx$$

$$\Leftrightarrow$$

$$L(Ux) = b$$

Solution by substitution:

$$Ly = b$$

$$Ux = y$$

Example

$$A = LU$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix} \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix}$$

- Without proper permutations of A, the factorization may fail.
- If $a_{11} = 0$, then l_{11} or u_{11} has to be zero, which implies either L or U is singular, since $a_{11} = l_{11}u_{11}$.

Pivoting

- Factorization without pivoting is numerical unstable, or even not possible.
  - interchange rows (partial pivoting)
  - interchange rows and cols (full pivoting)
- Partial pivoting is easier than full pivoting, since one don't need to track permutations.

Example

$$A = LU$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix} \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix}$$

- Without proper permutations of A, the factorization may fail.
- If $a_{11} = 0$, then l_{11} or u_{11} has to be zero, which implies either L or U is singular, since $a_{11} = l_{11}u_{11}$.

Pivoting

- Factorization without pivoting is numerical unstable, or even not possible.
  - interchange rows (partial pivoting)
  - interchange rows and cols (full pivoting)
- Partial pivoting is easier than full pivoting, since one don't need to track permutations.

Example

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$\Leftrightarrow$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & 0 & \hat{a}_{23} \\ 0 & 0 & \hat{a}_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ \hat{b}_2 \\ \hat{b}_3 \end{pmatrix}$$

Repair a singular A Matrix

- Perform full pivoting, that moves the singular rows and cols to the end.
- If corresponding b_i entries are almost zero, we can choose x_i also as zero.
- That fixes the simulation of a remaining MSL Mechanics example: MultiBody.Examples.Elementary. PointGravityWithPointMasses2

Example

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$\Leftrightarrow$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & 0 & \hat{a}_{23} \\ 0 & 0 & \hat{a}_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ \hat{b}_2 \\ \hat{b}_3 \end{pmatrix}$$

$$\begin{pmatrix} a_{11} & a_{13} & a_{12} \\ 0 & a_{33} & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ \hat{b}_3 \\ 1e^{-16} \end{pmatrix}$$

Repair a singular A Matrix

- Perform full pivoting, that moves the singular rows and cols to the end.
- If corresponding b_i entries are almost zero, we can choose x_i also as zero.
- That fixes the simulation of a remaining MSL Mechanics example: MultiBody.Examples.Elementary. PointGravityWithPointMasses2

- Compressed Sparse Row (CSR) format
  - Store non-zero values, corresponding column
  - pointer into value array corresponding to first non-zero(nz) in each row
  - Storage required $= 2nz + n$
- Frontal methods
  - Variant of Gauss elimination that automatically avoids operations involving zero terms.
  - Based on symbolical analysis of the sparsity pattern.
  - Based on proper permutation of the matrix.

Example

$$J = \begin{pmatrix} 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & * \\ * & 0 & * & 0 & * \\ 0 & * & 0 & 0 & 0 \\ 0 & * & 0 & * & * \end{pmatrix}$$

$$A_i = [0, 1, 2, 5, 6, 9]$$

$$A_p = [5, 5, 1, 3, 5, 2, 2, 4, 5]$$

$$A_x = [*, *, *, *, *, *, *, *, *]$$

Why tearing belong to this topic?

- Tearing reduce the size of systems.
- Reduction makes the system more dense.
- It is a kind of sparsity exploitation.

Why tearing belong to this topic?

- Tearing reduce the size of systems.
- Reduction makes the system more dense.
- It is a kind of sparsity exploitation.

Example

$$F_1 : z_1 + f(x) z_3 - 6 = 0$$
$$F_2 : f(y) z_3 - z_2 + 3 z_1 = 0$$
$$F_3 : 3 z_1 + 2 z_2 + z_3 = 0$$

Why tearing belong to this topic?

- Tearing reduce the size of systems.
- Reduction makes the system more dense.
- It is a kind of sparsity exploitation.

Example

$$F_1 : z_1 + f(x) z_3 - 6 = 0$$
$$F_2 : f(y) z_3 - z_2 + 3 z_1 = 0$$
$$F_3 : 3 z_1 + 2 z_2 + z_3 = 0$$

Select z_3 as tearing variable

$$F_1 : z_1 = -f(x) z_3 + 6$$
$$F_2 : z_2 = f(y) z_3 + 3 z_1$$
$$F_3 : z_3 = z_1 + 2 z_2$$
$$\Leftrightarrow z_3 = 42$$

Why tearing belong to this topic?

- Tearing reduce the size of systems.
- Reduction makes the system more dense.
- It is a kind of sparsity exploitation.

Example

$$F_1 : z_1 + f(x) z_3 - 6 = 0$$
$$F_2 : f(y) z_3 - z_2 + 3 z_1 = 0$$
$$F_3 : 3 z_1 + 2 z_2 + z_3 = 0$$

Select z_3 as tearing variable

$$F_1 : z_1 = -f(x) z_3 + 6$$
$$F_2 : z_2 = f(y) z_3 + 3 z_1$$
$$F_3 : z_3 = z_1 + 2 z_2$$
$$\Leftrightarrow z_3 = 42$$

Notes

- Now we need only solve linear for z_3
- symbolical jacobians are used to calculate $\frac{\partial F3}{\partial z_3}$
- Once $z3$ has been found, the first two assignments return $z1, z2$.

Usage in OpenModelica

- -ls <solver_name>
- `-lv LOG_LS, LOG_LS_V`
- `-lv LOG_STATS_V`
- compiler flag to activate/de-activate linear tearing.

Usage in OpenModelica

- Solver Name
  - `lapack`
  - `totalpivot`
  - `umfpack`
  - `lis`

Dense Solvers

- LAPACK
  - Standard linear solver package.
  - Well known library for solving linear systems.
  - What else to say about it?
- Total pivot
  - Self written linear solver for full control.
  - Abilities to solve under-determined systems.

Sparse Solvers

- UMFPACK
  - Sparse solver suite.
  - Direct sparse linear solver.
  - Well tested and heavily used(e.g. Matlab).
- Lis
  - Iterative linear solvers suite.
  - Iterative solvers work only for well-posed problems.
  - This libraries works better for a really big N.

Dense Solvers

- LAPACK
  - Standard linear solver package.
  - Well known library for solving linear systems.
  - What else to say about it?
- Total pivot
  - Self written linear solver for full control.
  - Abilities to solve under-determined systems.

Sparse Solvers

- UMFPACK
  - Sparse solver suite.
  - Direct sparse linear solver.
  - Well tested and heavily used(e.g. Matlab).
- Lis
  - Iterative linear solvers suite.
  - Iterative solvers work only for well-posed problems.
  - This libraries works better for a really big N.

Dense Solvers

- LAPACK
  - Standard linear solver package.
  - Well known library for solving linear systems.
  - What else to say about it?
- Total pivot
  - Self written linear solver for full control.
  - Abilities to solve under-determined systems.

Sparse Solvers

- UMFPACK
  - Sparse solver suite.
  - Direct sparse linear solver.
  - Well tested and heavily used(e.g. Matlab).
- Lis
  - Iterative linear solvers suite.
  - Iterative solvers work only for well-posed problems.
  - This libraries works better for a really big N.

Dense Solvers

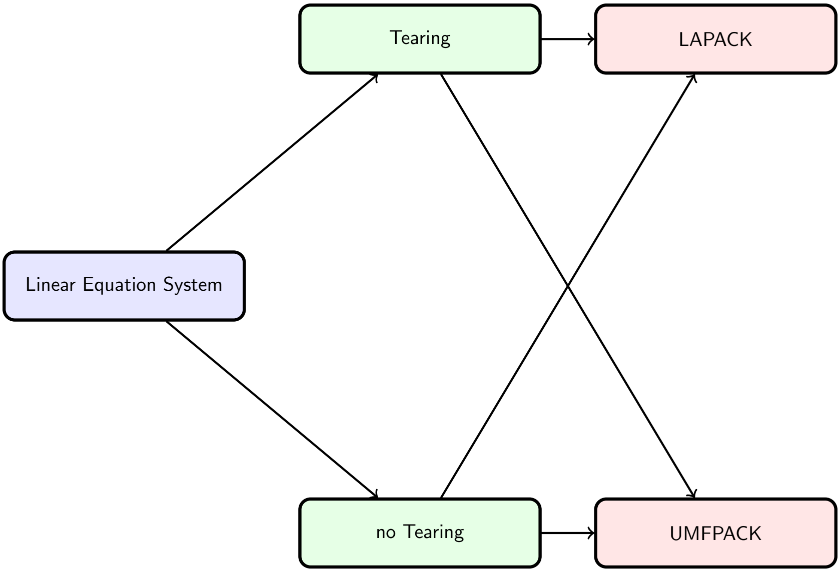
- LAPACK
  - Standard linear solver package.
  - Well known library for solving linear systems.
  - What else to say about it?
- Total pivot
  - Self written linear solver for full control.
  - Abilities to solve under-determined systems.

Sparse Solvers

- UMFPACK
  - Sparse solver suite.
  - Direct sparse linear solver.
  - Well tested and heavily used(e.g. Matlab).
- Lis
  - Iterative linear solvers suite.
  - Iterative solvers work only for well-posed problems.
  - This libraries works better for a really big N.

Benchmarks

Overview in OpenModelica



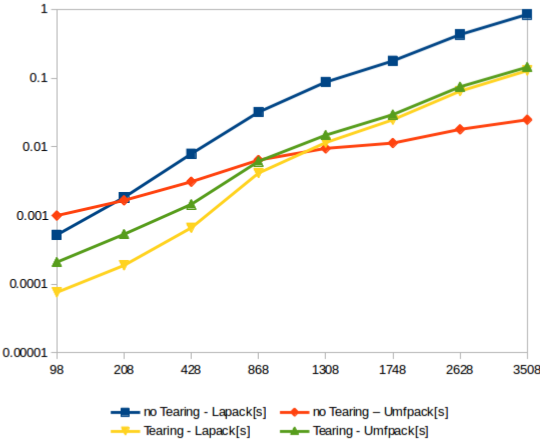
FH Bielefeld
University of
Applied Sciences

NPendlum

- Break-even point around $N = 1500$ and sparsity 0.17.
- After Tearing the linear system is 100% dense.

NPendlum Statistics

N	Size linear system Tearing	noTearing
10	10	98
20	20	208
40	40	428
80	80	868
120	120	1308
160	160	1748
240	240	2628
320	320	3508

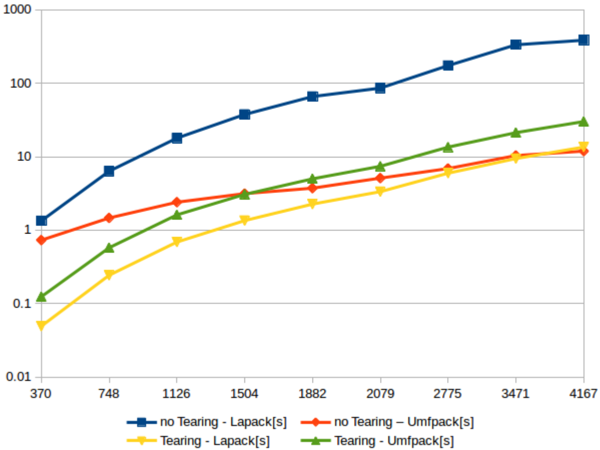


EngineV6 -> EngineV72

- Break-even point beyond N = 4000.
- After Tearing the linear system is still sparse.

EngineVN Statistics

N	Size linear system Tearing	noTearing
6	31	370
12	61	748
18	91	1126
24	121	1504
30	151	1882
36	181	2079
48	241	2775
60	301	3471
72	361	4167



MSL

- Not really given, since the largest linear system are around 400.

other Libraries

- ???

Summary

FH Bielefeld
University of
Applied Sciences

- Who can tell me which Libraries should be tested?
- Happy testing with your models!
- OpenModelica achieved a lot improvements in the last year on the solutions of linear equation systems.
- For MSL model Tearing with LAPACK superiors UMFPACK.

- Further work:
  - Combine UMFPACK with an integrator for bigger models.

Summary

- Who can tell me which Libraries should be tested?
- Happy testing with your models!
- OpenModelica achieved a lot improvements in the last year on the solutions of linear equation systems.
- For MSL model Tearing with LAPACK superiors UMFPACK.

- Further work:
  - Combine UMFPACK with an integrator for bigger models.

So long, and Thanks for All the infrastructure, help and the fish ;)

- Who can tell me which Libraries should be tested?
- Happy testing with your models!
- OpenModelica achieved a lot improvements in the last year on the solutions of linear equation systems.
- For MSL model Tearing with LAPACK superiors UMFPACK.
- Further work:
  - Combine UMFPACK with an integrator for bigger models.