Some implementation issues on the GKO algorithm

Federico Poloni[1,2]
From discussions with: A. Aricò, D. Bini, V. Olshevsky

[1]Dipartimento di Matematica
Università di Pisa

[2]Scuola Normale Superiore, Pisa

Cortona, 17 September 2008

Introduction

Our problem Solution of a matrix equation (NARE) with Cauchy-like matrices

(but I am not talking about this)

- Matrix iterations working on Cauchy-like matrices
- Led us to investigate on the existing algorithms – GKO
- Some (small) results that could be interesting also outside our problem

Cauchy-like matrices

Definition

C is Cauchy-like if there are $D_x = \operatorname{diag}(x)$, $D(y) = \operatorname{diag}(y)$ such that

$$D_x C - C D_y = G \cdot B = \text{▯} \cdot \text{▭} \qquad (\text{rank } r \ll n)$$

If $x_i \neq y_j$, then C_{ij} can be recovered from the generators:

$$C_{ij} = \frac{G(i,:) \cdot B(:,j)}{x_i - y_j}$$

If this is not always possible, C is partially reconstructible

Notable example: $(r = 2)$ from Toeplitz matrices, after a Fourier change of base

GKO: the idea

GKO algorithm [Gohberg–Kailath–Olshevsky, '95]

Theorem

The Schur complement of a Cauchy-like matrix is Cauchy-like. Its generators are a rank-1 update of $G(2{:}n,:)$ and $B(:,2{:}n)$.

Gaussian elimination working on G and B only, reconstructing elements when needed

GKO step by step

$$\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \qquad \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \quad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}$$

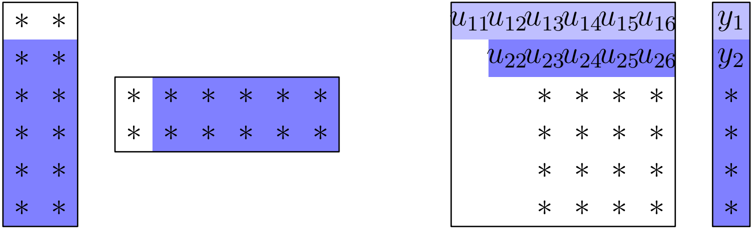
- reconstruct first column of L and use it to solve $Ly = b$ incrementally
- reconstruct first row of U and store it
- update the generators G and B

GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ * \\ * \\ * \\ * \\ * \end{bmatrix}
$$

- reconstruct first column of L and use it to solve $Ly = b$ incrementally
- reconstruct first row of U and store it
- update the generators G and B

GKO step by step



- reconstruct first column of L and use it to solve $Ly = b$ incrementally
- reconstruct first row of U and store it
- update the generators G and B

GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & * & * & * \\ & & & * & * & * \\ & & & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ * \\ * \\ * \end{bmatrix}
$$

- reconstruct first column of L and use it to solve $Ly = b$ incrementally
- reconstruct first row of U and store it
- update the generators G and B

GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & u_{44} & u_{45} & u_{46} \\ & & & & * & * \\ & & & & * & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ * \\ * \end{bmatrix}
$$

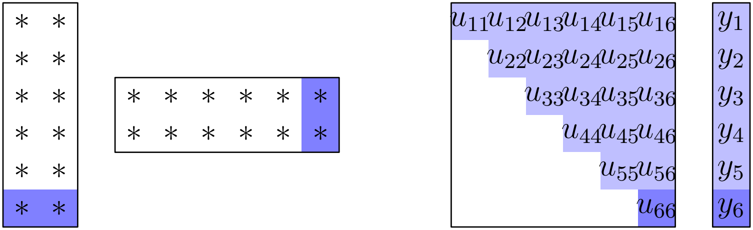
- reconstruct first column of L and use it to solve $Ly = b$ incrementally
- reconstruct first row of U and store it
- update the generators G and B

GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & u_{44} & u_{45} & u_{46} \\ & & & & u_{55} & u_{56} \\ & & & & & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ * \end{bmatrix}
$$

- reconstruct first column of L and use it to solve $Ly = b$ incrementally
- reconstruct first row of U and store it
- update the generators G and B

GKO step by step



- reconstruct first column of L and use it to solve $Ly = b$ incrementally
- reconstruct first row of U and store it
- update the generators G and B

and finally...

GKO step by step

$$\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \qquad \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & u_{44} & u_{45} & u_{46} \\ & & & & u_{55} & u_{56} \\ & & & & & u_{66} \end{bmatrix} \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}$$

- reconstruct first column of L and use it to solve $Ly = b$ incrementally
- reconstruct first row of U and store it
- update the generators G and B

and finally. . .

- solve $Ux = y$ by back-substitution

GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & u_{44} & u_{45} & u_{46} \\ & & & & u_{55} & u_{56} \\ & & & & & u_{66} \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}
$$

Problem We need to store U

- Why $O(n^2)$ temporaries for $O(n)$ input and output size?
- The maximum size that fits in memory is reduced
- Slower memory access (cache misses matter)

A solution: the extended matrix

Idea: first in [Kailath–Chun, '94], fully exploited by [Rodriguez, '06]
Matlab code [Aricò–Rodriguez]

$x = C^{-1}b$ is the Schur complement of the first block in

$$\begin{bmatrix} C & b \\ -I & 0 \end{bmatrix}$$

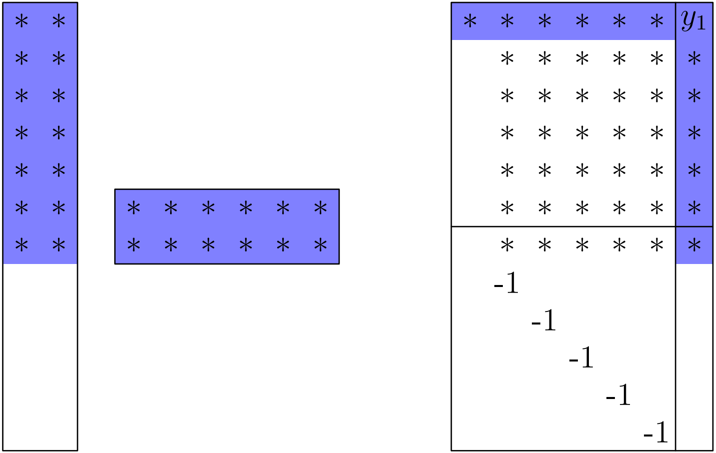
(b may be either $n \times 1$ or $n \times s$, multiple right-hand side)

- n steps of GKO on the extended matrix
- mixed Gaussian elimination: 1st column=GKO, 2nd column=traditional
- $-I$ is partially reconstructible Cauchy-like wrt $\operatorname{diag}(y), \operatorname{diag}(y)$
- you need not store the matrix U

Extended matrix GKO step by step

$$
\begin{bmatrix}
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
 & \\
 & \\
 & \\
 & \\
 & \\
 &
\end{bmatrix}
\qquad
\begin{bmatrix}
* & * & * & * & * & * \\
* & * & * & * & * & *
\end{bmatrix}
\qquad
\left[\begin{array}{cccccc|c}
* & * & * & * & * & * & b_1 \\
* & * & * & * & * & * & b_2 \\
* & * & * & * & * & * & b_3 \\
* & * & * & * & * & * & b_4 \\
* & * & * & * & * & * & b_5 \\
* & * & * & * & * & * & b_6 \\
\hline
-1 & & & & & & \\
 & -1 & & & & & \\
 & & -1 & & & & \\
 & & & -1 & & & \\
 & & & & -1 & & \\
 & & & & & -1 &
\end{array}\right]
$$

Extended matrix GKO step by step



Extended matrix GKO step by step

$$
\left[\begin{array}{cc}
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
 & \\
 & \\
 & \\
 &
\end{array}\right]
\qquad
\left[\begin{array}{cccccc}
* & * & * & * & * & * \\
* & * & * & * & * & *
\end{array}\right]
\qquad
\left[\begin{array}{cccccc|c}
* & * & * & * & * & * & y_1 \\
 & * & * & * & * & * & y_2 \\
 & & * & * & * & * & * \\
 & & * & * & * & * & * \\
 & & * & * & * & * & * \\
 & & * & * & * & * & * \\
\hline
 & & * & * & * & * & * \\
 & & * & * & * & * & * \\
 & & -1 & & & & \\
 & & & -1 & & & \\
 & & & & -1 & & \\
 & & & & & -1 &
\end{array}\right]
$$

Extended matrix GKO step by step

$$
\left[\begin{array}{cc}
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
 & \\
 & \\
 &
\end{array}\right]
\left[\begin{array}{cccccc}
* & * & * & * & * & * \\
* & * & * & * & * & *
\end{array}\right]
\qquad
\left[\begin{array}{cccccc|c}
* & * & * & * & * & * & y_1 \\
 & * & * & * & * & * & y_2 \\
 & & * & * & * & * & y_3 \\
 & & & * & * & * & * \\
 & & & * & * & * & * \\
 & & & * & * & * & * \\
\hline
 & & & * & * & * & * \\
 & & & * & * & * & * \\
 & & & * & * & * & * \\
 & & & -1 & & & \\
 & & & & -1 & & \\
 & & & & & -1 &
\end{array}\right]
$$

Extended matrix GKO step by step

$$
\begin{array}{|cc|}
\hline
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
 & \\
 & \\
\hline
\end{array}
\quad
\begin{array}{|cccccc|}
\hline
* & * & * & * & * & * \\
* & * & * & * & * & * \\
\hline
\end{array}
\qquad
\begin{array}{|cccccc|c|}
\hline
* & * & * & * & * & * & y_1 \\
 & * & * & * & * & * & y_2 \\
 & & * & * & * & * & y_3 \\
 & & & * & * & * & y_4 \\
 & & & & * & * & * \\
 & & & & * & * & * \\
\hline
 & & & & * & * & * \\
 & & & & * & * & * \\
 & & & & * & * & * \\
 & & & & * & * & * \\
 & & & & -1 & & \\
 & & & & & -1 & \\
\hline
\end{array}
$$

Extended matrix GKO step by step

$$
\left[\begin{array}{cc}
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
 &
\end{array}\right]
\quad
\left[\begin{array}{cccccc}
* & * & * & * & * & * \\
* & * & * & * & * & *
\end{array}\right]
\quad
\left[\begin{array}{cccccc|c}
* & * & * & * & * & * & y_1 \\
 & * & * & * & * & * & y_2 \\
 & & * & * & * & * & y_3 \\
 & & & * & * & * & y_4 \\
 & & & & * & * & y_5 \\
 & & & & & * & * \\
\hline
 & & & & & * & * \\
 & & & & & * & * \\
 & & & & & * & * \\
 & & & & & * & * \\
 & & & & & * & * \\
 & & & & & -1 &
\end{array}\right]
$$

Extended matrix GKO step by step

$$
\left[\begin{array}{cc}
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & *
\end{array}\right]
\quad
\left[\begin{array}{cccccc}
* & * & * & * & * & * \\
* & * & * & * & * & *
\end{array}\right]
\quad
\left[\begin{array}{cccccc|c}
* & * & * & * & * & * & y_1 \\
 & * & * & * & * & * & y_2 \\
 & & * & * & * & * & y_3 \\
 & & & * & * & * & y_4 \\
 & & & & * & * & y_5 \\
 & & & & & * & y_6 \\
\hline
 & & & & & & x_1 \\
 & & & & & & x_2 \\
 & & & & & & x_3 \\
 & & & & & & x_4 \\
 & & & & & & x_5 \\
 & & & & & & x_6
\end{array}\right]
$$

Extended matrix GKO step by step

$$
\begin{bmatrix}
* & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & *
\end{bmatrix}
\qquad
\begin{bmatrix}
* & * & * & * & * & * \\
* & * & * & * & * & *
\end{bmatrix}
\qquad
\left[\begin{array}{cccccc|c}
* & * & * & * & * & * & y_1 \\
 & * & * & * & * & * & y_2 \\
 & & * & * & * & * & y_3 \\
 & & & * & * & * & y_4 \\
 & & & & * & * & y_5 \\
 & & & & & * & y_6 \\
\hline
 & & & & & & x_1 \\
 & & & & & & x_2 \\
 & & & & & & x_3 \\
 & & & & & & x_4 \\
 & & & & & & x_5 \\
 & & & & & & x_6
\end{array}\right]
$$

Extended matrix GKO

Notice:

- In the $-I$ block, we divide by $y_i - y_j$ (with $j > i$):
  y must be injective (true in most applications)
- The $-I$ block diagonal is not reconstructible
  Luckily whenever we need an element, it is -1

Cost: $6rn^2$ instead of $4rn^2$ flops of original GKO
($+2n^2s$ for back-substitutions with a $n \times s$ right-hand side)

but only $2n$ buffer space is needed
In practice, faster for large values of n

Another solution: the back-and-forth method

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ 0 & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ 0 & 0 & u_{33} & u_{34} & u_{35} & u_{36} \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ * \\ * \\ * \end{bmatrix}
$$

At each GKO step, we discard the top row of G and column of B: $G(k,:)$, $B(:,k)$

In practice, they stay in memory (no unnecessary allocations!)

Idea: can we use these to undo one GKO step?

Another solution: the back-and-forth method

$$\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \qquad \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ 0 & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ 0 & 0 & u_{33} & u_{34} & u_{35} & u_{36} \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \end{bmatrix} \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ * \\ * \\ * \end{bmatrix}$$

- u_{kk} (pivot) can be recovered: $u_{kk} = \frac{G(k,:)B(:,k)}{x_k - y_k}$
- So can the rest of the u-row, using the value of B *after* step k:

$$u_{k\ell} = \frac{G(k,:)B_{before}(:,\ell)}{x_k - y_\ell} = \frac{G(k,:)B_{after}(:,\ell)}{y_k - y_\ell}$$

- Using u-row and B_{after}, we can undo the update to get B_{before}

Back-and-forth GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix}
$$

- GKO as usual
- Keep in memory the old parts of G and B
- Do not keep the old u_{ij}'s

Back-and-forth GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ * \\ * \\ * \\ * \\ * \end{bmatrix}
$$

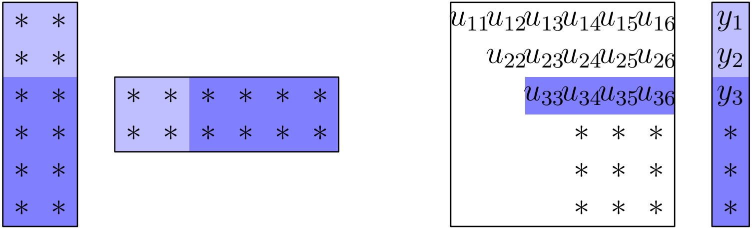
- GKO as usual
- Keep in memory the old parts of G and B
- Do not keep the old u_{ij}'s

Back-and-forth GKO step by step

$$
\begin{array}{|cc|}\hline
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\ \hline
\end{array}
\quad
\begin{array}{|cccccc|}\hline
* & * & * & * & * & * \\
* & * & * & * & * & * \\ \hline
\end{array}
\qquad
\begin{array}{|cccccc|}\hline
u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\
 & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\
 & & * & * & * & * \\
 & & * & * & * & * \\
 & & * & * & * & * \\
 & & * & * & * & * \\ \hline
\end{array}
\quad
\begin{array}{|c|}\hline
y_1 \\
y_2 \\
* \\
* \\
* \\
* \\ \hline
\end{array}
$$

- GKO as usual
- Keep in memory the old parts of G and B
- Do not keep the old u_{ij}'s

Back-and-forth GKO step by step



- GKO as usual
- Keep in memory the old parts of G and B
- Do not keep the old u_{ij}'s

Back-and-forth GKO step by step

$$\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \qquad \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & u_{44} & u_{45} & u_{46} \\ & & & & * & * \\ & & & & * & * \end{bmatrix} \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ * \\ * \end{bmatrix}$$

- GKO as usual
- Keep in memory the old parts of G and B
- Do not keep the old u_{ij}'s

Back-and-forth GKO step by step

$$\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \qquad \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & u_{44} & u_{45} & u_{46} \\ & & & & u_{55} & u_{56} \\ & & & & & * \end{bmatrix} \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ * \end{bmatrix}$$

- GKO as usual
- Keep in memory the old parts of G and B
- Do not keep the old u_{ij}'s

Back-and-forth GKO step by step

$$\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \qquad \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & u_{44} & u_{45} & u_{46} \\ & & & & u_{55} & u_{56} \\ & & & & & u_{66} \end{bmatrix} \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix}$$

- GKO as usual
- Keep in memory the old parts of G and B
- Do not keep the old u_{ij}'s

Back-and-forth GKO step by step

$$\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \qquad \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & u_{44} & u_{45} & u_{46} \\ & & & & u_{55} & u_{56} \\ & & & & & u_{66} \end{bmatrix} \quad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ x_6 \end{bmatrix}$$

For $k = n$ to 1:

- Reconstruct the kth row of u
- Use it to solve the kth equation of $Ux = y$ by back-substitution
- “downdate” B to its old value at step k of GKO

Back-and-forth GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & & u_{44} & u_{45} & u_{46} \\ & & & & u_{55} & u_{56} \\ & & & & * & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ x_5 \\ x_6 \end{bmatrix}
$$

For $k = n$ to 1:

- Reconstruct the kth row of u
- Use it to solve the kth equation of $Ux = y$ by back-substitution
- "downdate" B to its old value at step k of GKO

Back-and-forth GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix}
u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\
 & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\
 & & u_{33} & u_{34} & u_{35} & u_{36} \\
 & & & u_{44} & u_{45} & u_{46} \\
 & & & * & * & * \\
 & & & * & * & *
\end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}
$$

For $k = n$ to 1:

- Reconstruct the kth row of u
- Use it to solve the kth equation of $Ux = y$ by back-substitution
- "downdate" B to its old value at step k of GKO

Back-and-forth GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & u_{33} & u_{34} & u_{35} & u_{36} \\ & & * & * & * & * \\ & & * & * & * & * \\ & & * & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ y_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}
$$

For $k = n$ to 1:

- Reconstruct the kth row of u
- Use it to solve the kth equation of $Ux = y$ by back-substitution
- "downdate" B to its old value at step k of GKO

Back-and-forth GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \\ & * & * & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}
$$

For $k = n$ to 1:

- Reconstruct the kth row of u
- Use it to solve the kth equation of $Ux = y$ by back-substitution
- "downdate" B to its old value at step k of GKO

Back-and-forth GKO step by step

$$\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix} \quad \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \qquad \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

For $k = n$ to 1:

- Reconstruct the kth row of u
- Use it to solve the kth equation of $Ux = y$ by back-substitution
- "downdate" B to its old value at step k of GKO

Back-and-forth GKO step by step

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\quad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}
$$

For $k = n$ to 1:

- Reconstruct the kth row of u
- Use it to solve the kth equation of $Ux = y$ by back-substitution
- "downdate" B to its old value at step k of GKO

Back-and-forth GKO: considerations

- "downdate" of G is not needed
- cost: $6rn^2$: same as Extended Matrix
- memory: input size $+ 2n$ temps: same as Extended Matrix
- both require $y_i \neq y_j$ for $i \neq j$
- similar numerical behaviour

Are they the same algorithm? No:

Key questions: where is L in Extended Matrix? Where is U?

Extended Matrix – where are L and U?

$$
\left[\begin{array}{cc}
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
* & * \\
 & \\
 & \\
 &
\end{array}\right]
\quad
\left[\begin{array}{cccccc}
* & * & * & * & * & * \\
* & * & * & * & * & *
\end{array}\right]
\qquad\qquad
\left[\begin{array}{cccccc|c}
* & * & * & * & * & * & y_1 \\
 & * & * & * & * & * & y_2 \\
 & & * & * & * & * & y_3 \\
 & & & u_{44} & * & * & y_4 \\
 & & & * & * & * & * \\
 & & & * & * & * & * \\
\hline
 & & & v_{14} & * & * & * \\
 & & & v_{24} & * & * & * \\
 & & & v_{34} & * & * & * \\
 & & & \text{-}1 & & & \\
 & & & & \text{-}1 & & \\
 & & & & & \text{-}1 &
\end{array}\right]
$$

L: upper part as in GKO

U : $\quad (U^{-1})_{ij} = -\dfrac{v_{ij}}{u_{jj}}$

EM vs. BF: numerical experiments

Forward errors

n	EM	BF
10	1.6E-12	1.6E-12
100	1.4E-05	1.4E-05
500	8.0E-01	8.0E-01

ill-conditioned Cauchy-like

nodes$=1+0.3(i-j)$

n	EM	BF
10	7.9E-11	6.9E-11
100	2.0E-07	1.0E-07
500	1.3E-07	1.2E-07

Gaussian Toeplitz

$a = 0.9$

Caveat: not to be taken too seriously:

- Implementation matters, small optimizations = huge differences
- Processor, cache size, cache efficiency issues

EM vs. BF: numerical experiments

CPU time

n	EM	BF	plain
100	1.3E-03	1.2E-03	8.5E-04
1000	1.1E-01	9.7E-02	8.6E-02
3000	1.03E+00	8.6E-01	1.7E+00
10000	1.3E+01	1.0E+01	3.5E+01

Caveat: not to be taken too seriously:

- Implementation matters, small optimizations = huge differences
- Processor, cache size, cache efficiency issues

EM vs. BF: other factors

- BF: *a posteriori* error estimate: did we reconstruct the original generators properly?
  LU stability + generator growth
- BF: preview of the solution: after part 1, the entries of x arrive one at each step.
  In Toeplitz computations, lower-sampled "preview"
- BF: the inner steps take less memory (vs. EM: every step takes $2nr$ memory)
  Should fit better into cache

Trummer-like matrices: definitions

Definition

A Trummer-like matrix is a Cauchy-like matrix in which the non-reconstructible entries are the diagonal ones, i.e.

$$D_x T - T D_x = G \cdot B = \square \cdot \square \qquad (\text{rank } r \ll n)$$

Appear in many contests:

- Trummer's problem [Gerasoulis et al., 88]
- Toeplitz computations, e.g. [Kailath–Olshevsky '97]
- Integral equations, e.g. the matrix equation we were solving

Displacement rank and algorithms

How to store them?

- Store generators G, B as every Cauchy-like matrix, and
- Store the diagonal separately

Structure is preserved

If $\mathrm{TRk}(T) := \mathrm{Rk}(D_x T - T D_x)$

- $\mathrm{TRk}(T + S) = \mathrm{TRk}(T) + \mathrm{TRk}(S)$
- $\mathrm{TRk}(TS) = \mathrm{TRk}(T) + \mathrm{TRk}(S)$
- $\mathrm{TRk}(T^{-1}) = \mathrm{TRk}(T)$

Our goal: space-efficient fast Trummer-like matrix computations:
$T \cdot v$, $T \cdot S$, $T^{-1} \cdot v$, $T^{-1} \cdot S$

Matrix-vector operations

Matrix–vector product: easy! Recover T from its generators one row at a time, apply traditional M-v algorithm

Linear system solving $T^{-1}v$ idea in [Kailath–Olshevsky, '97]: Traditional GKO + store and update diagonal elements separately

$$
\begin{bmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{bmatrix}
\qquad
\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix}
\qquad\qquad
\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} & u_{15} & u_{16} \\ & u_{22} & u_{23} & u_{24} & u_{25} & u_{26} \\ & & * & * & * & * \\ & & * & * & * & * \\ & & * & * & * & * \\ & & * & * & * & * \end{bmatrix}
\quad
\begin{bmatrix} y_1 \\ y_2 \\ * \\ * \\ * \\ * \end{bmatrix}
$$

Matrix–matrix operations

Let $\nabla T := D_x T - T D_x$

Matrix–matrix product $T \cdot S$

- generators: $\nabla(TS) = T\nabla(S) + \nabla(T)S$
- diagonal: recover and multiply

Inverse T^{-1} (and product $T^{-1} \cdot S$)

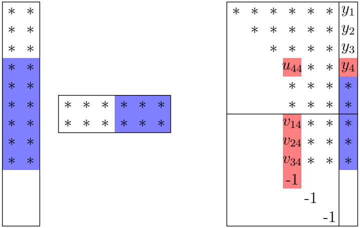
- generators: $\nabla(T^{-1}) = -T^{-1}\nabla(T)T^{-1}$
- diagonal: ??

No obvious algorithm to get $\mathrm{diag}(T^{-1})$

How to get diag(T^{-1})?

Entries of U^{-1}: explicitly available with the Extended Matrix version of GKO

*k*th step: *k*th column of U^{-1}



Entries of L^{-1}: repeat on T^T: the LU factors of T^T are $U^T L^T$ (up to diagonal scaling with the pivots)

*k*th step: *k*th row of L^{-T}

We get the right entries at the right time to compute the diagonal:

$$U^{-1}L^{-1} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ & 2 & 3 & 4 \\ & & 3 & 4 \\ & & & 4 \end{bmatrix} \begin{bmatrix} 1 & & & \\ 2 & 2 & & \\ 3 & 3 & 3 & \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

Implementing GKO+invdiag

Not exactly GKO twice: many computations are the same

$$\text{Schur\_compl}(T^T) = [\text{Schur\_compl}(T)]^T$$

- GKO is born symmetric: given generators of T, compute generators of Schur_compl(T) — $4rn^2$ ops
- EM version: some extra work on G — $6rn^2$ ops
- Now we restore symmetry by doing the same on B — $8rn^2$ ops

In $(12r+3)n^2$ ops we can build a full set of generators for T^{-1}:

- Solve $G' = T^{-1}G$
- Solve $B' = T^{-T}B$
- Compute $\text{diag}(T^{-1})$ with the above algorithm

Some numerical results

GKO+invdiag vs. a simpler strategy: choose v,

$$T^{-1}v = \mathrm{diag}(T^{-1})v + \left[T^{-1} - \mathrm{diag}(T^{-1})\right] v$$

- $T^{-1}v$ computed with GKO
- $\left[T^{-1} - \mathrm{diag}(T^{-1})\right] v$ computed easily from the generators
- solve for $\mathrm{diag}(T^{-1})v$

This strategy loses accuracy because of cancellation errors:

n	GKO+invdiag	solve for $\mathrm{diag}(T^{-1})v$
10	4.4E-16	5.8E-15
100	2.6E-14	1.4E-11
500	1.4E-12	3.9E-10
10	4.2E-15	1.2E-08
50	9.0E-08	1.6E-02

(random-generated Trummer-like M-matrices, diag+rank-1 matrices)

To sum up

- New $O(n)$-storage GKO version (Back-and-forth)
  Competitive with EM, some nice $+$'s
- GKO+invdiag to get $\text{diag}(T^{-1})$ for a Trummer-like T
  Allows fast Trummer-like matrix computations

  Also works when $\text{diag}(T)$ is reconstructible but ill-conditioned

To sum up

- New $O(n)$-storage GKO version (Back-and-forth)
  Competitive with EM, some nice $+$'s
- GKO+invdiag to get $\text{diag}(T^{-1})$ for a Trummer-like T
  Allows fast Trummer-like matrix computations

  Also works when $\text{diag}(T)$ is reconstructible but ill-conditioned

Thank you for your attention