Rekursiv omordning av egenvärden i en matris på Schurform

Recursive eigenvalue reordering in real Schur form

Maria Näsholm

14 juni 2006
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Robert Granat
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Sammanfattning

Den här rapporten introducerar och partiellt utvärderar en nyutvecklad rekursiv algoritm som flyttar egenvärden i en matris som är på Schurform. En kvasitriangulär matris T är på Schurform då $T = Q^T AQ$, där Q är ortogonal och $A \in \Re^{n \times n}$ är en tät matris.

Den rekursiva algoritmen delar matrisen T i mindre delar $T = \begin{bmatrix} T_{11} & T_{12} \\ & T_{22} \end{bmatrix}$ och därefter T_{11} och T_{22} rekursivt i mindre delar tills dess att matriserna är mindre än en viss given storlek $NB \times NB$. Sen flyttar vi om egenvärdena i de små matriserna med en existerande algoritm. Efter omflyttningen uppdaterar vi de delar utav matriserna T och Q som berörs utav flyttningen. Sedan flyttas egenvärderna från T_{22} till T_{11}.

När $10 - 85$ % av egenvärderna i matrisen ska flyttas är den rekursiva algoritmen snabbare än standardalgoritmen i LAPACK.

En tänkbar förbättring av den nya algoritmen är att specialisera den algoritm som flyttar egenvärderna i basfallet och låta rekursionen bli ännu djupare, kanske ner till 4×4 matriser.

Abstract

This Master Thesis report is about a recursive algorithm that reorder eigenvalues in a real Schur form matrix. If $A \in \Re^{n \times n}$, then there exists an orthogonal $Q \in \Re^{n \times n}$ such that $T = Q^T AQ$ is a upper quasi-triangular matrix. Then the matrix T is in real Schur form.

The recursive algorithm split matrix T into smaller parts $T = \begin{bmatrix} T_{11} & T_{12} \\ & T_{22} \end{bmatrix}$. Then we split T_{11} and T_{22} recursively into smaller parts until they are less than a given size $NB \times NB$. Then we reorder the eigenvalues in the smallest matrices with an existing algorithm and update parts of matrix T and Q that have been affected of the reorders. After that we move eigenvalues from T_{22} to T_{11}.

The recursive algorithm is faster than LAPACK:s standard algorithm when we reorder $10 - 85$ % of the eigenvalues.

A possible improvement of the recursive algorithm is to specialize the algorithm that reorder the eigenvalues in the base case and let the recursion become more deep, maybe until 4×4 matrices.

Innehåll

Figurer

Algoritmer

Kapitel 1

Introduktion

Linjär algebra är idag användbart inom många områden, till exempel vid simuleringar av olika fysikaliska fenomen. I många av problemen ska ett så kallat *egenvärdesproblem*,

$$Ax = \lambda x,$$

lösas i något steg av beräkningen. Det är en beräkning som kan ta lång tid och en effektivisering och uppsnabbning av att lösa egenvärdesproblemet är önskvärt. Denna förbättring utav algoritmen får dock inte ske på bekostnad utav lösningens riktighet.

Genom att beräkna Schurformen, $Q^T AQ = T$, kan man lösa egenvärdesproblemet $Ax = \lambda x$ för en tät matris utan att explicit beräkna A:s egenvektorer [3]. Det är lätt att beräkna egenvärderna till A då de finns på diagonalen i T [3]. Om $Q = [q_1, \ldots, q_n]$ är en kolumnpartitionering av Q kan *Schurvektorerna* $q_1, \ldots, q_n$ användas istället för egenvektorerna om de ordnas på lämligt sätt.

Ibland är det bra att kunna sortera egenvärderna i Schurformen efter till exempel ett visst mönster. Detta ger upphov till invarianta underrum. Ett underrum $S \subseteq \mathcal{C}^n$ med egenskapen att

$$x \in S \Rightarrow Ax \in S$$

sägs vara invariant för A.

Idag finns det ett flertal algoritmer som omordnar egenvärderna med direkta metoder. I LAPACK finns en standardalgoritm `DTRSEN` [2] och Robert Granat har tillsammans med Bo Kågström utvecklat en *blockad* motsvarighet [4] utifrån Daniel Kressners idé om uppskjutna uppdateringar [7]. Med blockad menas (i stora drag) att matrisen delas in i mindre delmatriser som bättre matchar minneshierarkierna i dagens datorer. Omordningen sker lokalt i delmatrisen och sedan uppdateras de delar i matrisen som påverkas utav omordningen.

I dag finns ingen algoritm som omordnar egenvärderna med hjälp av *rekursion*. Med rekursion menas att funktionen anropar sig själv inuti funktionen för att dela upp problemet i mindre delar. I varje steg kontrolleras ifall problemet uppfyller ett visst rekursionsvillkor, det är det så kallade *basfallet*. Då löses ett litet delproblem och det returneras. Sedan används lösningen från två eller flera delproblem för att lösa det ursprungliga problemet. När man använder rekursion vill man lösa ett mindre och enklare problem i basfallet än det ursprungliga. Om det extra arbetet som utförs på grund av att vi använder oss av rekursion inte är allt för kostsamt (i tid och flops) så är det ofta effektivare att använda rekursion i och med att vi löser flera små enklare problem

istället för ett stort och mer komplicerat. Rekurisv blockning ger dessutom ofta positiva cacheffekter.

I den här rapporten kommer vi först att gå igenom vad en Schurform är och hur man räknar ut den. Sedan kommer vi att titta på hur man gör för att byta plats på två egenvärden i en matris på Schurform. Efter det kommer vi att titta lite på några algoritmer som omordnar egenvärdena i Schurformen med direkta metoder. Sedan kommer vi att titta på en rekursiv metod som omordnar egenvärden i Schurformen och försöka utvärdera den. Till sist kommer vi att ge några förslag på förbättringar av den rekursiva algoritmen.

Kapitel 2

Problembeskrivning och målsättning

I nuläget finns det inga rekursiva algoritmer som omordnar egenvärderna i en matris som är på Schurform. Det här arbetet går ut på att utveckla en rekursiv algoritm för detta ändamål. Den rekursiva algoritmen som vi utvecklar ska sedan jämföras med standardalgoritmen i LAPACK som omordnar egenvärden i en matris på Schurform. Målet är att den rekursiva algoritmen ska vara snabbare än den befintliga i LAPACK. Det som talar för att vi ska lyckas med en uppsnabbning är att vi kommer att dela in problemet i mindre och enklare bitar. Men när vi gör det innebär det även att vi kommer att utföra fler flops.

Kapitel 3

Teori

I det har kapitlet ska vi titta lite närmare på Schurformen. Hur man beräknar en matris Schurform och hur omordning av egenvärden går till i en matris på Schurform. Slutligen kommer vi att titta närmare på nuvarande algoritmer och nya blockade algoritmer som omordnar egenvärden i en matris på Schurform.

3.1 Schurform

En övertriangulär matris med 1×1 och/eller 2×2 block längs diagonalen kallas för en *kvasitriangulär* matris.

Sats 3.1.1. *(7.4.1 i [3]) En matris $A \in \Re^{n \times n}$ har Schurformen T om vi kan hitta en ortogonal matris $Q \in \Re^{n \times n}$ så att*

$$Q^T A Q = T = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1m} \\ 0 & R_{22} & \cdots & R_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{mm} \end{bmatrix},$$

där R_{ii} är en 1×1 matris eller en 2×2 matris med komplexa konjugata egenvärden.

Med komplexa konjugatet innebär att 2×2 matrisen är på formen

$$R_{ii} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix},$$

där $a, b \in \Re$.

Observera att i en triangulär matris är alla egenvärden längs diagonalen, det gäller alltså även för en matris på Schurform [3].

3.1.1 Beräkna Schurformen

För att beräkna en matris Schurform beräknas först matrisens Hessenbergform och sedan utförs iterativa QR-algoritmer.

En matris $A(n \times n)$ har *Hessenbergformen*

$$H = U_0^T A U_0,$$

där $U_0^T U_0 = I$ och U_0 är en produkt av $n-2$ Householdermatriser, $P_1 \dots P_{n-2}$. Där varje Householdermatris, P_k reducerar kolumn k så att det blir nollor under subdiagonalen. (Subdiagonalen i en matris är de element som ligger under diagonalen.)

En *Householdermatris* är en $n \times n$ matris

$$P_k = I - \frac{2}{v_k^T v_k} v_k v_k^T,$$

där v_k är kolumn k i matrisen A [3]. En Householdermatris är ortogonal och symmetrisk [3].

För att reducera elementen på subdiagonalen använder vi oss utav en iterativ QR-algoritm, se [3] för en närmare beskrivning.

3.1.2 Standardmetoder för att ordna egenvärden i Schurformen

Att omordna egenvärden i en matris som är på Schurformen är ett sätt att räkna ut en matris invarianta underrum.

Vi börjar med att titta på en 2×2 matris

$$Q_F^T A Q_F = T_F = \begin{bmatrix} \lambda_1 & t_{12} \\ 0 & \lambda_2 \end{bmatrix}, \lambda_1 \neq \lambda_2,$$

där vi vill byta plats på λ_1 och λ_2. Då ska vi hitta en ortogonal matris Q_D så att

$$Q_D^T T_F Q_D = \begin{bmatrix} \lambda_2 & X \\ 0 & \lambda_1 \end{bmatrix}.$$

Notera att $T_F x = \lambda_2 x$ då $x = \begin{bmatrix} t_{12} \\ \lambda_2 - \lambda_1 \end{bmatrix}$.

För att hitta ett Q_D använder vi oss utav *Givens rotation*

$$G(i,k,\theta) = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & cos(\theta) & \dots & sin(\theta) & \dots & 0 \\ \vdots & & & \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & -sin(\theta) & \dots & cos(\theta) & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 0 \end{bmatrix},$$

vilken roterar "ik-planet" θ grader. Låt Q_D vara en Givens rotation så att den andra komponenten i $Q_D^T x$ är 0. Observera att Q_D är ortogonal. Om $Q = Q_F Q_D$ då är

$$(Q^T A Q)e_1 = (Q_D^T Q_F^T A Q_F Q_D)e_1 = Q_D^T T_F (Q_D e_1) = \lambda_2 Q_D^T (Q_D e_1) = \lambda_2 e_1$$

och $Q^T A Q$ måste vara på formen

$$Q^T A Q = \begin{bmatrix} \lambda_2 & \pm t_{12} \\ 0 & \lambda_1 \end{bmatrix}.$$

Algoritm 1 sammanfattar hur man omordnar egenvärden i en matris på Schurform utan 2×2 block.

Algoritm 1 Ordnar egenvärden i en matris på Schur form (se alg 7.6.1 i [3])

Input: En ortogonal matris $Q \in \Re^{n\times n}$, en övertriangulär matris $T = Q^T AQ$ och $\delta = \lambda_1, \ldots \lambda_p$ av $\lambda(A)$.
Output: En ortogonal matris Q_D där $Q_D^T T Q_D = S$ är en övertringulär matris och $s_{11}, \ldots, s_{pp} = \delta$. T och Q skrivs över av QQ_D respektive S.

while $\{t_{11}, \ldots, t_{pp}\} \neq \delta$ **do**
  for $k = 1 : n-1$ **do**
    if $t_{kk} \notin \delta$ och $t_{k+1,k+1} \in \delta$ **then**
      $[c, s] = \mathbf{givens}(T(k, k+1), T(k+1, k+1) - T(k, k))$
      $T(k : k+1, k : n) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T T(k : k+1, k : n)$
      $T(1 : k+1, k : k+1) = T(1 : k+1, k : k+1) \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$
      $Q(1 : n, k : k+1) = Q(1 : n, k : k+1) \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$
    end if
  end for
end while

Algoritm 1 behöver $k(12n)$ flops där k är antalet swaps. En övre gräns på k är $(n-p)p$ [3]. Det här sättet kan till exempel användas till att sortera egenvärderna stigande eller fallande längs diagonalen.

I LAPACK finns algoritmen `SLAEXC` som Bai och Demmel har utvecklat [2]. Den byter plats på 2×2 och 1×1 block i en matris på Schurform, som vi nu ska titta närmare på:

Antag att blockmatrisen $T = \begin{bmatrix} T_{11} & T_{12} \\ & T_{22} \end{bmatrix}$, där T_{11} är $p \times p$ och T_{22} är $q \times q$ där $p, q = 1, 2$. T_{11} och T_{22} har inga gemensamma egenvärden, det vill säga $\lambda(T_{11}) \cap \lambda(T_{22}) = \emptyset$. Vi kan dela upp matrisen

$$T = \begin{bmatrix} T_{11} & T_{12} \\ & T_{22} \end{bmatrix} = \begin{bmatrix} I_p & -X \\ & I_q \end{bmatrix} \begin{bmatrix} T_{11} & 0 \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} I_p & X \\ & I_q \end{bmatrix},$$

där X är lösningen på Sylvesterekvationen $T_{11}X - XT_{22} = T_{12}$. X är unik ty $\lambda(T_{11}) \cap \lambda(T_{22}) = \emptyset$. Beräkna nu en ortogonal matris $Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}$ så att

$$Q^T \begin{bmatrix} -X \\ I_q \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix}.$$

Antag vidare att R och Q_{12}^T är inverterbara och vi har:

$$\begin{aligned}
Q^T T Q &= Q^T \begin{bmatrix} T_{11} & T_{12} \\ & T_{22} \end{bmatrix} Q \\
&= Q^T \begin{bmatrix} I_p & -X \\ & I_q \end{bmatrix} \begin{bmatrix} T_{11} & 0 \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} I_p & X \\ & I_q \end{bmatrix} Q \\
&= \begin{bmatrix} R & Q_{11}^T \\ 0 & Q_{12}^T \end{bmatrix} \begin{bmatrix} T_{22} & 0 \\ 0 & T_{11} \end{bmatrix} \begin{bmatrix} R^{-1} & -R^{-1}Q_{11}^T Q_{12}^{-T} \\ 0 & Q_{12}^{-T} \end{bmatrix} \\
&= \begin{bmatrix} RT_{22}R^{-1} & -RT_{22}R^{-1}Q_{11}^T Q_{12}^{-T} + Q_{11}^T T_{11} Q_{12}^{-T} \\ 0 & Q_{12}^T T_{11} Q_{12}^{-T} \end{bmatrix} \\
&= \begin{bmatrix} \tilde{T_{22}} & \tilde{T_{12}} \\ & \tilde{T_{11}} \end{bmatrix}
\end{aligned}$$

Vi har nu bytt plats på egenvärderna i T_{11} och T_{22}. Den här metoden fungerar dock inte om egenvärderna i T_{11} är väldigt nära egenvärderna i T_{22} [2].

3.2 Standardalgoritm som omordnar egenvärden

I LAPACK finns en algoritm som omordnar egenvärden i Schurformen; `DTRSEN`. Den använder bubbelsortering när den flyttar egenvärderna ett och ett, se algoritm 2. Denna algoritm kräver $O(kn^2)$ flops, där k är antalet egenvärden som ska flyttas. [7].

Algoritm 2 LAPACKs rutin som ordnar egenvärden i en matris på Schurform

Input: $T(M \times M)$ i kvasitriangulär form. $Q(M \times M)$ en tät matris, $select(1 \times M)$.
Output: $T(M \times M)$ i kvasitriangulär form, $Q(M \times M)$ en tät matris.

function $[T, Q]$= **dtrsen**(T, Q)
  $j \leftarrow 0$
  for $i \leftarrow 1, \ldots, m$ **do**
    if $\lambda(T_{ii}) \subset \Lambda_s$ **then**
      $j \leftarrow j + 1$, $select(j) \leftarrow i$
    end if
  end for
  $top \leftarrow 0$
  for $l \leftarrow 1, \ldots, j$ **do**
    for $i \leftarrow select(l), select(l) - 1, \ldots, top + 1$ **do**
      Byt plats på $T_{i-1,i-1}$ och $T_{i,i}$ med hjälp av en ortogonal transformation och uppdatera de övriga raderna och kolumnerna.
    end for
    $top \leftarrow top + 1$
  end for

3.3 Nya blockade algoritmer som omordnar egenvärden

Daniel Kressner har utvecklat en ny blockad algoritm som omordnar egenvärderna i Schurformen [7]. Istället för att flytta egenvärderna ett och ett samlas de ihop i ev stycken åt gången, där ev är givet. Ett ”fönster” av storleken $w \times w$, $ev < w$, placeras

så att det ev:te egenvärdet är i det högra nedre hörnet av fönstret. Sedan omordnas egenvärderna inom fönstret och de delar utav matrisen som berörs av omflyttningen uppdateras innan fönstret flyttas så att det ligger med det nedre högra hörnet över det ev:te egenvärdet igen. När de första ev antalet egenvärden är samlade i matrisens övre vänstra hörn flyttas nästa ev egenvärden tills dess att alla egenvärden är flyttade. Se algoritm 3.

Algoritm 3 Blockalgoritm som ordnar egenvärden i en matris på Schurform [7]

Input: $T(n \times n)$ i kvasitriangulär form. $Q(n \times n)$ en tät matris, w fönsterstorleken och $ev \leq w$ max antal egenvärden som flyttas i varja fönster, $select(1 \times n)$.
Output: $T(n \times n)$ i kvasitriangulär form, $Q(n \times n)$ en tät matris.

function $[T, Q] = (T, Q, w, ev, select)$
  $i_{ord} \leftarrow 0$ % i_{ord} antalet egenvärden som är på sin slutliga plats
  while $i_{ord} < \#\Lambda_s$ **do**
    %Hitta de första $n_{ev} \leq$ osorterade egenvärderna från toppen
    $n_{ev} \leftarrow 0$, $i_{hi} \leftarrow i_{ord} + 1$
    while $n_{ev} \leq ev$ och $\imath_{hi} \leq n$ **do**
      if $T_{ii} \in \Lambda_s$ **then**
        $n_{ev} \leftarrow n_{ev} + 1$
      end if
      $i_{hi} \leftarrow i_{hi} + 1$
    end while
    %Omordna egenvärderna i fönster efter fönster mot toppen.
    while $i_{hi} > i_{ord} + n_{ev}$ **do**
      $i_{low} \leftarrow max(i_{ord} + 1, i_{hi} - w + 1$, $n_w \leftarrow i_{hi} - i_{low} + 1)$
      Använd algoritm 2 i det aktiva fönstret $T(i_{low} : i_{hi}, i_{low} : i_{hi})$ och omordna de $k \leq n_{ev}$ utvalda egenvärderna till toppen av fönstret. Låt tillhörande ortogonala transformationsmatrisen betecknas U.
      Uppdatera $T(i_{low} : i_{hi}, i_{hi} + 1 : n) \leftarrow U^T T(i_{low} : i_{hi}, i_{hi} + 1 : n)$.
      Uppdatera $T(1 : i_{low} - 1, i_{low} : i_{hi}) \leftarrow T(1 : i_{low} - 1, i_{low} : i_{hi})U$.
      Uppdatera $Q(1 : n, i_{low} : i_{hi}) \leftarrow Q(1 : n, i_{low} : i_{hi}U$.
      $i_{hi} \leftarrow i_{low} + k - 1$
    end while
    $i_{ord} \leftarrow i_{ord} + n_{ev}$
  end while

Kressners algoritm är signifikant bättre än standardalgoritm `DTRSEN` i LAPACK [7]. När matrisstorleken är tillräckligt stor använder Kressners algoritm mindre än 25% av den tid som det tar för `DTRSEN`.

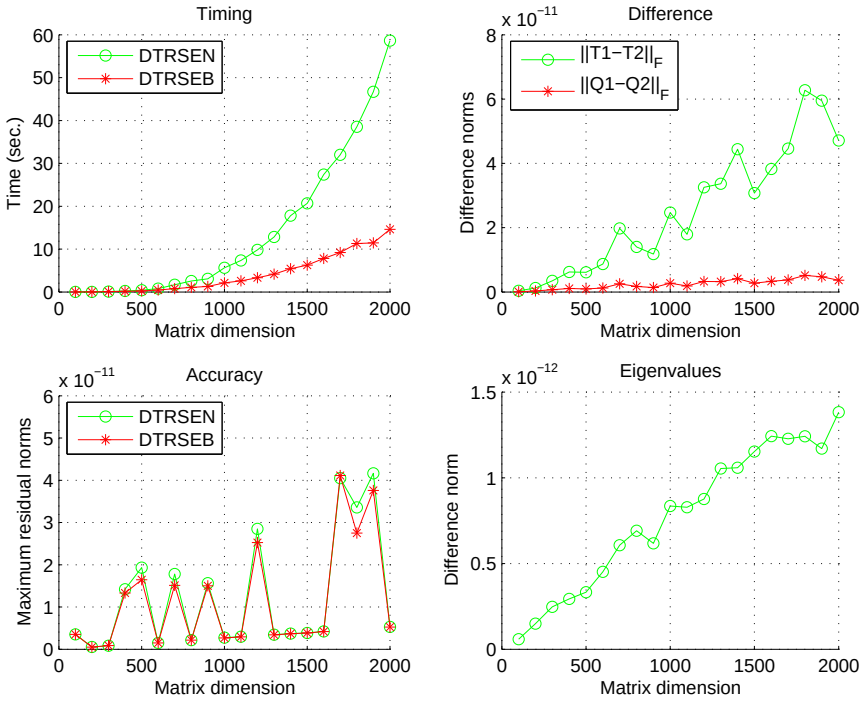
Robert Granat och Bo Kågström har utvecklat en blockalgoritm [4], `DTRBSE` utifrån Kressners algoritm [7]. Den börjar med att dela in matrisen $T(m \times m)$ i block med en fix blockstorlek på $mb \times mb$ ($mb < m$ är input). Då det finns egenvärden som ska flyttas väljs det första blocket längs diagonalen (närmast det övre vänstra hörnet). Sedan flyttas egenvärderna inom blocket så att de ligger samlade på en plats (som vanligt i det övre vänstra hörnet). Sedan flyttas alla samlade egenvärden från det blocket till blocket närmast över längs diagonalen tills dess att de ligger samlad med eventuella andra egenvärden längst upp i matrisen T:s vänstra hörn. Mellan varje flyttning från block till block uppdateras de delar utav matriserna T och Q som påverkas av omordningen. Se algoritm 4 för detaljer.

Algoritm 4 Blockalgoritm som ordnar egenvärden i en matris på Schurform [4]

Input: $T(M \times M)$ i kvasitriangulär form. $Q(M \times M)$ en tät matris, $mb \leq M$ blockstorleken, $select(1 \times M)$.
Output: $T(M \times M)$ i kvasitriangulär form, $Q(M \times M)$ en tät matris.

function [T, Q]= **dtrbse**(T, Q, mb, $select$)
  $lstind = 1$
  $lstblk = 1$
  $fstblk =$ Första blocket längs diagonalen i T där det finns utvalda egenvärden
  $db = \lceil m/mb \rceil$
  while $fstblk \leq db$ **do**
    for $i = fstblk : lstblk : -1$ **do**
      $\Lambda_i =$utvalda egenvärden som ska samlas i block i
      $clsum =$ antal element i Λ_i
      if $clsum > mb - 1$ **then**
        $dvclstr = true$
        $\Lambda_i = \Lambda_i/2$
        $clsum = clsum/2$
      else
        $dvclst = false$
      end if
      if $i = fstblk$ och icke $dvclstr$ och valt 2×2 block i
      $T(i \cdot mb : i \cdot mb + 1, i \cdot mb : i \cdot mb + 1)$ **then**
        Flytta det delade 2×2 blocket och samla Λ_i i det aktuella blockets övre vänstra hörn.
        Uppdaterar de tillhörande raderna och kolumnerna i T och Q.
      end if
      if $clsum > 0$ **then**
        Flytta Λ_i över gränsen och samla dem i det övre vänstra hörnet i nästa block.
        Uppdatera de tillhörande raderna och kolumnerna i T och Q.
      end if
    end for
    $lstind = lstind + 1$
    $lstblk =$mod$(lstind, mb)$
    if icke $dvclstr$ **then**
      $fstblk = fstblk + 1$
    end if
  end while

Den här algoritmen är också betydligt snabbare än LAPACKs `DTRSEN`, se bilden överst till vänster i Figur 3.1. I samma bild ser vi även att Granats och Kågströms algoritm är betydligt snabbare desto större matrisstorleken blir.



Figur 3.1: Grafer över Granats och Kågströms algoritms prestanda kontra standardalgoritm i LAPACKs prestanda

Kapitel 4

Utförande

4.1 Hur arbetet utfördes

Utifrån idéer om hur den rekursiva algoritmen ska fungera implementerades den först i MATLAB. I MATLAB kunde vi sedan utveckla algoritmen ytterligare då vissa brister upptäcktes. När implementationen var klar i MATLAB och vi var nöjd med vår rekursiva algoritm fortsatte utvecklingen i programspråket FORTRAN, vilket är det språk som den slutliga algoritmen ska vara implementerad i.

När vi implementerat en rekursiv fungerande algoritm i FORTRAN optimerade vi den ytterligare genom att bland annat byta ut funktionen som vi använder när vi når basfallet i rekursionen från `DTRSEN` till `DTRBSE` och ändra matrismultiplikationerna till en optimerad motsvarighet både i `RDTRSEB` och `DTRBSE`.

4.2 Rekursiv algoritm

`RDTRSEB` är en rekursiv algoritm där vi använder oss utav `DTRBSE` när vi når basfallet i rekursionen. Basfallet uppstår då matriserna T $(M \times M)$ och Q $(M \times M)$ är mindre än en förutbestämd storlek $NB \times NB$. I varje rekursionssteg delar vi på matriserna på följande sätt

$$T = \begin{bmatrix} T_{11} & T_{12} \\ & T_{22} \end{bmatrix} \tag{4.1}$$

och

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \tag{4.2}$$

i delarna T_{11}, T_{12}, T_{22}, Q_{11}, Q_{12}, Q_{21} samt Q_{22}. I nästa steg delar vi på samma sätt T_{11} och T_{22} och så vidare tills dess att storleken är mindre än den förutbestämda blockstorleken NB. Sen använder vi oss utav `DTRBSE` för att omordna egenvärderna i varje delmatris till

$$\tilde{T}_{11} = Z_1^T T_{11} Z_1 \tag{4.3}$$

samt

$$\tilde{T}_{22} = Z_2^T T_{22} Z_2, \tag{4.4}$$

där Z_1 och Z_2 är två ortogonala matriser som omordnar egenvärderna i T_{11} respektive T_{22}. Efter omordningen av egenvärderna uppdaterar vi T till

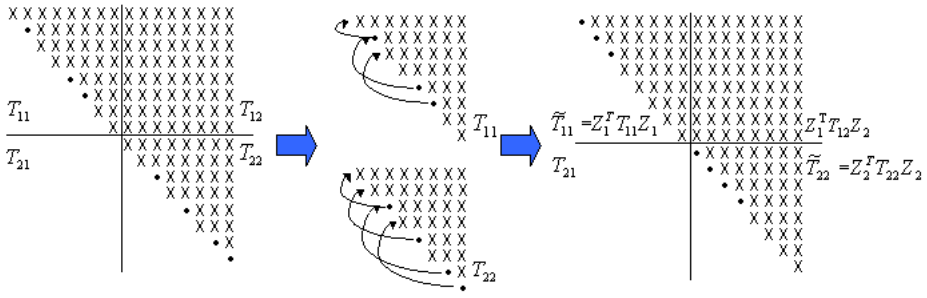
$$\tilde{T} = \begin{bmatrix} \tilde{T}_{11} & Z_1^T T_{12} Z_2 \\ & \tilde{T}_{22} \end{bmatrix} \tag{4.5}$$

och Q till

$$\tilde{Q} = \begin{bmatrix} Q_{11}Z_1 & Q_{12}Z_2 \\ Q_{21}Z_1 & Q_{22}Z_2 \end{bmatrix}. \tag{4.6}$$

Observera att vi inte har flyttat alla valda egenvärden längst upp i T_{11} utan att de ligger samlad på två ställen, några är längst upp i T_{11} och några samlad längst upp i T_{22}.

Ett exempel på hur det kan se ut ser vi i Figur 4.1 där vi anropar `RDTRSEB` för T_{11} och T_{22} samt uppdaterar T_{12} med avseende på omordningen.



Figur 4.1: En illustration över hur vi flyttar egenvärderna, där • betecknar de egenvärden som ska flyttas.

För att inte dela på något 2×2 block i matrisen T tar vi hänsyn till de 1×1 och 2×2 blocken som finns på diagonalen. Om delningspunkten hamnar mitt i ett 2×2 block delar vi just under blocket istället, se Algoritm 5 där vi beskriver vår rekursiva algoritm.

För att flytta egenvärderna från T_{22} till T_{11} väljer vi ut den del utav T som berörs av flyttningen. Det vill säga, vi väljer en delmatris som börjar just under de egenvärden som vi har flyttat i T_{11} och sträcker sig över de egenvärden som ska flyttas från T_{22}, se Figur 4.2. Även om vi endast vill flytta om egenvärderna i det utvalda området i matrisen berörs även andra delar utav matrisen. Vi börjar med att omordna egenvärderna i den utvalda delen av matrisen så att

$$\tilde{T}_u = Z_u^T T_u Z_u, \tag{4.7}$$

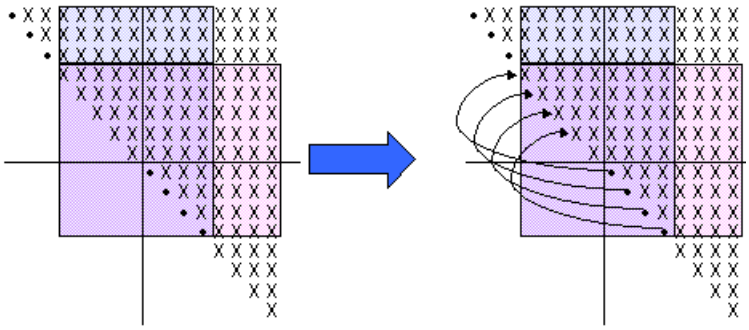
där Z_u är den ortogonala matris som omordnar egenvärderna i T_u. Sedan uppdaterar vi de delar utav matrisen som berörs av flyttningen. Genom att muliplicera den delen av matrisen som ligger ovanför det utvalda området med Z_u från höger och den delen av matrisen som ligger till höger om det utvalda området multipliceras med Z_u^T från vänster.

När vi ska flytta de valda egenvärderna över gränsen till T_{11} använder vi oss utav vår rekursiva algoritm igen. Det finns dock några tillfällen då vi inte kan använda den. Det är då ”huvudmatrisens” delningspunkt sammanfaller eller blir väldigt nära vår delmatris delningspunkt. Rekursionen når då aldrig basfallet utan det enda som inträffar är att

Algoritm 5 Rekursiv algoritm som ordnar egenvärden i en matris på Schurform

Input: $T(M \times M)$ i kvasitriangulär form. $Q(M \times M)$ en tät matris. blks, blockstorleken som anger när vi ska byta till standardalgoritm. select$(1 \times M)$, anger vilka element vi ska flytta (1-flytta, 0-behålla).
Output: $T(M \times M)$ i kvasitriangulär form, $Q(M \times M)$ tät matris. $select(1 \times M)$, m som anger hur många element som har flyttas.

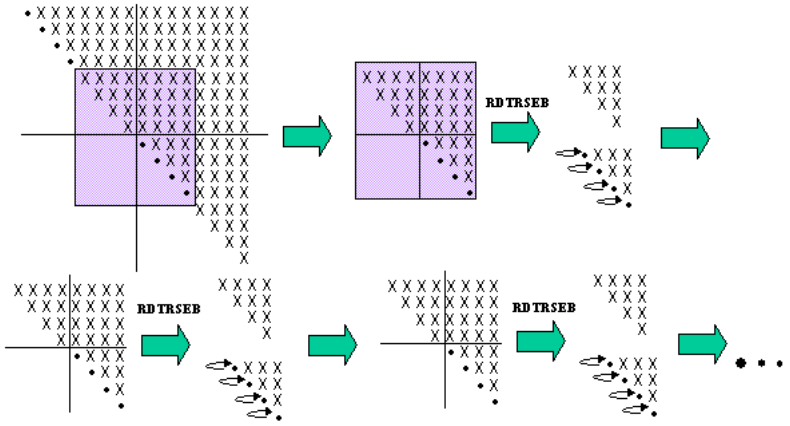
function $[T, Q, select, m] =$ **rdtrseb**$(T, Q, blks, select)$
  if $M > blks$ **then**
    if $T(M/2{+}1, M/2) \neq 0$ **then**
      $k = M/2{+}1$;
    else
      $k = M/2$;
    end if
    $[T_{11}, Z_1, select(1{:}k), m_1] =$ **rdtrseb**$(T_{11}, Q_{11}, blks, select(1{:}k))$;
    $[T_{22}, Z_2, select(k{+}1{:}M), m_2] =$ **rdtrseb**$(T_{22}, Q_{22}, blks, select(k{+}1{:}M))$;
    $T_{12} = Z_1^T T_{12} Z_2$;
    $Q = [Q_{11}Z_1, Q_{12}Z_2; Q_{21}Z_1, Q_{22}Z_2]$;
    Flytta egenvärderna över gränsen, se algoritm 6.
  else
    $[T, Q, select, m] =$**dtrseb**$(select, T, Q)$;
  end if



Figur 4.2: En illustration över hur vi väljer vilken delmatris som vi ska använda för att flytta egenvärden från $\tilde{T}_{22}$ till $\tilde{T}_{11}$.

vi delar på matrisen T, omordnar i varje del (vilket inte behövs eftersom det redan är omordnat) och sedan anropar vi `RDTRSEB` igen för att flytta över gränsen. I Figur 4.3 är ett enkelt exempel på när rekursionen fastnar illustrerat.

När det inträffar att rekursionen fastnar använder vi oss utav `DTRBSE` på delmatrisen, vilken vi använder på en så liten del av matrisen som möjligt. För att kunna göra det flyttar vi endast över de egenvärden som finns i T_{22} just över gränsen med hjälp av `DTRBSE` och sedan använder vi oss av `RDTRSEB` för att flytta dessa egenvärden igen (om det behövs) till just under de egenvärden som från början fanns i T_{11}. Då det finns fler egenvärden i T_{22} än platser ovanför delningspunkten flyttar vi endast det antal som får plats i T_{11} och sedan använder vi `RDTRSEB` för att ordna egenvärderna i T_{22} igen.



Figur 4.3: Ett exempel på hur rekursionen kan ”fastna”

Se algoritmbeskrivning 6 i Appendix för en detaljerad beskrivning på hur vi flyttar egenvärderna över gränsen.

Ett annat (kanske mer instabilt) sätt att flytta egenvärderna över gränsen är att flytta alla egenvärden i T_{22} på en gång med algoritmen som finns beskriven i 3.3. Detta har dock inte provats i denna rapport.

4.2.1 Rekursionsdjup

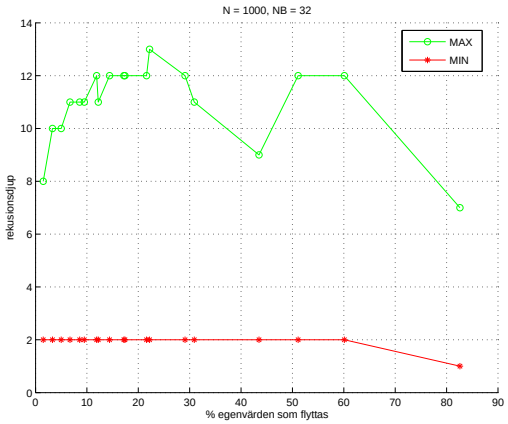
För djupare kunskaper om vår algoritm har vi först implementerat den i MATLAB och där tagit reda på bland annat rekursionsdjupet för en 1000×1000 matris med en blockstorlek på 32. Rekursiondjupet är det antal gånger som algoritmen anropar sig själv. I Figur 4.4 har vi tittat på hur rekursionsdjupet ändras då antalet egenvärden vi ska flytta ändras. I Figur 4.5 har vi tittat på hur rekursiondjupet ändras då matrisdimensionen varierar.

Att det maximala rekursiondjupet blir djupare än de sex gångerna vi behöver dela på matrisen för att den ska bli mindre än 32×32 beror på att vi anropar `RDTRSEB` för att flytta över egenvärderna från en del till en annan. Att rekursionsdjupet i Figur 4.4 är mindre då få, många eller vid ungefär 50% av egenvärderna flyttas kan bero på att de delmatriser som vi utför rekursionen på oftare innehåller endast element som ska flyttas eller element som inte ska flyttas, det vill säga vi avbryter rekursionen tidigare.

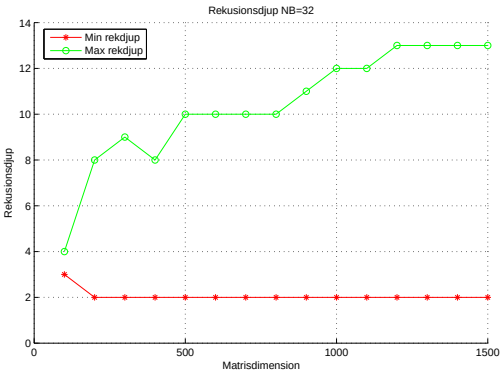
I Figur 4.5 ser vi att rekursionsdjupet blir större då matrisstorleken växer. Det beror på att blockstorleken är större i förhållande till matrisstorleken för små matriser. Det vill säga vi behöver inte dela på matrisen lika många gånger för en mindre matris för att den ska bli lika stor eller mindre än blockstorleken.

4.2.2 Implemetationsdetaljer

Eftersom vår algoritm är rekursiv använder vi oss av en mer optimerad algoritm när vi utför matrismultiplikation, istället för `DGEMM` använder vi `RECSY_GEMM`. Vi har även ersatt `DGEMM` i `DTRBSE` med `RECSY_GEMM` [9].



Figur 4.4: Rekursionsdjup där antalet element som flyttas varierar (skapad i MATLAB med liknande algoritm)



Figur 4.5: Rekursionsdjup där matrisstorleken varierar (skapad i MATLAB med motsvarande algoritm)

Första gången vår algoritm anropas kontrollerar vi att parametervärdena är rimliga. När algoritmen anropar sig själv rekursivt utförs ej dessa kontroller utan vi förutsätter att det är rätt. På så sätt utförs inte fler kontroller i `RDTRSEB` än nödvändigt.

Innan vi utför någon rekursion kontrollerar vi ifall det finns några egenvärden som ska flyttas, om det inte finns returnerar vi utan att utföra rekursionen.

Kapitel 5

Resultat och diskussion

5.1 Resultat

Testkörningarna har utförts på "Sarek", en dual AMD Opteron (2.2GHz) med 8 GB minne. Vi har kompilerat vår implementation med kompilatorn "mpif90" och länkat med RECSY [9] och GOTO-BLAS [8].

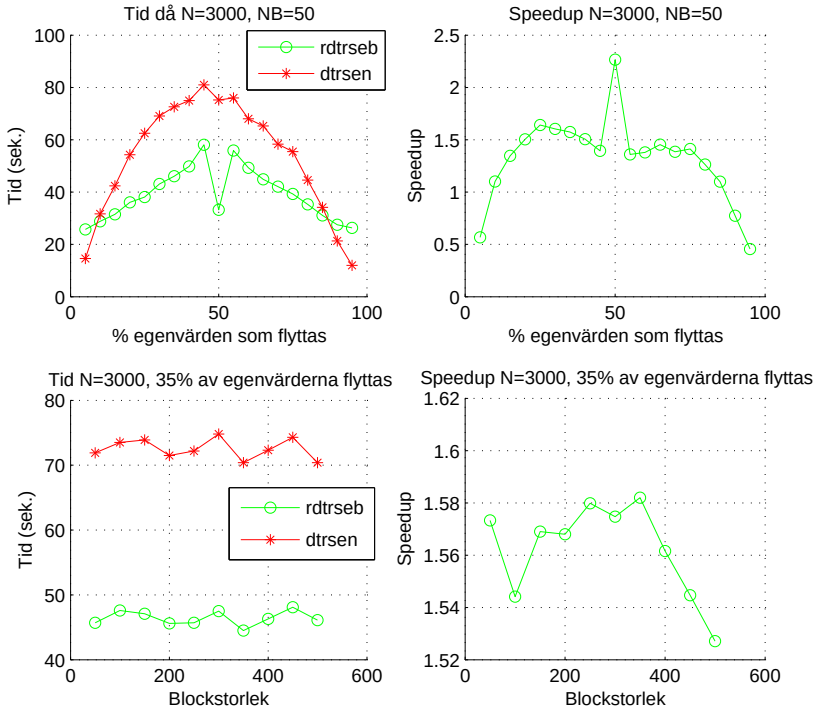
Om inte annat är angivet är de numeriska exprimenten utförda på en slumpad 3000×3000 matris som är reducerad på Schurform. Vid reduceringen använde vi LAPACKs rutiner `DGEHRD` för att beräkna Hessenbergreduktionen. Sedan satt vi alla små element till exakt 0 för att till sist använda `DHSEQR` för att räkna ut matrisens Schurform [1]. Vi har även slumpat vilka egenvärden vi ska flytta och antalet egenvärden vi flyttar anges i procent av alla egenvärden.

5.1.1 Prestanda

Vi jämför vår algoritm `RDTRSEB` med `DTRSEN` som finns i LAPACK. Med hjälp av *uppsnabbning (speedup)* kan vi på ett enkelt sätt se hur prestandan förändras. Uppsnabbning beräknas genom att dividera tiden det tar att köra problemet genom den algoritm vi jämför med genom tiden det tar att köra samma problem genom den algoritm som ska bli jämförd. I det här fallet blir det tiden det tar för `DTRSEN` dividerat med tiden för `RDTRSEB`.

När matrisstorleken är över 1000×1000 är `RDTRSEB` snabbare än `DTRSEN` och uppsnabbningen ligger på ungefär 1.5 för en 3000×3000 matris, i gynsamma fall kan den dock vara över 2. I grafen längst till höger i första raden i Figur 5.2 ser vi att uppsnabbningen är växande då matrisstorleken ökar, detta är ett tecken på att det är ännu mer förmånligt att använda `RDTRSEB` i stället för `DTRSEN` på stora matrisstorlekar.

`RDTRSEB` är dock inte alltid snabbare utan när det är väldigt få eller väldigt många egenvärden som ska flyttas blir den långsammare än `DTRSEN`. Med få egenvärden menas under 10% och med många egenvärden menas fler än 85%, se grafen till höger i första raden i Figur 5.1. Det kan bero på att de utvalda egenvärderna ligger jämnt fördelade över hela diagonalen. Det medför att även om det är få egenvärden som ska flyttas måste relativt mycket "extra" arbete utföras på grund av rekursionen. Vi kan även se att det finns en topp i uppsnabbningen i Figur 5.1 när 50% av egenvärderna flyttas. Denna topp kan bero på att det oftare förekommer delmatriser som endast har egenvärden som ska flyttas eller inga egenvärden som ska flyttas. Den kan också bero på att vi använder



Figur 5.1: Tidtagning och uppsnabbning för olika matrisdimensioner och blockstorlekar.

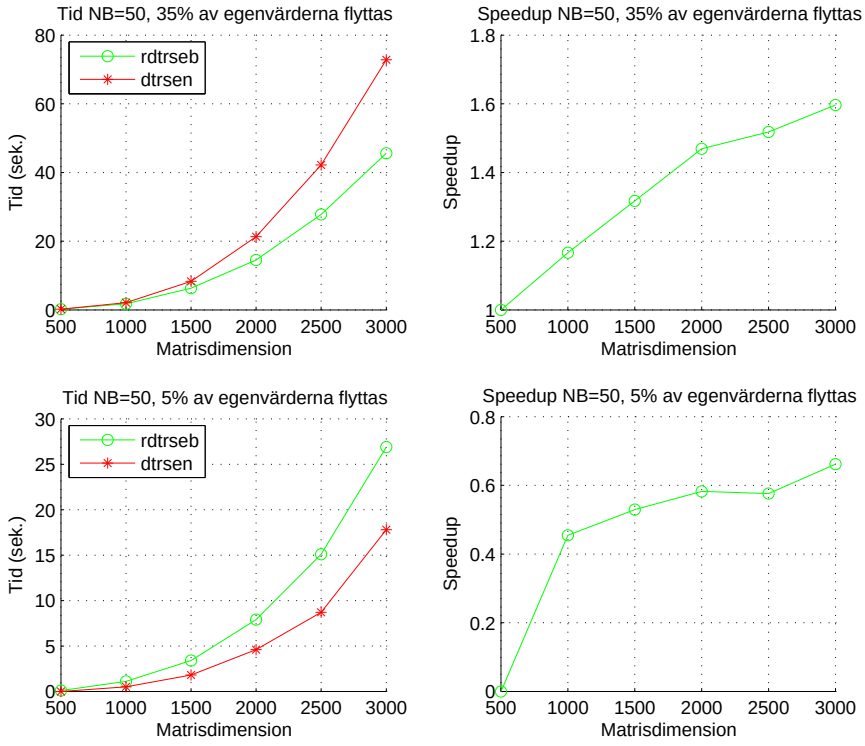
`DTRBSE` oftare för att flytta egenvärderna över ”gränsen”. Samma tendens ser vi i Figur 4.4 där rekursionsdjupet blir lägre vid ungefär 50%.

I Figur 5.2 har vi plottat hur uppsnabbningen förändras när matrisstorleken förändras för både när 35% och 5% av egenvärderna flyttas.

I grafen längst till höger på rad två i Figur 5.1 ser vi att uppsnabbningen är kring 1.5, den kan dock bli betydligt sämre då blockstorleken ökar.

5.1.2 Numerisk stabilitet

För att testa den numeriska stabiliteten i `RDTRSEB` har vi beräknat normer och residualer. Antag att T är vår ursprungliga matris på Schurform där vi inte har omordnat några egenvärden, T_1 samt Q_1 är de matriser vi får fram från `RDTRSEB` och att T_2 samt Q_2 är de vi får från `DTRSEN`. Vi har då beräknat *Frobeniusnormerna* ($||A||_F^2 = \sum\sum a_{ij}^2$) $||T_1 - T_2||_F$ och $||Q_1 - Q_2||_F$ samt residualerna $||Q_1 T_1 Q_1^T - T||_F$ och $||T_1 - Q_1^T T Q_1||_F$. Residualerna har blivit små, se Tabell 5.1.3, det vill säga ungefär 10^{-10} varje gång medan normerna $||T_1 - T_2||_F$ och $||Q_1 - Q_2||_F$ oftast är mindre än 10^{-10} men ibland har de legat på storleksordningen 10^3. Det beror på att algoritmen inte är framåt stabil. Så länge residualerna är omkring 10^{-10} kan vi acceptera att normerna är omkring 10^2.



Figur 5.2: Tidtagning och uppsnabbning för olika antal egenvärden som flyttas.

5.1.3 Resultattabell

Nedan visas en tabell med resultaten från testkörningarna. Där T_1 samt Q_1 är resultaten från `DTRSEN` och T_2 samt Q_2 är resultaten från `RDTRSEB`. I tabellen är N matrisstorleken, NB blockstorleken, t_1 tiden för `DTRSEN`, t_2 tiden för `RDTRSEB` och $\%\lambda$ är % egenvärden som flyttas. Vi har även redovisat Frobeniusnormerna $||T_1 - T_2||_F$, och $||Q_1 - Q_2||_F$, $\sqrt{\sum_{i=1}^{n}(\lambda_i^{(1)} - \lambda_i^{(2)})^2}$ (skillnaden mellan egenvärderna) samt residualerna $||Q_1 T_1 Q_1^T - T||_F$ och $||T_1 - Q_1^T T Q_1||_F$.

5.2 Framtida arbete

En vidareutveckling av `RDTRSEB` skulle kunna vara att skapa en mer specialicerad algoritm när vi når basfallet i kombination med djupare rekursion. Exempelvis använda rekursionen tills dess att matrisstorleken är 4×4 eller 3×3 och använda en specialicerad högoptimerad algoritm när vi når basfallet. Då kan vi bland annat ta bort kontrollerna som finns i början av `DTRBSE` och på så sätt utföra mindre arbete i onödan. En ytterligare förbättring vore att vidareutveckla metoden att flytta hela eller delar av konsekutiva

N	NB	t_1	t_2	$\|\|T_1 - T_2\|\|_F$	$\|\|Q_1 - Q_2\|\|_F$	$\sqrt{\sum_{i=1}^{n}(\lambda_i^{(1)} - \lambda_i^{(2)})^2}$	$\|\|Q_1 T_1 Q_1^T - T\|\|_F$	$\|\|T_1 - Q_1^T T Q_1\|\|_F$	$\%\lambda$
3000	50	14.6	25.7	$0.1E+04$	$0.7E+02$	$0.1E-11$	$0.1E-10$	$0.1E-10$	5
3000	50	31.7	28.8	$0.1E+04$	$0.6E+02$	$0.1E-11$	$0.2E-10$	$0.2E-10$	10
3000	50	42.4	31.5	$0.1E+04$	$0.7E+02$	$0.2E-11$	$0.3E-10$	$0.3E-10$	15
3000	50	54.3	36.1	$0.1E+04$	$0.6E+02$	$0.2E-11$	$0.3E-10$	$0.3E-10$	20
3000	50	62.5	38.1	$0.9E+03$	$0.5E+02$	$0.2E-11$	$0.3E-10$	$0.3E-10$	25
3000	50	69.1	43.1	$0.9E+03$	$0.5E+02$	$0.2E-11$	$0.4E-10$	$0.4E-10$	30
3000	50	72.6	46.1	$0.8E+03$	$0.5E+02$	$0.3E-11$	$0.5E-10$	$0.4E-10$	35
3000	50	75.0	49.8	$0.7E+03$	$0.4E+02$	$0.3E-11$	$0.5E-10$	$0.5E-10$	40
3000	50	81.0	58.1	$0.7E+03$	$0.4E+02$	$0.3E-11$	$0.5E-10$	$0.5E-10$	45
3000	50	75.2	33.2	$0.7E+03$	$0.3E+02$	$0.3E-11$	$0.5E-10$	$0.5E-10$	50
3000	50	76.0	55.9	$0.4E+03$	$0.2E+02$	$0.3E-11$	$0.5E-10$	$0.5E-10$	55
3000	50	68.0	49.3	$0.1E+03$	$0.5E+01$	$0.3E-11$	$0.5E-10$	$0.5E-10$	60
3000	50	65.3	44.9	$0.2E+03$	$0.6E+01$	$0.3E-11$	$0.5E-10$	$0.5E-10$	65
3000	50	58.3	42.1	$0.1E-09$	$0.3E-10$	$0.3E-11$	$0.6E-10$	$0.5E-10$	70
3000	50	55.5	39.3	$0.9E-10$	$0.4E-10$	$0.3E-11$	$0.6E-10$	$0.5E-10$	75
3000	50	44.6	35.3	$0.2E-09$	$0.2E-10$	$0.3E-11$	$0.5E-10$	$0.5E-10$	80
3000	50	34.2	31.1	$0.1E-09$	$0.2E-10$	$0.3E-11$	$0.4E-10$	$0.4E-10$	85
3000	50	21.3	27.5	$0.6E-10$	$0.2E-10$	$0.3E-11$	$0.2E-10$	$0.2E-10$	90
3000	50	12.0	26.3	$0.4E-10$	$0.1E-10$	$0.2E-11$	$0.1E-10$	$0.1E-10$	95
500	50	0.2	0.2	$0.3E-10$	$0.1E-10$	$0.9E-12$	$0.4E-11$	$0.4E-11$	35
1000	50	2.1	1.8	$0.4E-10$	$0.2E-10$	$0.1E-11$	$0.1E-10$	$0.1E-10$	35
1500	50	8.3	6.3	$0.8E-10$	$0.3E-10$	$0.2E-11$	$0.2E-10$	$0.2E-10$	35
2000	50	21.3	14.5	$0.1E+03$	$0.3E+01$	$0.2E-11$	$0.3E-10$	$0.3E-10$	35
2500	50	42.2	27.8	$0.2E+03$	$0.6E+01$	$0.3E-11$	$0.4E-10$	$0.4E-10$	35
3000	50	72.8	45.6	$0.8E+03$	$0.5E+02$	$0.3E-11$	$0.4E-10$	$0.4E-10$	35
3000	50	71.9	45.7	$0.9E+03$	$0.5E+02$	$0.3E-11$	$0.4E-10$	$0.4E-10$	35
3000	100	73.5	47.6	$0.8E+03$	$0.5E+02$	$0.3E-11$	$0.4E-10$	$0.4E-10$	35
3000	150	73.9	47.1	$0.8E+03$	$0.5E+02$	$0.3E-11$	$0.4E-10$	$0.4E-10$	35
3000	200	71.5	45.6	$0.8E+03$	$0.5E+02$	$0.2E-11$	$0.4E-10$	$0.4E-10$	35
3000	250	72.2	45.7	$0.8E+03$	$0.5E+02$	$0.2E-11$	$0.4E-10$	$0.4E-10$	35
3000	300	74.8	47.5	$0.9E+03$	$0.5E+02$	$0.2E-11$	$0.4E-10$	$0.4E-10$	35
3000	350	70.4	44.5	$0.9E+03$	$0.5E+02$	$0.2E-11$	$0.4E-10$	$0.4E-10$	35
3000	400	72.3	46.3	$0.9E+03$	$0.5E+02$	$0.2E-11$	$0.4E-10$	$0.4E-10$	35
3000	450	74.3	48.1	$0.8E+03$	$0.4E+02$	$0.2E-11$	$0.5E-10$	$0.4E-10$	35
3000	500	70.4	46.1	$0.8E+03$	$0.5E+02$	$0.2E-11$	$0.4E-10$	$0.4E-10$	35
500	50	0.0	0.1	$0.7E-11$	$0.3E-11$	$0.2E-12$	$0.1E-11$	$0.1E-11$	5
1000	50	0.5	1.1	$0.2E-10$	$0.6E-11$	$0.6E-12$	$0.4E-11$	$0.4E-11$	5
1500	50	1.8	3.4	$0.3E+03$	$0.2E+02$	$0.7E-12$	$0.7E-11$	$0.7E-11$	5
2000	50	4.6	7.9	$0.9E+03$	$0.4E+02$	$0.8E-12$	$0.1E-10$	$0.1E-10$	5
2500	50	8.7	15.1	$0.9E+03$	$0.5E+02$	$0.1E-11$	$0.1E-10$	$0.1E-10$	5
3000	50	17.8	26.9	$0.1E+04$	$0.6E+02$	$0.1E-11$	$0.1E-10$	$0.1E-10$	5

Tabell 5.1: En tabell med resultat från alla testkörningar.

egenvärdes kluster över gränsen i algoritm 5 och 6.

Kapitel 6

Tack

Jag vill tacka min handledare Robert Granat, på institutionen för Datavetenskap vid Umeå univeristet, för all hjälp jag har fått under arbetets gång.

Jag vill även tacka High Performance Computing Center North (HPC2N) i Umeå för att jag har fått använda deras datorer till testkörningar av min algoritm.

Jag vill även tacka min familj och mina vänner för allt stöd.

Referenser

[1] E. Anderson, Z. Bai, C. Bischof. J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenny, S. Ostrouchov and D. Sorensen. *LAPACK User's Guide.* Third Edition. SIAM Publications, 1999.

[2] Z. Bai och J.W. Demmel. On swapping diagonal blocks in real Schur form. *Linear Algebra Appl.*, 186:73-95, 1993.

[3] G. H. Golub och C. F. Van Loan *Matrix Computations.* Johns Hopkins University Press Baltimore, MD, third edition. 1996

[4] R. Granat och B. Kågström Parallel eigenvalue reordering in real Schur form. *In progress*

[5] I. Jonsson and B. Kågström. Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part I: One-Sided and Coupled Sylvester-Type Equations, *ACM Trans. Math. Software*, Vol. 28, No. 4, pp 393–415, 2002.

[6] I. Jonsson and B. Kågström. Recursive Blocked Algorithms for Solving Triangular Matrix Equations—Part II: Two-Sided and Generalized Sylvester and Lyapunov Equations, *ACM Trans. Math. Software*, Vol. 28, No. 4, pp 416–435, 2002.

[7] D. Kressner *Block algorithms for reordering standard and generalized Schur forms. ACM Transactions on Mathematical Software*, Vol. 32, No 3, September 2005, Pages 1-12.

[8] LAPACK - Linear Algebra Package. Se *http://www.netlib.org/lapack/.*

[9] RECSY- High Performance Library for Sylvester type Matrix Equations. Se *http://www.cs.umu.se/∼isak/recsy.html.*

Bilaga A

Algoritmbeskrivning

Den här algoritmen beskriver i detalj hur vi flyttar egenvärderna över ”gränsen” i `RDTRSEB` (Algoritm 5).

Algoritm 6 Algoritm som flyttar egenvärden över ”gränsen” i algoritm 5

% om det inte finns några egenvärden som ska flyttas
if $m_2 = 0 || m_1 = k$ **then**
  $m = m_1 + m_2$;
  return;
  % om delningspunkten är välidgt nära den ”gamla” delningspunkten
else if $k - m_1 = m_2 \;||\; k - m_1 = m_2 - 1 \;||\; k - m_1 - 1 = m_2 \;||\; k - m_1 - 2 = m_2 \;||\; k - m_1 = m_2 - 2$
then
  % om det är färre egenvärden som ska flyttas än vad det finns platser i $T_1 1$
  if $m_2 \leq k - m_1$ **then**
    if $k - m2 > 0$ *och* $T(k - m_2{+}1, k - m_2) \neq 0$ **then**
      $s_2 = k - m_2 - 1$;
    else
      $s_2 = k - m_2$;
    end if
    % Flyttar först egenvärderna precis över gränsen
    $[T(s_2{+}1{:}\,k + m_2, s_2{+}1{:}\,k + m_2), Z, select(s_2{+}1{:}\,k + m_2)] =$ **dtrbse**$(select(s_2{+}1{:}\,k + m_2),$
    $T(s_2{+}1{:}\,k + m_2,\ s_2{+}1{:}\,k + m_2,\ I_{k-s_2+m_2})$;
    $$T = \begin{bmatrix} T(1{:}\,s_2, 1{:}\,s_2) & T(1{:}\,s_2, s_2{+}1{:}\,k + m_2)Z & T(1{:}\,s_2, k + m_2{+}1{:}\,M) \\ T(s_2{+}1{:}\,k + m_2, 1{:}\,s_2) & T(s_2{+}1{:}\,k + m_2, s_2{+}1{:}\,k + m_2) & Z^T T(s_2{+}1{:}\,k + m_2, k + m_2{+}1{:}\,M) \\ & T(k + m_2{+}1{:}\,M, 1{:}\,M) & \end{bmatrix};$$
    $Q = [Q(1{:}\,M, 1{:}\,s_2),\ Q(1{:}\,M, s_2{+}1{:}\,k + m_2)Z,\ Q(1{:}\,M, k + m_2{+}1{:}\,M)]$;
    % om det finns fler platser än egenvärden som flyttas flytta även egenvärderna den ”sista” biten i T_{11}
    if $m_2 \neq k - m_1$ **then**
      $[T(m_1{+}1{:}\,k, m_1{+}1{:}\,k),\ Z,\ select(m_1{+}1{:}\,k)] =$ **rdtrseb** $(T(m_1{+}1{:}\,k, m_1{+}1{:}\,k),\ I_{k-m_1},\ blks,$
      $select(m_1{+}1{:}\,k))$;
      $$T = \begin{bmatrix} [T(1{:}\,m_1, 1{:}\,m_1) & T(1{:}\,m_1, m_1{+}1{:}\,k)Z & T(1{:}\,m_1, k{+}1{:}\,M) \\ T(m_1{+}1{:}\,k, 1{:}\,m_1) & T(m_1{+}1{:}\,k, m_1{+}1{:}\,k) & Z^T T(m_1{+}1{:}\,k, k{+}1{:}\,M) \\ & T(k{+}1{:}\,M, 1{:}\,M) & \end{bmatrix};$$
      $Q = [Q(1{:}\,M, 1{:}\,m_1),\ Q(1{:}\,M, m_1{+}1{:}\,k)Z,\ Q(1{:}\,M, k{+}1{:}\,M)]$;
    end if
  else
    % det finns fler egenvärden som ska flyttas i T_{22} än det finns platser i T_{11}
    if $T(k + k - m_1{+}1, k + k - m_1) \neq 0$ **then**
      $s_2 = k + k - m_1{+}1;\ s_3 = k{+}1$;
    else
      $s_2 = k + k - m_1;\ s_3 = k$;
    end if
    % flytta endast så många som får plats
    $[T(m_1{+}1{:}\,s_2, m_1{+}1{:}\,s_2), Z, select] =$**dtrseb**$(select(m_1{+}1{:}\,s_2),\ T(m_1{+}1{:}\,s_2,\ m_1{+}1{:}\,s_2),\ I_{s_2-m_1})$;
    $$T = \begin{bmatrix} T(1{:}\,m_1, 1{:}\,m_1) & T(1{:}\,m_1, m_1{+}1{:}\,s_2)Z & T(1{:}\,m_1, s_2{+}1{:}\,M) \\ T(m_1{+}1{:}\,s_2, 1{:}\,m_1) & T(m_1{+}1{:}\,s_2, m_1{+}1{:}\,s_2) & Z^T T(m_1{+}1{:}\,s_2, s_2{+}1{:}\,M) \\ & T(s_2{+}1{:}\,M, 1{:}\,M) & \end{bmatrix};$$
    $Q = [Q(1{:}\,M, 1{:}\,m_1),\ Q(1{:}\,M, m_1{+}1{:}\,s_2)Z,\ Q(1{:}\,M, s_2{+}1{:}\,M)]$;
    %flytta de egenvärden som inte fick plats i T_{11}
    $[T(s_3{+}1{:}\,k + m_2, s_3{+}1{:}\,k + m_2),\ Z,\ select(s_3{+}1{:}\,k + m_2)] =$ **rdtrseb**$(T(s_3{+}1{:}\,k + m_2, s_3{+}1{:}\,k + m_2),$
    $I_{k+m_2-s_3},\ blks,\ select(s_3{+}1{:}\,k + m_2))$;
    $$T = \begin{bmatrix} T(1{:}\,s_3, 1{:}\,s_3) & T(1{:}\,s_3, s_3{+}1{:}\,k + m_2)Z & T(1{:}\,s_3, k + m_2{+}1{:}\,M) \\ T(s_3{+}1{:}\,k + m_2, 1{:}\,s_3) & T(s_3{+}1{:}\,k + m_2, s_3{+}1{:}\,k + m_2) & Z^T T(s_3{+}1{:}\,k + m_2, k + m_2{+}1{:}\,M) \\ & T(k + m_2{+}1{:}\,M, 1{:}\,M) & \end{bmatrix};$$
    $Q = [Q(1{:}\,M, 1{:}\,s_3),\ Q(1{:}\,M, s_3{+}1{:}\,k + m_2)Z,\ Q(1{:}\,M, k + m_2{+}1{:}\,M)$;
  end if
else
  %om delningspunkten inte är väldigt nära den ”gamla” delningspunkten flytta alla egenvärden rekursivt direkt över gränsen.
  $[T(m_1{+}1{:}\,k + m_2, m_1{+}1{:}\,k + m_2),\ Z,\ select(m_1{+}1{:}\,k + m_2),\ m_3] =$ **rdtrseb**$(select(m_1{+}1{:}\,k + m_2),$
  $T(m_1{+}1{:}\,k + m_2, m_1{+}1{:}\,k + m_2),\ I_{k+m_2-m_1})$;
  $T =$
  $$\begin{bmatrix} [T(1{:}\,m_1, 1{:}\,m_1) & T(1{:}\,m_1, m_1{+}1{:}\,k + m_2)Z & T(1{:}\,m_1, k + m_2{+}1{:}\,M) \\ T(m_1{+}1{:}\,k + m_2, 1{:}\,m_1) & T(m_1{+}1{:}\,k + m_2, m_1{+}1{:}\,k + m_2) & Z^T T(m_1{+}1{:}\,k + m_2, k + m_2{+}1{:}\,M) \\ & T(k + m_2{+}1{:}\,M, 1{:}\,M) & \end{bmatrix};$$
  $Q = \begin{bmatrix} Q(1{:}\,M, 1{:}\,m_1) & Q(1{:}\,M, m_1{+}1{:}\,k + m_2)Z & Q(1{:}\,M, k + m_2{+}1{:}\,M) \end{bmatrix}$;
end if
$m = m_1 + m_2$;

Bilaga B

Källkod

```
      RECURSIVE SUBROUTINE RDTRSEB( JOB, COMPQ, SELECT, N, NB, T, LDT,
     $                   Q, LDQ, ER, EI, M, S, SEP, WORK, LWORK, IWORK,
     $                   LIWORK, INFO )
*
*     -- LAPACK-style routine --
*     Dept. Computing Science Univ. of Umeå, Sweden
*     written by Maria Näsholm, 11/05, 2006.
*     Optimized version 1.
*
      IMPLICIT NONE
*
*     .. Scalar Arguments ..
      CHARACTER          COMPQ, JOB
      INTEGER            INFO, LDQ, LDT, LIWORK, LWORK, M, N, NB
      DOUBLE PRECISION   S, SEP
*     ..
*     .. Array Arguments ..
      LOGICAL            SELECT( * )
      INTEGER            IWORK( * )
      DOUBLE PRECISION   Q( LDQ, * ), T( LDT, * ), EI( * ), WORK( * ),
     $                   ER( * )
*     ..
*
*     Purpose
*     =======
*
*     RDTRSEB reorders the real Schur factorization of a real matrix
*     A = Q*T*Q**T, so that a selected cluster of eigenvalues appears in
*     the leading diagonal blocks of the upper quasi-triangular matrix T,
*     and the leading columns of Q form an orthonormal basis of the
*     corresponding right invariant subspace.
*
*     Optionally the routine computes the reciprocal condition numbers of
*     the cluster of eigenvalues and/or the invariant subspace.
*
*     T must be in Schur canonical form (as returned by DHSEQR), that is,
*     block upper triangular with 1-by-1 and 2-by-2 diagonal blocks; each
*     2-by-2 diagonal block has its diagonal elemnts equal and its
*     off-diagonal elements of opposite sign.
*
*     This subroutine uses a delay and accumulate procedure for performing
*     all orthogonal updates in level 3 operations.
```

```
*
*     Arguments
*     =========
*
*     JOB     (input) CHARACTER*1
*     Specifies whether condition numbers are required for the
*     cluster of eigenvalues (S) or the invariant subspace (SEP):
*     = 'N': none;
*     = 'E': for eigenvalues only (S);
*     = 'V': for invariant subspace only (SEP);
*     = 'B': for both eigenvalues and invariant subspace (S and
*     SEP).
*
*     COMPQ   (input) CHARACTER*1
*     = 'V': update the matrix Q of Schur vectors;
*     = 'N': do not update Q.
*
*     SELECT  (input/output) LOGICAL array, dimension (N)
*     SELECT specifies the eigenvalues in the selected cluster. To
*     select a real eigenvalue w(j), SELECT(j) must be set to
*     .TRUE.. To select a complex conjugate pair of eigenvalues
*     w(j) and w(j+1), corresponding to a 2-by-2 diagonal block,
*     either SELECT(j) or SELECT(j+1) or both must be set to
*     .TRUE.; a complex conjugate pair of eigenvalues must be
*     either both included in the cluster or both excluded.
*     On output the (partial) reordering is displayed.
*
*     N       (input) INTEGER
*     The order of the matrix T. N >= 0.
*
*     NB      (input) INTEGER
*     The block size to use to delay and accumulate the orthogonal
*     updates. 3 <= NB <= N.
*
*     T       (input/output) DOUBLE PRECISION array, dimension (LDT,N)
*     On entry, the upper quasi-triangular matrix T, in Schur
*     canonical form.
*     On exit, T is overERitten by the reordered matrix T, again in
*     Schur canonical form, EIth the selected eigenvalues in the
*     leading diagonal blocks.
*
*     LDT     (input) INTEGER
*     The leading dimension of the array T. LDT >= max(1,N).
*
*     Q       (input/output) DOUBLE PRECISION array, dimension (LDQ,N)
*     On entry, if COMPQ = 'V', the matrix Q of Schur vectors.
*     On exit, if COMPQ = 'V', Q has been postmultiplied by the
*     orthogonal transformation matrix which reorders T; the
*     leading M columns of Q form an orthonormal basis for the
*     specified invariant subspace.
*     If COMPQ = 'N', Q is not referenced.
*
*     LDQ     (input) INTEGER
*     The leading dimension of the array Q.
*     LDQ >= 1; and if COMPQ = 'V', LDQ >= N.
*
*     ER      (output) DOUBLE PRECISION array, dimension (N)
*     EI      (output) DOUBLE PRECISION array, dimension (N)
*     The real and imaginary parts, respectively, of the reordered
*     eigenvalues of T. The eigenvalues are stored in the same
*     order as on the diagonal of T, EIth ER(i) = T(i,i) and, if
*     T(i:i+1,i:i+1) is a 2-by-2 diagonal block, EI(i) > 0 and
```

```
*     EI(i+1) = -EI(i). Note that if a complex eigenvalue is
*     sufficiently ill-conditioned, then its value may differ
*     significantly from its value before reordering.
*
*     M       (output) INTEGER
*     The dimension of the specified invariant subspace.
*     0 < = M <= N.
*
*     S       (output) DOUBLE PRECISION
*     If JOB = 'E' or 'B', S is a lower bound on the reciprocal
*     condition number for the selected cluster of eigenvalues.
*     S cannot underestimate the true reciprocal condition number
*     by more than a factor of sqrt(N). If M = 0 or N, S = 1.
*     If JOB = 'N' or 'V', S is not referenced.
*
*     SEP     (output) DOUBLE PRECISION
*     If JOB = 'V' or 'B', SEP is the estimated reciprocal
*     condition number of the specified invariant subspace. If
*     M = 0 or N, SEP = norm(T).
*     If JOB = 'N' or 'E', SEP is not referenced.
*
*     WORK    (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
*     On exit, if INFO = 0, WORK(1) returns the optimal LWORK.
*
*     LWORK   (input) INTEGER
*     The dimension of the array WORK.
*     If JOB = 'N', LWORK >= max(1,N);
*     if JOB = 'E', LWORK >= M*(N-M);
*     if JOB = 'V' or 'B', LWORK >= 2*M*(N-M).
*
*     If LWORK = -1, then a workspace query is assumed; the routine
*     only calculates the optimal size of the WORK array, returns
*     this value as the first entry of the WORK array, and no error
*     message related to LWORK is issued by XERBLA.
*
*     IWORK   (workspace) INTEGER array, dimension (LIWORK)
*     IF JOB = 'N' or 'E', IWORK is not referenced.
*
*     LIWORK  (input) INTEGER
*     The dimension of the array IWORK.
*     If JOB = 'N' or 'E', LIWORK >= 1;
*     if JOB = 'V' or 'B', LIWORK >= M*(N-M).
*
*     If LIWORK = -1, then a workspace query is assumed; the
*     routine only calculates the optimal size of the IWORK array,
*     returns this value as the first entry of the IWORK array, and
*     no error message related to LIWORK is issued by XERBLA.
*
*     INFO    (output) INTEGER
*     = 0: successful exit
*     < 0: if INFO = -i, the i-th argument had an illegal value
*     = 1: reordering of T failed because some eigenvalues are too
*     close to separate (the problem is very ill-conditioned);
*     T may have been partially reordered, and ER and EI
*     contain the eigenvalues in the same order as in T; S and
*     SEP (if requested) are set to zero.
*
*     Further Details
*     ===============
*
*     RDTRSEB first collects the selected eigenvalues by computing an
*     orthogonal transformation Z to move them to the top left corner of T.
```

```
*     In other words, the selected eigenvalues are the eigenvalues of T11
*     in:
*
*     Z'*T*Z = ( T11 T12 ) n1
*              (  0  T21 ) n2
*                 n1  n2
*
*     where N = n1+n2 and Z' means the transpose of Z. The first n1 columns
*     of Z span the specified invariant subspace of T.
*
*     If T has been obtained from the real Schur factorization of a matrix
*     A = Q*T*Q', then the reordered real Schur factorization of A is given
*     by A = (Q*Z)*(Z'*T*Z)*(Q*Z)', and the first n1 columns of Q*Z span
*     the corresponding invariant subspace of A.
*
*     The reciprocal condition number of the average of the eigenvalues of
*     T11 may be returned in S. S lies between 0 (very badly conditioned)
*     and 1 (very well conditioned). It is computed as follows. First we
*     compute R so that
*
*     P = ( I  R ) n1
*         ( 0  0 ) n2
*           n1 n2
*
*     is the projector on the invariant subspace associated EIth T11.
*     R is the solution of the Sylvester equation:
*
*     T11*R - R*T21 = T12.
*
*     Let F-norm(M) denote the Frobenius-norm of M and 2-norm(M) denote
*     the two-norm of M. Then S is computed as the lower bound
*
*     (1 + F-norm(R)**2)**(-1/2)
*
*     on the reciprocal of 2-norm(P), the true reciprocal condition number.
*     S cannot underestimate 1 / 2-norm(P) by more than a factor of
*     sqrt(N).
*
*     An approximate error bound for the computed average of the
*     eigenvalues of T11 is
*
*     EPS * norm(T) / S
*
*     where EPS is the machine precision.
*
*     The reciprocal condition number of the right invariant subspace
*     spanned by the first n1 columns of Z (or of Q*Z) is returned in SEP.
*     SEP is defined as the separation of T11 and T21:
*
*     sep( T11, T21 ) = sigma-min( C )
*
*     where sigma-min(C) is the smallest singular value of the
*     n1*n2-by-n1*n2 matrix
*
*     C  = kprod( I(n2), T11 ) - kprod( transpose(T21), I(n1) )
*
*     I(m) is an m by m identity matrix, and kprod denotes the Kronecker
*     product. We estimate sigma-min(C) by the reciprocal of an estimate of
*     the 1-norm of inverse(C). The true reciprocal 1-norm of inverse(C)
*     cannot differ from sigma-min(C) by more than a factor of sqrt(n1*n2).
*
*     When SEP is small, small changes in T can cause large changes in
```

```
*     the invariant subspace. An approximate bound on the maximum angular
*     error in the computed right invariant subspace is
*
*     EPS * norm(T) / SEP
*
*     ======================================================================
*
      LOGICAL            FIRST
      SAVE               FIRST
      DATA               FIRST /.TRUE./

*     .. Parameters ..
      INTEGER            MMULT
      DOUBLE PRECISION   ZERO, ONE
      PARAMETER          ( MMULT = 30, ZERO = 0.0D+0,
     $                     ONE = 1.0D+0 )
*     ..
*     .. Local Scalars ..
      LOGICAL            LQUERY, PAIR, SWAP, WANTBH, WANTQ, WANTS,
     $                   WANTSP, SPLIT, SWP2B2, SW1BS2, SW2BS2,
     $                   SWP1B1, SWP2B1, SWP1B2, SW2SB2, SW2SB1,
     $                   UPDATE, DIDSWP, FXSPLT, DVCLSTR
      INTEGER            IERR, K, KASE, KK, KS, LIWMIN, LWMIN, N1, N2,
     $                   NN, EBLOCK, SBLOCK, KKS, J, IT, KKK, LLL, DB,
     $                   LSTBLK, FSTBLK, DIM, DIM1, DIM2, KSL, KKL, II,
     $                   INDX, MLOC, I, FSTIND, LSTIND, LSTLOC, SELIND,
     $                   ROWS, COLS, CLSUM, ITT, DIFF, CLSUM2, SLAB
      DOUBLE PRECISION   EST, RNORM, SCALE, SEPLOC, SLOC, ELEM
      INTEGER            Z1, Z2, Z, SUMMA, WRK
      INTEGER            LDZ, LDZ1, LDZ2, S2, S3, M1, M2, M3
*     ..
*     .. External Functions ..
      LOGICAL            LSAME, REORDR
      INTEGER            LSUM
      DOUBLE PRECISION   DLANGE
      EXTERNAL           LSAME, DLANGE
*     ..
*     .. External Subroutines ..
      EXTERNAL           DLACON, DLACPY, DTRSYL, XERBLA,
     $                   RECSY_GEMM, DGEMM, ILACPY, DTRBSE, RECSYCT
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          ABS, MAX, SQRT, LOG, SIZE
*     ..
*     .. Local Functions ..
      INTEGER            ICEIL
      ICEIL( N, NB ) = (N + NB - 1) / NB
*     ..
*     .. Executable Statements ..
*
*     Decode and test the input parameters
*
      IF( FIRST ) THEN
         WANTBH = LSAME( JOB, 'B' )
         WANTS = LSAME( JOB, 'E' ) .OR. WANTBH
         WANTSP = LSAME( JOB, 'V' ) .OR. WANTBH
         WANTQ = LSAME( COMPQ, 'V' )
*
         INFO = 0
         LQUERY = ( LWORK.EQ.-1 )
         IF( .NOT.LSAME( JOB, 'N' ) .AND. .NOT.WANTS .AND. .NOT.WANTSP )
     $          THEN
```

```
            INFO = -1
         ELSE IF( .NOT.LSAME( COMPQ, 'N' ) .AND. .NOT.WANTQ ) THEN
            INFO = -2
         ELSE IF( N.LT.0 ) THEN
            INFO = -4
         ELSE IF( NB.LT.3) THEN
            INFO = -5
         ELSE IF( LDT.LT.MAX( 1, N ) ) THEN
            INFO = -7
         ELSE IF( LDQ.LT.1 .OR. ( WANTQ .AND. LDQ.LT.N ) ) THEN
            INFO = -9
         ELSE
*
*     Set M to the dimension of the specified invariant subspace,
*     and test LWORK and LIWORK.
*
            M = 0
            PAIR = .FALSE.
            DO 10 K = 1, N
               IF( PAIR ) THEN
                  PAIR = .FALSE.
               ELSE
                  IF( K.LT.N ) THEN
                     IF( T( K+1, K ).EQ.ZERO ) THEN
                        IF( SELECT( K ) )
     $                      M = M + 1
                     ELSE
                        PAIR = .TRUE.
                        IF( SELECT( K ) .OR. SELECT( K+1 ) )
     $                      M = M + 2
                     END IF
                  ELSE
                     IF( SELECT( N ) )
     $                   M = M + 1
                  END IF
               END IF
 10         CONTINUE
*
            N1 = M
            N2 = N - M
            NN = N1*N2
*
            SUMMA = 0
            DO 12 K = 1, FLOOR(LOG(REAL(N)))+1
               SUMMA = SUMMA +  1*(N/K)**2
12          CONTINUE
            IF( WANTSP ) THEN
               LWMIN = MAX( 1, SUMMA )
               LIWMIN = MAX( 1, SUMMA )
            ELSE IF( LSAME( JOB, 'N' ) ) THEN
               LWMIN = MAX( 1, N )
               LIWMIN = 1
            ELSE IF( LSAME( JOB, 'E' ) ) THEN
               LWMIN = MAX( 1, N )
               LIWMIN = 1
            END IF
*
            IF( LWORK.LT.LWMIN .AND. .NOT.LQUERY ) THEN
               INFO = -16
            ELSE IF( LIWORK.LT.LIWMIN .AND. .NOT.LQUERY ) THEN
               INFO = -18
            END IF
```

```
         END IF
*
         IF( INFO.EQ.0 ) THEN
            WORK( 1 ) = LWMIN
            IWORK( 1 ) = LIWMIN
         END IF
*
         IF( INFO.NE.0 ) THEN
            CALL XERBLA( 'RDTRSEB', -INFO )
            RETURN
         ELSE IF( LQUERY ) THEN
            RETURN
         END IF
         FIRST =.FALSE.
      END IF
*
*     Quick return if possible.
*
         M = 0
         DO 15 K = 1, N
            IF(SELECT(K)) THEN
               M = M + 1
            END IF
15       CONTINUE
         IF( M.EQ.N .OR. M.EQ.0 ) THEN
            IF(M.EQ.0) THEN
            END IF
            IF( WANTS )
     $           S = ONE
            IF( WANTSP )
     $           SEP = DLANGE( '1', N, N, T, LDT, WORK )
            GO TO 90
         END IF
*
*     If the size of T is greater then the given size NB
*
      IF( N.GT.NB ) THEN
*
*        Set the splitting point for T so that point dont split any
*        2 x 2 block
*
         IF( T(FLOOR(REAL(N/2))+1,FLOOR(REAL(N/2))).NE.ZERO ) THEN
              K = FLOOR(REAL(N/2+1))
         ELSE
              K = FLOOR(REAL(N/2))
         END IF
         Z = 1
         Z1 = 1
         LDZ1 = K
         LDZ2 = N-K
         Z2 = Z1 + K**2
         WRK = Z2 + (N-K)**2
*
*        Call rdtrseb recursivly for T11 and T22
*
         CALL DLASET('ALL', K, K, ZERO, ONE, WORK(Z1), LDZ1 )
         CALL RDTRSEB(JOB, COMPQ, SELECT(1), K, NB, T(1,1), LDT,
     $              WORK(Z1), LDZ1, ER, EI, M1, S, SEP, WORK(WRK),
     $              LWORK, IWORK, LIWORK, INFO )
         IF( INFO.EQ.1 ) THEN
            RETURN
         END IF
```

```
         CALL DLASET('ALL', N-K, N-K, ZERO, ONE, WORK(Z2), LDZ2 )
         CALL RDTRSEB(JOB, COMPQ, SELECT(K+1), N-K, NB,
     $               T(K+1,K+1), LDT, WORK(Z2), LDZ2, ER, EI, M2,
     $               S, SEP, WORK(WRK), LWORK, IWORK, LIWORK, INFO )
         IF( INFO.EQ.1 ) THEN
            RETURN
         END IF
*
*        Update T and Q
*
         CALL RECSY_GEMM( 1, K, N-K, K, ONE, WORK(Z1), LDZ1,
     $           T(1, K+1), LDT, ZERO, WORK(WRK), N, 0 )
         CALL RECSY_GEMM( 0, K, N-K, N-K, ONE, WORK(WRK), N,
     $           WORK(Z2), LDZ2, ZERO, T(1, K+1), LDT, 0 )
         CALL RECSY_GEMM(0, N, K, K, ONE, Q(1,1), LDQ,
     $           WORK(Z1), LDZ1, ZERO, WORK(WRK), N, 0 )
         CALL DLACPY('All', N, K, WORK(WRK), N, Q(1, 1), LDQ )
         CALL RECSY_GEMM(0, N, N-K, N-K, ONE, Q(1, K+1), LDQ,
     $           WORK(Z2), LDZ2, ZERO, WORK(WRK), N, 0 )
         CALL DLACPY('All', N, N-K, WORK(WRK), N, Q(1, K+1), LDQ )
*
*        Move the selected eigenvalues in T22 over the border to T11
*
*        Quick return if possible (none eigenvalues to move)
         IF(M2.EQ.ZERO .OR. M1.EQ.K ) THEN
            M = M1+M2
            GOTO 90
*
*        If the splitting point is close to the "old" splitting
*        point
*
*
*        If there are less eigenvalues to move from T22 then places
*        to move to in T11
         ELSE IF( K-M1.EQ.M2 .OR. K-M1.EQ.M2-1 .OR. K-M1-1.EQ.M2 .OR.
     $           K-M1-2.EQ.M2 .OR. K-M1.EQ.M2-2 ) THEN
*           Move the eigenvalues from T22 just over the border to
*           T11
            IF(M2.LE.K-M1 )THEN
               IF(K-M2.GT.ZERO .AND. T(K-M2+1, K-M2).NE.ZERO )THEN
                  S2=K-M2-1
               ELSE
                  S2=K-M2
               END IF

               LDZ = K+M2-S2
               Z = 1
               WRK = Z + LDZ**2
               CALL DLASET('All', K+M2-S2, K+M2-S2, ZERO, ONE,
     $                     WORK(Z), LDZ )
               CALL DTRSEB(JOB, COMPQ, SELECT(S2+1), K+M2-S2,
     $              MIN(NB, K+M2-S2),
     $              T(S2+1, S2+1), LDT, WORK(Z), LDZ, ER, EI, M3,
     $              S, SEP, WORK(WRK), LWORK, IWORK, LIWORK, INFO )

               IF( INFO.EQ.1 ) THEN
                  RETURN
               END IF
*              Update T and Q
               CALL RECSY_GEMM(0, S2, K+M2-S2, K+M2-S2, ONE,
     $                   T(1, S2+1), LDT, WORK(Z), LDZ, ZERO,
     $                   WORK(WRK), N, 0 )
```

```
               CALL DLACPY('All', S2, K+M2-S2, WORK(WRK), N,
     $                     T(1, S2+1), LDT )
               CALL RECSY_GEMM(1, K+M2-S2, N-K-M2, K+M2-S2, ONE,
     $                    WORK(Z), LDZ, T(S2+1, K+M2+1), LDT,
     $                    ZERO, WORK(WRK), N, 0)
               CALL DLACPY('All', K+M2-S2, N-K-M2, WORK(WRK), N,
     $                     T(S2+1, K+M2+1), LDT )
               CALL RECSY_GEMM(0, N, K+M2-S2, K+M2-S2, ONE,
     $                    Q(1, S2+1), LDQ, WORK(Z), LDZ,
     $                    ZERO, WORK(WRK), N ,0)
               CALL DLACPY('All', N, K+M2-S2, WORK(WRK), N,
     $                     Q(1, S2+1), LDQ )
*
*              Reorder values in T11
*
               IF(M2.NE.K-M1 ) THEN
                  LDZ = K-M1
                  WRK = Z + LDZ**2
                  CALL DLASET('All', K-M1, K-M1, ZERO, ONE,
     $                        WORK(Z), LDZ )
                  CALL RDTRSEB(JOB, COMPQ, SELECT(M1+1), K-M1, NB,
     $                        T(M1+1, M1+1), LDT, WORK(Z), LDZ, ER,
     $                        EI, M3, S, SEP, WORK(WRK), LWORK, IWORK,
     $                        LIWORK, INFO)
                  IF( INFO.EQ.1 ) THEN
                     RETURN
                  END IF
*                 Update T and Q
                  CALL RECSY_GEMM(0, M1, K-M1, K-M1, ONE,
     $                       T(1, M1+1), LDT, WORK(Z), LDZ, ZERO,
     $                       WORK(WRK), N, 0 )
                  CALL DLACPY('All', M1, K-M1, WORK(WRK), N,
     $                        T(1, M1+1), LDT )
                  CALL RECSY_GEMM(1, K-M1, N-K, K-M1, ONE,
     $                       WORK(Z), LDZ, T(M1+1, K+1), LDT, ZERO,
     $                       WORK(WRK), N, 0 )
                  CALL DLACPY('All', K-M1, N-K, WORK(WRK), N,
     $                        T(M1+1, K+1), LDT )
                  CALL RECSY_GEMM(0, N, K-M1, K-M1, ONE,
     $                       Q(1, M1+1), LDQ, WORK(Z), LDZ, ZERO,
     $                       WORK(WRK), N, 0 )
                  CALL DLACPY('All', N, K-M1, WORK(WRK), N,
     $                        Q(1, M1+1), LDQ )

                  END IF
*
*           Don't move more values from T22 then necessary over the
*           border to T11
*
            ELSE IF (M2 .GT. K-M1 ) THEN
               IF(T(K+K-M1+1, K+K-M1).NE.ZERO )THEN
                   S2 = K+K-M1+1
                   S3 = K+1
               ELSE
                   S2=K+K-M1
                   S3 = K
               END IF
               LDZ = N
               Z = 1
               WRK = Z + LDZ**2
               CALL DLASET('All', S2-M1, S2-M1, ZERO, ONE,
     $                     WORK(Z), LDZ )
```

```
               CALL DTRSEB(JOB, COMPQ, SELECT(M1+1), S2-M1, !M2??
     $                     MIN(NB, S2-M1),
     $                     T(M1+1, M1+1), LDT, WORK(Z), LDZ, ER, EI,
     $                     M3, S, SEP, WORK(WRK), LWORK, IWORK,
     $                     LIWORK, INFO )
               IF( INFO.EQ.1 ) THEN
                  RETURN
               END IF
*              Update T and Q
               CALL RECSY_GEMM(0, M1, S2-M1, S2-M1, ONE,
     $                    T(1, M1+1), LDT, WORK(Z), LDZ, ZERO,
     $                    WORK(WRK), N, 0 )
               CALL DLACPY('All', M1, S2-M1, WORK(WRK), N,
     $                     T(1, M1+1), LDT )
               CALL RECSY_GEMM(1, S2-M1, N-S2, S2-M1, ONE,
     $                      WORK(Z), LDZ, T(M1+1, S2+1), LDT, ZERO,
     $                      WORK(WRK), N, 0 )
               CALL DLACPY('All', S2-M1, N-S2, WORK(WRK), N,
     $                     T(M1+1, S2+1), LDT )
               CALL RECSY_GEMM(0, N, S2-M1, S2-M1, ONE,
     $                     Q(1, M1+1), LDQ, WORK(Z), LDZ, ZERO,
     $                     WORK(WRK), N, 0)
               CALL DLACPY('All', N, S2-M1, WORK(WRK), N,
     $                     Q(1, M1+1), LDQ )
               LDZ = K+M2-S3
               Z = 1
               WRK = Z + LDZ**2
               CALL DLASET('All', K+M2-S3, K+M2-S3, ZERO, ONE,
     $                     WORK(Z), LDZ )
*              Reorder values in T22 again
               CALL RDTRSEB(JOB, COMPQ, SELECT(S3+1), K+M2-S3, NB,
     $                      T(S3+1, S3+1), LDT, WORK(Z), LDZ, ER, EI,
     $                      M3, S, SEP, WORK(WRK), LWORK, IWORK,
     $                      LIWORK, INFO )
               IF( INFO.EQ.1 ) THEN
                  RETURN
               END IF
*              Update T and Q
               CALL RECSY_GEMM(0, S3, K+M2-S3, K+M2-S3, ONE,
     $                    T(1, S3+1), LDT, WORK(Z), LDZ, ZERO,
     $                    WORK(WRK), N, 0 )
               CALL DLACPY('All', S3, K+M2-S3, WORK(WRK), N,
     $                     T(1, S3+1), LDT )
               CALL RECSY_GEMM(1, K+M2-S3, N-K-M2, K+M2-S3, ONE,
     $                     WORK(Z), LDZ, T(S3+1, K+M2+1), LDT, ZERO,
     $                     WORK(WRK) , N, 0 )
               CALL DLACPY('All', K+M2-S3, N-K-M2, WORK(WRK), N,
     $                     T(S3+1, K+M2+1), LDT )
               CALL RECSY_GEMM(0, N, K+M2-S3, K+M2-S3, ONE,
     $                    Q(1, S3+1), LDQ, WORK(Z), LDZ, ZERO,
     $                    WORK(WRK), N, 0 )
               CALL DLACPY('All', N, K+M2-S3, WORK(WRK), N,
     $                     Q(1, S3+1), LDQ )
            END IF
*
*       If the splitting point is sufficiently away from the "old" splitting
*       point move selected values over the border recursively
*
        ELSE
            LDZ = K+M2-M1
            WRK = Z + LDZ**2
            CALL DLASET('All', K+M2-M1, K+M2-M1, ZERO, ONE,
```

```
     $                   WORK(Z), LDZ )
           CALL RDTRSEB(JOB, COMPQ, SELECT(M1+1), K+M2-M1, NB,
     $          T(M1+1, M1+1), LDT, WORK(Z), LDZ, ER, EI, M3, S,
     $          SEP, WORK(WRK), LWORK, IWORK, LIWORK, INFO )
           IF( INFO.EQ.1 ) THEN
              RETURN
           END IF
*          Update T and Q
           CALL RECSY_GEMM(0, M1, K+M2-M1, K+M2-M1, ONE,
     $                T(1, M1+1), LDT, WORK(Z), LDZ, ZERO,
     $                WORK(WRK), N, 0 )
           CALL DLACPY('All', M1, K+M2-M1, WORK(WRK), N,
     $                     T(1, M1+1), LDT )
           CALL RECSY_GEMM(1, K+M2-M1, N-K-M2, K+M2-M1, ONE,
     $                WORK(Z), LDZ, T(M1+1, K+M2+1), LDT, ZERO,
     $                WORK(WRK), N, 0 )
           CALL DLACPY('All', K+M2-M1, N-K-M2, WORK(WRK), N,
     $                 T(M1+1, K+M2+1), LDT )
           CALL RECSY_GEMM(0, N, K+M2-M1, K+M2-M1, ONE,
     $                 Q(1, M1+1), LDQ, WORK(Z), LDZ, ZERO,
     $                 WORK(WRK), N, 0 )
           CALL DLACPY('All', N, K+M2-M1, WORK(WRK), N,
     $                 Q(1, M1+1), LDQ )
        END IF
        M = M1 +M2
*
*     basecase if NB < N
*
      ELSE
        WANTQ = LSAME( COMPQ, 'V' )
         CALL DTRSEB(JOB, COMPQ, SELECT, N, MIN(NB, N),
     $            T, LDT, Q, LDQ, ER, EI, M,
     $            S, SEP, WORK(1), LWORK, IWORK, LIWORK, INFO )
        IF( INFO.EQ.1 ) THEN
           RETURN
        END IF
      END IF
*
*
       N1 = M
       N2 = N - M
       NN = N1*N2
*
      IF( WANTS ) THEN
*
*     Solve Sylvester equation for R:
*
*     T11*R - R*T21 = scale*T12
*
         CALL DLACPY( 'F', N1, N2, T( 1, N1+1 ), LDT, WORK(1), N1 )
         CALL RECSYCT( 1, SCALE, N1, N2, T, LDT, T( N1+1, N1+1 ),
     $        LDT, WORK(1), N1, INFO )
*
*     Estimate the reciprocal of the condition number of the cluster
*     of eigenvalues.
*
         RNORM = DLANGE( 'F', N1, N2, WORK(1), N1, WORK )
         IF( RNORM.EQ.ZERO ) THEN
            S = ONE
         ELSE
            S = SCALE / ( SQRT( SCALE*SCALE / RNORM+RNORM )*
     $           SQRT( RNORM ) )
```

```
        END IF
      END IF
*
      IF( WANTSP ) THEN
*
*     Estimate sep(T11,T21).
*
         EST = ZERO
         KASE = 0
  80     CONTINUE
         CALL DLACON( NN, WORK( NN+1 ), WORK(1), IWORK, EST, KASE )
         IF( KASE.NE.0 ) THEN
            IF( KASE.EQ.1 ) THEN
*
*     Solve  T11*R - R*T21 = scale*X.
*
               CALL RECSYCT( 1, SCALE, N1, N2, T, LDT,
     $              T( N1+1, N1+1 ), LDT, WORK(1), N1, INFO )
            ELSE
*
*     Solve  T11'*R - R*T21' = scale*X.
*
               CALL RECSYCT( 7, SCALE, N1, N2, T, LDT,
     $              T( N1+1, N1+1 ), LDT, WORK(1), N1, INFO )
            END IF
            GO TO 80
         END IF
*
         SEP = SCALE / EST
      END IF
*
  90  CONTINUE
*
*     Store the output eigenvalues in ER and EI.
*
      DO 100 K = 1, N
         ER( K ) = T( K, K )
         EI( K ) = ZERO
  100  CONTINUE
      DO 110 K = 1, N - 1
         IF( T( K+1, K ).NE.ZERO ) THEN
            EI( K ) = SQRT( ABS( T( K, K+1 ) ) )*
     $           SQRT( ABS( T( K+1, K ) ) )
            EI( K+1 ) = -EI( K )
         END IF
  110 CONTINUE
*
      WORK( 1 ) = LWMIN
      IWORK( 1 ) = LIWMIN
*
      RETURN
*
*     End of RDTRSEB
*
      END
```