Robust Solution of Triangular Linear Systems *

Carl Christian Kjelgaard Mikkelsen, Lars Karlsson

May 30, 2017

Abstract

Let $\boldsymbol{T} \in \mathbb{C}^{n \times n}$ be a non-singular upper triangular matrix and let $\boldsymbol{b} \in \mathbb{C}^n$. We show how to compute a scaling $\alpha \in (0, 1]$ and $\boldsymbol{x} \in \mathbb{C}^n$, such that not only is $\boldsymbol{T}\boldsymbol{x} = \alpha \boldsymbol{b}$, but all intermediate results are bounded by a threshold Ω. This problem is relevant when estimating condition numbers or computing eigenvectors. Our scalar algorithm is similar to `xLATRS` from LAPACK. We explain why it is difficult to parallelize these algorithms. We then develop a robust blocked algorithm which can be executed in parallel using a run-time systems such as StarPU.

Contents

1 Introduction

Let $T \in \mathbb{C}^{n \times n}$ be a non-singular upper triangular matrix and let $\boldsymbol{b} \in \mathbb{C}^{\boldsymbol{n}}$. In this note we study the problem of computing a scalar $\alpha \in (0, 1]$ such that the solution $\boldsymbol{x} \in \mathbb{C}^n$ of the scaled linear system

$$\boldsymbol{T}\boldsymbol{x} = \alpha \boldsymbol{b}$$

*NLAFET Working Note 9. Report UMINF 17.09, Dept. Computing Science, Umeå University, SE 901 87 Umeå, Sweden.

can be computed without exceeding a threshold Ω. Useful values of Ω are all less than the largest value which can be represented.

This problem is relevant in the context of estimating condition numbers [1] and computing eigenvectors [3]. In both cases it is necessary to solve triangular linear systems which are potentially very ill-conditioned. When computing eigenvectors we are solving systems of the form $(T - \lambda I)x = b$ where λ. If there are many nearby eigenvalues, then overflow is a distinct possibility, because $\boldsymbol{T} - \lambda \boldsymbol{I}$ has many small diagonal elements.

The algorithms developed by Edward Anderson have been implemented in `xLATRS` and the principles have been integrated into `xTREVC` which computes eigenvectors of triangular matrices. ScaLAPACK contains complex flavors `PCTREVC` and `PZTREVC` which call `PxLATTRS` to solve triangular linear systems robustly. These auxiliary routines have been derived from `xLATRS` using PBLAS. This significantly limits the parallel efficiency of `PxTREVC` when numerical scaling is required.

In this note we develop a scalar and a blocked algorithm for solving (1) with respect to $(\alpha, \boldsymbol{x})$. The blocked algorithm can be executed in parallel using a task bask run-time system such as StarPU [2]. The solve and update kernels are replaced with robust variants which operate on *augmented* vectors $\langle \alpha, \boldsymbol{x} \rangle$ rather than regular vectors.

The parallel overhead is increased marginally compared with a task based implementation of non-robust backward substitution. In the long run, this will allow us to remove the parallel bottleneck associated with the robust parallel computation of eigenvectors.

2 A scalar algorithm

In this section we derive a robust scalar algorithm for solving the scaled upper triangular linear system (1). Our algorithm is closely related to the algorithm implemented in `xLATRS` in `LAPACK`, [1]. Our primary contribution is to simplify and extend the analysis of the underlying problem.

An upper triangular linear system can be solved using scalar backward substitution, i.e., Alg. 1 `ScalarSolve`. It consists of a sequence of scalar divisions

$$x \leftarrow b/t, \quad t \neq 0 \tag{1}$$

and linear updates

$$z \leftarrow y - tx.$$

In Section 2.1 and Section 2.2 we show how to prevent overflow for each of these fundamental operations. We then derive a robust variant of scalar backward substitution in Section 2.4.

2.1 Robust scalar divisions

The scalar division (1) cannot exceed Ω, if $|b| \leq |t|\Omega$. In general, we seek a scaling α such that

$$|\alpha b| \leq |t|\Omega, \tag{2}$$

which implies that the scaled division

$$y \leftarrow \frac{(\alpha b)}{t} \tag{3}$$

cannot exceed Ω. The computation of a suitable scaling α is the subject of the following theorem.

Algorithm 1: $\boldsymbol{x} = \texttt{ScalarSolve}(\boldsymbol{T}, \boldsymbol{b})$

Data: A non-singular upper triangular matrix $\boldsymbol{T} \in \mathbb{C}^{n \times n}$ and a vector $\boldsymbol{b} \in \mathbb{C}^n$.
Result: The solution $\boldsymbol{x}$ of the linear system $\boldsymbol{T}\boldsymbol{x} = \boldsymbol{b}$.

```
x ← b;
for j = n, n − 1, . . . , 1 do
    x_j ← x_j / t_jj;
    for i = 1, 2, . . . , j − 1 do
        x_i ← x_i − t_ij x_j;
return x;
```

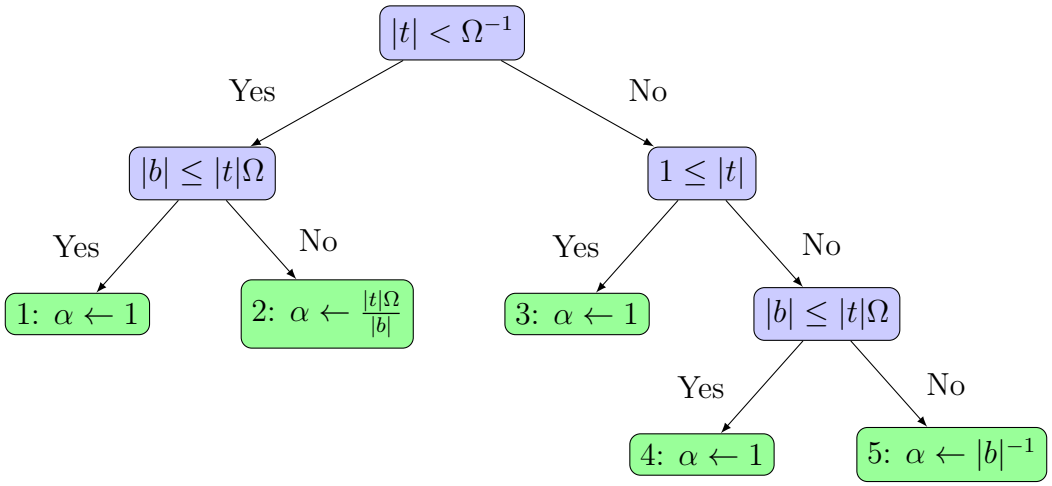

Figure 1: Computing a scaling factor α such that the division (3) cannot overflow.

Theorem 1. *Let $b \in \mathbb{C}$ and let $t \neq 0$ such that b/t is defined. If $|b| \le \Omega$, then the scaling factor α produced by the flowchart in Fig. 1 satisfies $\alpha \in (0, 1]$ and ensures that the result of the scaled division* (3) *satisfies $|y| \le \Omega$.*

Proof. There are five cases to consider, one for each leaf shown in Fig. 1. In each case, the proof consists of verifying that the central inequality (2) is satisfied for the given value of α.

1. We already have $|b| \le |t|\Omega$, so no scaling is necessary.
2. We have $|b| > |t|\Omega$, which implies $\alpha = \frac{|t|\Omega}{|b|} < 1$ and $|y| = \frac{\alpha|b|}{|t|} = \Omega$.
3. We have $1 \le |t|$ and $|b| \le \Omega$, which implies $|b| \le |t|\Omega$. Therefore, no scaling is necessary.
4. We already have $|b| \le |t|\Omega$, so no scaling is necessary.
5. We have $|t| \ge \Omega^{-1}$ and $|b| > |t|\Omega$, so

$$\alpha = |b|^{-1} < |t|^{-1}\Omega^{-1} \le 1.$$

Moreover,

$$|y| = \frac{\alpha|b|}{|t|} = \frac{1}{|t|} \le \Omega.$$

This completes the proof. □

Remark 1. *Thm. 1 requires $|b| \leq \Omega$. If we also have $|t| \leq \Omega$, then the right hand side and the left hand side of each inequality in Fig. 1 can be evaluated without exceeding Ω.*

Alg. 2 `ProtectDivision` summarizes how to safely compute a scaling factor which prevents overflow in the scaled division (3).

Algorithm 2: $\alpha = \texttt{ProtectDivision}(b, t)$

Data: Numbers b and t such that $|b| \leq \Omega$ and $t \neq 0$.
Result: A scaling $\alpha \in (0, 1]$ such that the scaled division (3) cannot overflow.

```
α ← 1;
if |t| < Ω^{-1} then
    if |b| > |t|Ω then
        α ← |t|Ω / |b|;
else
    if |t| < 1 then
        if |b| > |t|Ω then
            α ← |b|^{-1};
return α;
```

2.2 Robust linear updates

We consider more the general problem of computing a scaling $\xi \in (0, 1]$, such that the scaled linear transformation

$$\boldsymbol{Z} \leftarrow \xi \boldsymbol{Y} - \boldsymbol{T}(\xi \boldsymbol{X})$$

satisfies $\|\boldsymbol{Z}\|_\infty \leq \Omega$. We begin by stating a condition which ensures that no scaling is necessary.

Theorem 2. *Let $\boldsymbol{Y}, \boldsymbol{T}$ and $\boldsymbol{X}$ be matrices such that $\boldsymbol{Z} = \boldsymbol{Y} - \boldsymbol{T}\boldsymbol{X}$ is defined. If*

$$\|\boldsymbol{Y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\boldsymbol{X}\|_\infty \leq \Omega,$$

then overflow is impossible in the calculation of $\boldsymbol{Z}$ regardless of the order of the necessary arithmetic operations.

Proof. By the triangle inequality we have the elementary estimate

$$|z_{ij}| \leq \|\boldsymbol{Z}\|_\infty \leq \|\boldsymbol{Y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\boldsymbol{X}\|_\infty.$$

It follows that each component of the *final* result $\boldsymbol{Z}$ is dominated by Ω, i.e., $|z_{ij}| \leq \Omega$. There are many ways to evaluate the the expression

$$z_{ij} = y_{ij} - \sum_k t_{ik} x_{kj},$$

but each *intermediate* result can be written as

$$z_{ij}(\mu, \boldsymbol{\sigma}) = \mu y_{ij} - \sum_k \sigma_k t_{ik} x_{kj}, \quad \mu \in \{0, 1\}, \quad \sigma_k \in \{0, 1\},$$

where the choice of, say, $\sigma_k = 1$ reflects that the corresponding term $t_{ik}x_{kj}$ has been included in the calculation. In all cases, the triangular inequality implies

$$|z_{ij}(\mu, \boldsymbol{\sigma})| \leq |y_{ij}| + \sum_k |t_{ik}||x_{kj}| = |\boldsymbol{Y}|_{ij} + (|\boldsymbol{T}||\boldsymbol{X}|)_{ij} \leq$$
$$\||\boldsymbol{Y}|\|_\infty + \||\boldsymbol{T}||\boldsymbol{X}|\|_\infty \leq \|\boldsymbol{Y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\boldsymbol{X}\|_\infty \leq \Omega.$$

It follows, that all components of all intermediate results are bounded by Ω, regardless of the order of the necessary arithmetic operations. □

In general, we seek a scaling factor ζ such that the scaled linear transformation

$$\boldsymbol{Z} \leftarrow (\zeta \boldsymbol{Y}) - \boldsymbol{T}(\zeta \boldsymbol{X}) \tag{4}$$

can be computed without exceeding Ω. The computation of a suitable scaling ζ is the subject of the following theorem.

Theorem 3. *Let $\boldsymbol{Y}$, $\boldsymbol{T}$, and $\boldsymbol{X}$ be matrices, such that $\boldsymbol{Z} = \boldsymbol{Y} - \boldsymbol{T}\boldsymbol{X}$ is defined. If*

$$\|\boldsymbol{Y}\|_\infty \leq \Omega, \quad \|\boldsymbol{T}\|_\infty \leq \Omega,$$

then the scaling factor ζ produced by the flowchart in Fig. 2 satisfies $\zeta \in (0, 1]$ and prevents overflow in the scaled linear update (4).

Proof. There are four cases to consider, one for each leaf shown in Fig. 2. In each case, the proof consists of verifying that

$$\|\zeta \boldsymbol{Y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\zeta \boldsymbol{X}\|_\infty \leq \Omega \tag{5}$$

for the given value of ζ. By Thm. 2 this ensures that the scaled linear update (4) cannot overflow

1. In this case,
$$\|\boldsymbol{T}\|_\infty \|\boldsymbol{X}\|_\infty \leq \Omega - \|\boldsymbol{Y}\|_\infty$$
and so the central inequality (5) is satisfied for all $|\zeta| \leq 1$.

2. In this case, we have $\zeta = \frac{1}{2}$. We then use
$$\|\boldsymbol{Y}\|_\infty \leq \Omega, \quad \|\boldsymbol{T}\|_\infty \leq \Omega, \quad \|\boldsymbol{X}\|_\infty \leq 1,$$
to estimate
$$\|\boldsymbol{Z}\|_\infty \leq \|\zeta \boldsymbol{Y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\zeta \boldsymbol{X}\|_\infty = \frac{1}{2}\|\boldsymbol{Y}\|_\infty + \frac{1}{2}\|\boldsymbol{T}\|_\infty \|\boldsymbol{X}\|_\infty \leq \Omega.$$

3. In this case,
$$\|\boldsymbol{T}\|_\infty \leq \frac{\Omega - \|\boldsymbol{Y}\|_\infty}{\|\boldsymbol{X}\|_\infty}$$
and so the central inequality (5) is satisfied for all $|\zeta| \leq 1$.

4. In this case $\|\boldsymbol{X}\|_\infty > 1$, so $\zeta = \frac{1}{2\|\boldsymbol{X}\|_\infty} < \frac{1}{2}$. Moreover,
$$\|\zeta \boldsymbol{Y}\|_\infty + \|\boldsymbol{T}\|_\infty \|\zeta \boldsymbol{X}\|_\infty < \frac{1}{2}\left(\|\boldsymbol{Y}\|_\infty + \|\boldsymbol{T}\|_\infty\right) \leq \Omega.$$

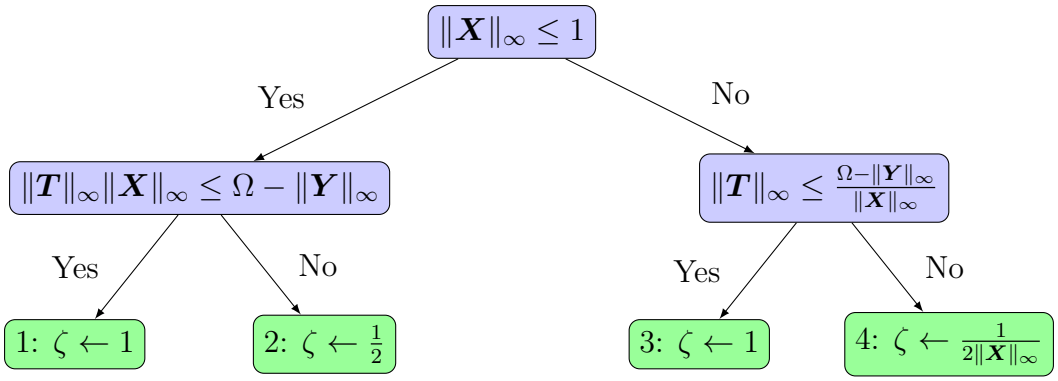

Figure 2: Computing a scaling factor ζ such that the linear update (4) cannot overflow.

This completes the proof. □

Remark 2. *Thm. 3 requires* $\|\boldsymbol{Y}\|_\infty \leq \Omega$ *and* $\|\boldsymbol{T}\|_\infty \leq \Omega$. *If we also have* $\|\boldsymbol{X}\|_\infty \leq \Omega$, *then the right hand side and the left hand side of each inequality in Fig. 2 can be evaluated without exceeding* Ω.

Alg. 3 `ProtectUpdate` summaries how to safely compute a scaling such that the scaled linear update (4) cannot overflow.

Algorithm 3: $\zeta = \texttt{ProtectUpdate}(y_{\text{norm}}, t_{\text{norm}}, x_{\text{norm}},)$

Data: Non-negative real numbers y_{norm}, t_{norm} and x_{norm} such that

$$\|\boldsymbol{Y}\|_\infty \leq y_{\text{norm}} \leq \Omega, \quad \|\boldsymbol{T}\|_\infty \leq t_{\text{norm}} \leq \Omega, \quad \|\boldsymbol{X}\|_\infty \leq x_{\text{norm}}.$$

Result: A scaling factor ζ such that

$$\zeta \left(y_{\text{norm}} + t_{\text{norm}} x_{\text{norm}}\right) \leq \Omega,$$

which implies that the scaled linear update (4) cannot exceed Ω.

```
ζ ← 1;
if x_norm ≤ 1 then
    if t_norm x_norm > Ω − b_norm then
        ζ ← 1/2;
else
    if t_norm > (Ω − b_norm)/x_norm then
        ζ ← 1/(2 x_norm);
return ζ;
```

2.3 Linear updates with linear growth

The following theorem is a weaker result which is useful in the context of solving linear systems.[1] It does not require the computation of $\|\boldsymbol{Y}\|_\infty$. We are interested in the situation where $\Omega' \ll \Omega$, i.e., when there is amble room for growth.

[1] This section will be moved and updated with additional information in the future.

Theorem 4. *Let ξ be the scaling factor computed using the simplified flowchart in Fig. 3. If $\|\boldsymbol{T}\|_\infty \le \Omega'$ and $\|\boldsymbol{Y}\|_\infty \le k\Omega'$, then $\boldsymbol{Z} = \xi\boldsymbol{Y} - \boldsymbol{T}(\xi\boldsymbol{X})$ satisfies $\|\boldsymbol{Z}\|_\infty \le (k+1)\Omega'$.*

Proof. The proof is straightforward and consists of verifying the conclusion in each of the three cases represented by the leaves in Fig. 3. In general, the triangle inequality implies

$$\|\boldsymbol{\xi}Z\|_\infty \le \xi\|\boldsymbol{Y}\|_\infty + \xi\|\boldsymbol{T}\|_\infty\|\boldsymbol{X}\|_\infty. \tag{6}$$

1. In this case, $\xi = 1$, $\|\boldsymbol{X}\|_\infty \le 1$ and (6) implies, that

$$\|\boldsymbol{Z}\|_\infty \le \|\boldsymbol{Y}\|_\infty + \|\boldsymbol{T}\|_\infty \le k\Omega' + \Omega' = (k+1)\Omega'.$$

2. In this case, $\xi = 1$, $\|\boldsymbol{T}\|_\infty\|\boldsymbol{X}\|_\infty \le \Omega'$ and (6) implies, that

$$\|\boldsymbol{Z}\|_\infty \le k\Omega' + \Omega' = (k+1)\Omega'.$$

3. In this case $\xi = \frac{1}{\|\boldsymbol{X}\|_\infty} < 1$ and

$$\|\boldsymbol{Z}\|_\infty \le \|\boldsymbol{Y}\|_\infty + \|\boldsymbol{T}\|_\infty \le (k+1)\Omega'.$$

This completes the proof. □

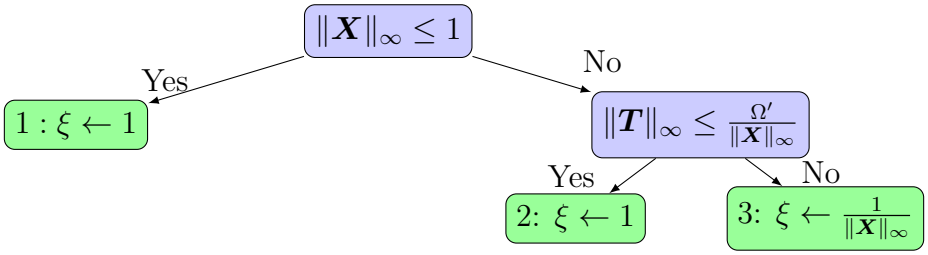

Figure 3: A simplified computation which allows for linear growth, see Thm. 4.

2.4 Robust scalar backward substitution

In Sections 2.1 and 2.2 we have shown how to protect a division and a linear update against overflow. It is straightforward to combine them into Alg. 4 `RobustScalarSolve`.

The reader should consider the assumptions placed on the input. Assumption (7) ensures that $\|\boldsymbol{T}_{1:j-1,j}\|_\infty \le \Omega$, so that $c_j = \|\boldsymbol{T}_{1:j-1,j}\|_\infty$ is a viable option. Assumption (8) ensures that $x_{\text{xmax}} \le \Omega$ initially. This critical property is preserved by virtue of the properties of Alg. 2 `ProtectDivision` and Alg. 3 `ProtectUpdate`.

The reader should also consider the problem of parallelizing `RobustScalarSolve`. Any scaling of $\boldsymbol{x}$ requires global communication as does the re-calculation of $x_{\max}$ at the end of each iteration. It is clear that alternatives should be explored.

Algorithm 4: $[\alpha, \boldsymbol{x}] = \texttt{RobustScalarSolve}(\boldsymbol{T}, \boldsymbol{b})$

Data: A non-singular upper triangular matrix $\boldsymbol{T} \in \mathbb{C}^{n\times n}$, and $\boldsymbol{b} \in \mathbb{C}^n$, subject to the following constraints

$$|t_{ij}| \leq \Omega \quad \text{for } i < j, \tag{7}$$

and

$$|b_j| \leq \Omega. \tag{8}$$

Result: A scaling $\alpha \in (0, 1]$ and the solution $\boldsymbol{x}$ of the scaled linear system

$$\boldsymbol{T}\boldsymbol{x} = \alpha\boldsymbol{b}$$

where α ensures that $\boldsymbol{x}$ can be computed without exceeding Ω.

Find c_j such that $\|T_{1:j-1,j}\|_\infty \leq c_j \leq \Omega$ for $j = 2, 3, \ldots, n$.;
$\alpha \leftarrow 1$, $\boldsymbol{x} \leftarrow \boldsymbol{b}$;
$x_{\max} \leftarrow \|\boldsymbol{x}\|_\infty$;
for $j \leftarrow n, n-1, \ldots, 1$ **do**
    $\beta = \texttt{ProtectDivision}(x_j, t_{jj})$;
    if $\beta \neq 1$ **then**
        $\boldsymbol{x} \leftarrow \beta\boldsymbol{x}$;
        $\alpha \leftarrow \beta\alpha$;
    $x_j \leftarrow x_j/t_{jj}$;
    if $j > 1$ **then**
        $\beta = \texttt{ProtectUpdate}(x_{\max}, c_j, |x_j|)$;
        if $\beta \neq 1$ **then**
            $\boldsymbol{x} \leftarrow \beta\boldsymbol{x}$;
            $\alpha \leftarrow \beta\alpha$;
        $\boldsymbol{x}_{1:j-1} \leftarrow \boldsymbol{x}_{1:j-1} - \boldsymbol{T}_{1:j-1,j}\boldsymbol{x}_j$;
        $x_{\max} \leftarrow \|\boldsymbol{x}_{1:j-1}\|_\infty$;
return $[\alpha, \boldsymbol{x}]$;

3 A blocked algorithm

Let $\boldsymbol{T} \in \mathbb{C}^{n\times n}$ be a non-singular upper triangular matrix, let $\boldsymbol{b} \in \mathbb{C}^n$ and partition the linear system $\boldsymbol{T}\boldsymbol{x} = \boldsymbol{b}$ conformally, i.e.,

$$\boldsymbol{T}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{T}_{11} & \boldsymbol{T}_{12} & \ldots & \boldsymbol{T}_{1N} \\ & \boldsymbol{T}_{22} & \ldots & \boldsymbol{T}_{2N} \\ & & \ddots & \vdots \\ & & & \boldsymbol{T}_{NN} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_N \end{bmatrix} = \boldsymbol{b}. \tag{9}$$

This system can be solved using blocked backward substitution as in Alg. 5 `BlockedSolve`.

In order to obtain a robust algorithm we must replace the two kernels

$$f(\boldsymbol{T}, \boldsymbol{b}) = \boldsymbol{T}^{-1}\boldsymbol{b}, \quad g(\boldsymbol{y}, \boldsymbol{T}, \boldsymbol{x}) = \boldsymbol{y} - \boldsymbol{T}\boldsymbol{x}$$

with robust variants. To this end, we will associate with each block $\boldsymbol{x}_i$ of the solution $\boldsymbol{x}$

Algorithm 5: $\boldsymbol{x} = \texttt{BlockedSolve}(\boldsymbol{T}, \boldsymbol{b})$

Data: A non-singular upper triangular matrix $\boldsymbol{T} \in \mathbb{C}^{n\times n}$ and $\boldsymbol{b} \in \mathbb{C}^n$ partitioned conformally as equation (9).
Result: The solution $\boldsymbol{x}$ of $\boldsymbol{T}\boldsymbol{x} = \boldsymbol{b}$.

```
x ← b;
for j = N, N − 1, ..., 1 do
    x_j ← f(T_jj, x_j);
    for i = 1, 2, ..., j − 1 do
        x_i ← g(x_i, T_ij, x_j);
return x;
```

a unique scaling factor $\alpha_i \in (0, 1]$ and all operations will be performed on such pairs. At this point, it is convenient to introduce the concept of an *augmented* vector.

Definition 1. *An augmented vector $\langle \alpha, \boldsymbol{x}\rangle$ consists of a scalar $\alpha \in (0, 1]$ and a vector $\boldsymbol{x} \in \mathbb{C}^n$ and represents the vector $\boldsymbol{y} = \alpha^{-1}\boldsymbol{x}$. Two augmented vectors $\langle \alpha, \boldsymbol{x}\rangle$ and $\langle \beta, \boldsymbol{y}\rangle$ are said be equivalent if they represent the same vector, i.e.,*

$$\langle \alpha, \boldsymbol{x}\rangle \sim \langle \beta, \boldsymbol{y}\rangle \Leftrightarrow \alpha^{-1}\boldsymbol{x} = \beta^{-1}\boldsymbol{y}.$$

Two augmented vectors $\langle \alpha, \boldsymbol{x}\rangle$ and $\langle \beta, \boldsymbol{y}\rangle$ are said to be consistently scaled if $\alpha = \beta$.

3.1 Robust block solve

Let $\boldsymbol{T} \in \mathbb{C}^{n\times n}$ be an non-singular upper triangular matrix such that $|t_{ij}| \leq \Omega$ for $i < j$ and let $\langle \beta, \boldsymbol{b}\rangle$ be an augmented vector for which $|b_j| \leq \Omega$. Then we can use `RobustScalarSolve` to compute an augmented vector $\langle \gamma, \boldsymbol{x}\rangle$ such that

$$\boldsymbol{T}\boldsymbol{x} = \gamma\boldsymbol{b},$$

where the scaling factor γ ensures that $\boldsymbol{x}$ can be computed without exceeding Ω. It follows that

$$\boldsymbol{T}\boldsymbol{x} = (\gamma\beta)\beta^{-1}\boldsymbol{b}$$

or equivalently

$$\boldsymbol{T}(\alpha^{-1}\boldsymbol{x}) = \beta^{-1}\boldsymbol{b}, \quad \alpha = \gamma\beta.$$

This procedure is formalized as Alg. 6.

3.2 Robust block update

Let $\langle \alpha, \boldsymbol{x}\rangle$ and $\langle \beta, \boldsymbol{y}\rangle$ be a pair of augmented vectors and let $\boldsymbol{T}$ be a matrix such that $\boldsymbol{z} = \boldsymbol{y} - \boldsymbol{T}\boldsymbol{x}$ is defined. In this section we show how to compute an augmented vector $\langle \nu, \boldsymbol{z}\rangle$ such that

$$\nu^{-1}\boldsymbol{z} = \beta^{-1}\boldsymbol{y} - \boldsymbol{T}(\alpha^{-1}\boldsymbol{x}).$$

Moreover, the scaling ν will ensure that $\boldsymbol{z}$ can be computed without overflow. There are two cases to consider, either $\alpha = \beta$ or $\alpha \neq \beta$.

Algorithm 6: $\langle \alpha, \boldsymbol{x} \rangle = \texttt{RobustBlockSolve}(\boldsymbol{T}, \langle \beta, \boldsymbol{b} \rangle)$

Data: A non-singular upper triangular matrix $\boldsymbol{T}$ and an augmented vector $\langle \beta, \boldsymbol{b} \rangle$ such that

$$|t_{ij}| \leq \Omega, \quad \text{for } i < j \quad \text{and} \quad |b_j| \leq \Omega.$$

Result: An augmented vector $\langle \alpha, \boldsymbol{x} \rangle$ such that

$$\boldsymbol{T}(\alpha^{-1}\boldsymbol{x}) = \beta^{-1}\boldsymbol{b}.$$

Moreover, the computation of $\boldsymbol{x}$ cannot exceed Ω.

1 $[\gamma, \boldsymbol{x}] \leftarrow \texttt{RobustScalarSolve}(\boldsymbol{T}, \boldsymbol{b})$;
2 $\alpha \leftarrow \gamma\beta$;
3 **return** $\langle \alpha, \boldsymbol{x} \rangle$;

If $\alpha = \beta$, then we can use Alg. 3 `ProtectUpdate` to compute a scaling ζ such that the scaled linear update

$$\boldsymbol{z} = \zeta\boldsymbol{y} - \boldsymbol{T}(\zeta\boldsymbol{x})$$

cannot exceed Ω. It follows immediately that

$$\nu^{-1}\boldsymbol{z} = \alpha^{-1}\boldsymbol{y} - \boldsymbol{T}(\alpha^{-1}\boldsymbol{x}),$$

provided that we choose $\nu = \alpha\zeta$.

If $\alpha \neq \beta$, then we replace $\langle \alpha, \boldsymbol{x} \rangle$ with $\langle \gamma, \boldsymbol{x}' \rangle$ and we replace $\langle \beta, \boldsymbol{y} \rangle$ with $\langle \gamma, \boldsymbol{y}' \rangle$ given by

$$\gamma = \min\{\alpha, \beta\}, \quad \boldsymbol{x}' = (\gamma/\alpha)\boldsymbol{x}, \quad \boldsymbol{y}' = (\gamma/\beta)\boldsymbol{y}. \tag{10}$$

It is straightforward to verify that

$$\langle \alpha, \boldsymbol{x} \rangle \sim \langle \gamma, \boldsymbol{x}' \rangle, \quad \langle \beta, \boldsymbol{y} \rangle \sim \langle \gamma, \boldsymbol{y}' \rangle.$$

Moreover, $\boldsymbol{x}'$ and $\boldsymbol{y}'$ can be computed without overflow, because $(\gamma/\alpha) \leq 1$ and $(\gamma/\beta) \leq 1$. Since $\langle \gamma, \boldsymbol{x}' \rangle$ and $\langle \gamma, \boldsymbol{y}' \rangle$ are consistently scaled, we can now use Alg. 3 `ProtectUpdate` to compute a scaling ζ such that the scaled linear update

$$\boldsymbol{z} = \zeta\boldsymbol{y}' - \boldsymbol{T}(\zeta\boldsymbol{x}')$$

cannot exceed Ω. From equation (10) it follows that

$$\boldsymbol{z} = \zeta(\gamma/\beta)\boldsymbol{y} - \boldsymbol{T}(\zeta(\gamma/\alpha)\boldsymbol{x})$$

or equivalently

$$\nu^{-1}\boldsymbol{z} = \beta^{-1}\boldsymbol{y} - \boldsymbol{T}(\alpha^{-1}\boldsymbol{x}),$$

provided that we choose $\nu = \zeta\gamma$ (as in the previous case).

The entire procedure is formalized as Alg. 7 `RobustBlockUpdate`.

3.3 Robust blocked backward substitution

Alg. 8 `RobustBlockedSolve` uses `RobustBlockSolve` and `RobustBlockUpdate` to solve an upper triangular system in a robust manner using augmented vectors.

Algorithm 7: $\langle \nu, \boldsymbol{z} \rangle = \texttt{RobustBlockUpdate}(\langle \beta, \boldsymbol{y} \rangle, \boldsymbol{T}, \langle \alpha, \boldsymbol{x} \rangle)$

Data: Augmented vectors $\langle \alpha, \boldsymbol{x} \rangle$, $\langle \beta, \boldsymbol{y} \rangle$ and a matrix $\boldsymbol{T}$ such that the linear transformation

$$\boldsymbol{z} = \boldsymbol{y} - \boldsymbol{T}\boldsymbol{x}$$

is defined and

$$\|\boldsymbol{y}\|_\infty \leq \Omega, \quad \|\boldsymbol{T}\|_\infty \leq \Omega, \quad \|\boldsymbol{x}\|_\infty \leq \Omega.$$

Result: An augmented vector $\langle \nu, \boldsymbol{z} \rangle$ such that

$$\nu^{-1}\boldsymbol{z} = \beta^{-1}\boldsymbol{y} - \boldsymbol{T}(\alpha^{-1}\boldsymbol{x})$$

and the scaling ν ensures that the computation of $\boldsymbol{z}$ cannot exceed Ω.

1 $\gamma = \min\{\alpha, \beta\}$;
2 $\boldsymbol{x} \leftarrow (\gamma/\alpha)\boldsymbol{x}$;
3 $\boldsymbol{y} \leftarrow (\gamma/\beta)\boldsymbol{y}$;
4 $\zeta \leftarrow \texttt{ProtectUpdate}(\|\boldsymbol{y}\|_\infty, \|\boldsymbol{T}\|_\infty, \|\boldsymbol{x}\|_\infty)$;
5 $\boldsymbol{z} \leftarrow \zeta\boldsymbol{y} - \boldsymbol{T}(\zeta\boldsymbol{x})$;
6 $\nu = \zeta\gamma$;
7 **return** $\langle \nu, \boldsymbol{z} \rangle$;

The correctness of the algorithm follows from the correctness of the two kernels. It is critical to examine the impact of the assumptions. Assumption (11) ensures that the diagonal blocks $\|\boldsymbol{T}_{jj}\|_\infty$ satisfy the requirements of Alg. 6 `RobustBlockedSolve`. Assumption (12) ensures that the super diagonal blocks $\boldsymbol{T}_{ij}$ satisfy the requirements of Alg. 7 `RobustBlockUpdate`. Finally, assumption (13) ensures that the augmented vectors $\langle \alpha_i, x_i \rangle$ are initialized in a manner which satisfy the requirements of both kernels. The critical property, i.e., $\|\boldsymbol{x}_j\|_\infty \leq \Omega$, is then carried inductively through the execution of the algorithm.

4 A robust parallel algorithm

It is clear that Alg. 8 `RobustBlockedSolve` is virtually identical to Alg. 5 `BlockedSolve`. The robust algorithm has been obtained by replacing the two central kernels with robust variants operating on augmented vectors. Both algorithms can be executed in parallel using a run-time system such as `StarPU`. The overhead for the robust algorithm is marginally larger because each augmented vector contains a single extra element (the scaling) which has to be communicated. Moreover, the robust algorithm requires one final exchange of information in order to obtain a consistent scaling for all the augmented vectors. Otherwise, the number of messages exchanged is the same for the two algorithms. If there are multiple right hand sides, then consistent scalings can be obtained using a single exchange of information.

5 Conclusion

We have carefully derived scalar and blocked algorithms for solving upper triangular linear systems in a robust manner. The robust blocked algorithm can be executed in parallel

Algorithm 8: $\langle \alpha, x \rangle = \texttt{RobustBlockedSolve}(\boldsymbol{T}, \boldsymbol{b})$

Data: A non-singular upper triangular matrix $\boldsymbol{T} \in \mathbb{C}^{n \times n}$ and $\boldsymbol{b} \in \mathbb{C}^n$ partitioned conformally as equation (9) and subject to the following constraints:

1. The matrix: All super diagonal elements t_{ij} must satisfy

$$|t_{ij}| \leq \Omega \tag{11}$$

and all super diagonal blocks must satisfy

$$\|\boldsymbol{T}_{ij}\|_\infty \leq \Omega. \tag{12}$$

2. The right hand side: All elements must satisfy

$$|b_j| \leq \Omega. \tag{13}$$

Result: An augmented vector $\langle \alpha, \boldsymbol{x} \rangle$ such that

$$\boldsymbol{T}\boldsymbol{x} = \alpha \boldsymbol{b}.$$

Moreover, the scaling ensures that calculation of $\boldsymbol{x}$ never exceeds Ω.

```
for i = 1, ..., N do
    ⟨α_i, x_i⟩ ← ⟨1, b_i⟩;
for j = N, N − 1, ..., 1 do
    ⟨α_j, x_j⟩ ← RobustBlockSolve(T_jj, ⟨α_j, x_j⟩);
    for i = 1, 2, ..., j − 1 do
        ⟨α_i, x_i⟩ ← RobustBlockUpdate(⟨α_i, x_i⟩, T_ij, ⟨α_j, x_j⟩)
α = min{α_1, α_2, ..., α_N};
for i = 1, 2, ..., N do
    x_i ← (α/α_i) x_i;
return ⟨α, x⟩;
```

using a run-time system such as `StarPU` with a parallel overhead which is marginally larger than the standard algorithm. The cost of obtaining a final consistent scaling can be amortized over multiple right hand sides.

Acknowledgment

This work is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671633. The authors are very grateful to Sven Hammarling (NAG) because he carefully read the manuscript and made several valuable suggestions and comments.

References

[1] Edward Anderson. Robust Triangular Solves for Use in Condition Estimation. LAWN 36, Cray Research Inc., August 1991.

[2] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, 23:187–198, February 2011.

[3] Mark R. Fahey. New Complex Parallel Eigenvalue and Eigenvector Routines. LAWN 153, Computational Migration Group, August 2001.