Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》



# 稠密对称矩阵特征值求解器的数学与算法

本文详细解析了目前常用的稠密对称矩阵特征值求解器使用的数学公式及算法。

如果页面内的数学公式未能正确渲染，请尝试刷新页面。

## 0. 简介

本文以 ELPA 所使用的数学公式和算法为基础。ELPA/ELPA2 是目前常用的稠密对称矩阵特征值求解器之一。ELPA 使用 2-stage 算法进行求解（实际上是 3-stage），即先将稠密对称矩阵变换为多对角对称矩阵 (sy2sb), 然后将对称多对角矩阵变换为对称三对角矩阵 (sb2st), 最后通过分治法求解对称三对角矩阵的所有特征值和特征向量。2-stage 指的是从原矩阵到对称三对角矩阵分了两步，与之相对的 1-stage 则是传统的一步到位从原矩阵变换为对称三对角矩阵。ELPA 和 SLATE 都采用了 2-stage 算法，而 ScaLAPACK 则采用了 1-stage 算法。

根据 ELPA 论文 [Marek2014, Auck2011], 其 sy2sb 和 sb2st 算法主要来自论文 [Bischof1994]. 然而 [Bischof1994] 并没有详细叙述 sy2sb 和 sb2st 算法，而是指向了一些别的论文。有些论文由于年代久远我找不到了，最后 sy2sb 算法参考了 [Bischof1994] 引用的 [Grimes1988], sb2st 算法主要参考了 [Bischof1994, Auck2011, Lang1993].

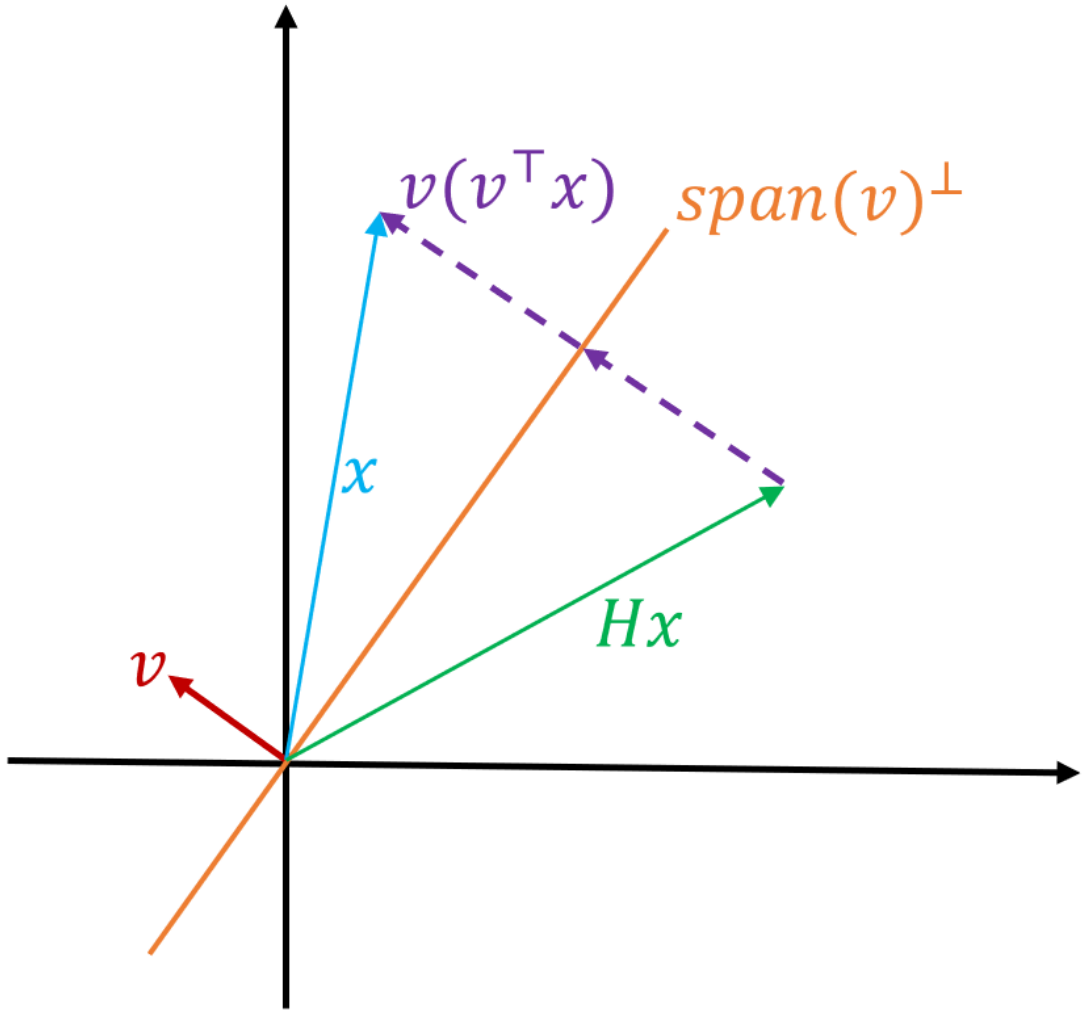
## 1. Householder 变换

大部分线性代数库的特征值求解器都基于以下原理：寻找一系列酉矩阵 $Q_1, Q_2, \cdots, Q_k$, 使得 $Q_1 Q_2 \cdots Q_k A Q_k^T \cdots Q_2^T Q_1^T = D$, $D$ 是包含所有特征值的对角矩阵，而正交矩阵的乘积 $V = Q_1 Q_2 \cdots Q_k$ 仍是正交矩阵。本文只考虑实对称矩阵和对应的正交矩阵。Householder 变换矩阵 $H(v) = I - 2vv^T/|v|_2$ 是最简单的正交矩阵之一。Householder 变换有一个简单的几何意义：$H(v)x$ 将向量 $x$ 反射到由法向量 $v$ 定义的超平面的另一侧。下图展示了这个几何变换，其中红色的 $v$ 是反射镜面的单位法向量，反射镜面是橙色的 $span(v)^{\perp}$：

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》






由于 Householder 是一个镜面变换，任意向量可以被一个 Householder 反射到一个指定的向量。利用这一特性，矩阵分解算法常用 Householder 变换将一个向量反射到一个特定向量。给定任意向量 $x$, 我们可以构造一个 Householder 变换 $H(\beta, v) = I - \beta vv^\top$ 使得 $H(\beta, v)x = y$, $y(1) = |x|_2$, $y(2:end) = 0$. 构造 $\beta, v$ 的数学推导和算法参见 [VL4], MATLAB 代码如下。

```
function [v, b] = house_vec(x)
% Householder Vector, Van Loan book 4th edition Algorithm 5.1.1
    m = length(x);
    if (m >= 2)
        s = x(2 : m)' * x(2 : m);
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
6           v = [1; x(2 : m)];
7       else
8           s = 0;
9           v = 1;
10      end
11      if ((s == 0) && (x(1) >= 0))
12          b = 0;
13      elseif ((s == 0) && (x(1) < 0))
14          b = 2;  % The book has a bug here
15      else
16          mu = sqrt(x(1) * x(1) + s);
17          if (x(1) <= 0)
18              v(1) = x(1) - mu;
19          else
20              v(1) = -s / (x(1) + mu);
21          end
22          v1_2 = v(1) * v(1);
23          b = 2 * v1_2 / (s + v1_2);
24          v = v / v(1);
25      end
26  end
```

这里有两个相关的性质在后面的算法会多次用到，值得注意：

- **性质1**：Householder vector $v$ 的第一个分量永远是 1.
- **性质2**：由矩阵乘法的定义，对于 Householder 矩阵 $H$ 和分块对角矩阵 $Q = blkdiag(I_{s-1}, H, I_{n-e})$，$A := Q^T A$ 将改变 $A(s:e,:)$，$A := AQ$ 将改变 $A(:,s:e)$.

另外第 14 行在 [VL4] 中的原文是 `b=-2`，这是错的，可以用一个简单的例子 $x = (-3,0,0)^T$ 验证。

在本节最后插播一则学术笑话庆祝艾尔海森即将上线：

- Householder 变换是一个镜面反射，艾尔海森的元素爆发技能也是镜面反射；
- Householder 变换把向量的能量都集中到向量的第一个分量，艾尔海森也让教令院所有的报告都集中送到他那里；
- Householder 的字面意思是有房者，艾尔海森也在须弥持有优质房产；
- Householder 变换和艾尔海森从未在原神中同台出现过。

综上所述，艾尔海森就是 Householder。

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

## 2. 基于 Householder 变换的 QR 分解

基于 Householder 变换的 QR 分解并不是本文主要关注的算法，但是后面其他算法需要用到 Householder QR, 而且对这个算法的理解有助于理解后面的算法。Householder QR 的思想是第 $k$ 步用一个 Householder 变换将矩阵第 $k$ 列对角线以下的元素变换为0，这样变换到最后一列的时候剩下的就是上三角矩阵 $R$, 所有的 Householder 矩阵依次乘起来得到正交矩阵 $Q$. 下面的算法来自 [VL4]:

```
1  function VR = house_qr(A)
2  % Householder QR, Van Loan book 4th edition Algorithm 5.2.1
3  % Output parameter:
4  %   VR : Same size as A, triu(VR(1:n, 1:n)) is R, tril(VR, -1) contains Householder vectors
5      [m, n] = size(A);
6      for j = 1 : n
7          [v, b] = house_vec(A(j : m, j));
8          t = v' * A(j : m, j : n);
9          A(j : m, j : n) = A(j : m, j : n) - b .* (v * t);
10         if (j < m), A(j+1 : m, j) = v(2 : m-j+1); end
11     end
12     VR = A;
13 end
```

LAPACK 中的 `dgeqr2` 使用的也是相同的算法和存储格式。在这个算法里，Householder vectors 存储于输入输出矩阵 $A$ 的严格下三角位置。由前文**性质1**，每个 Householder vector $v$ 的第一个分量可以不存，因此恰好不需要占用对角线的位置或者额外的存储空间。$b$ 虽然没有保存，但是可以由 $b = 2/|v|_2^2$ 还原出来。

注意，第 7 行生成的 $v$ 的长度并不是 $m$, 其对应的 Householder 矩阵只是一个完整的、可以和整个 $A$ 相乘的 Householder 矩阵的右下方对角块。由前文**性质2**, 这个完整的 Householder 矩阵左乘 $A$ 只会改变 $A(j:m,:)$.

Householder QR 没有显式生成正交矩阵 $Q$, 但是可以利用保存的 Householder vectors 直接计算 $Q$ 乘以任意矩阵 $X$. 下面的算法同样来自 [VL4]:

```
1  function Y = apply_house_Q_bwd(V, X)
2  % Backward accumulation of Householder matrices
3  % Van Loan book 4th edition Section 5.1.6
4  % Input parameters:
5  %   V : Size m * n, its strict lower part contains the Householder vectors
6  %   X : Input vectors, size m * k
7  % Output parameter:
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
8  %   Y : Y = Q1 * Q2 * ... * Qn * X
9      [m, n] = size(V);
10     Y = X;
11     for j = n : -1 : 1
12         v = [1; V(j+1 : m, j)];
13         b = 2 / (norm(v).^2);
14         % Y = Q_j * Y = (I - b * v * v') * Y
15         t = v' * Y(j : m, :);
16         Y(j : m, :) = Y(j : m, :) - (b .* v) * t;
17     end
18 end
```

这里说明一下，`_bwd` 这个后缀意思是逆序（backward）左乘 Householder 变换矩阵到 $X$，分解时的生成顺序是 $H_1, H_2, \cdots, H_k$, 但是左乘的顺序是 $H_k, H_{k-1}, \cdots, H_1$. 如果左乘顺序和分解顺序相同，那么由于 $H_i$ 都是方阵，每一次乘法都是两个方阵相乘。如果 $X$ 的列数小于 $n$, 逆序左乘比顺序左乘的复杂度更低。在这个算法里，如果输入的 `X = eye(size(A))`，那么输出的 `Y` 对应 `[Q, R] = qr(A, 0)` (thin QR) 里面的 `Q`.

Householder QR 也可以使用分块算法来将其原有的 BLAS-2 操作转换为 BLAS-3 操作以提高性能。`house_qr()` 和 `apply_house_Q_bwd()` 中主要操作都是矩阵向量乘法和 rank-1 update. 注意看 `house_qr()` 第 8, 9 行，假如给定一个整数 $b$, 那么第 9 行更新任何 $A(j:m, l), j < l < j + b$ 的时候，并不需要来自 $A(:, j+b:end)$ 的信息。也就是说，对 $A(:, j+b:end)$ 的更新可以被延迟，乃至进一步被合并。从数学上说，由于对 trailing matrix $A(j:m, j:n)$ 的更新都是 rank-1 update, $k$ 个 rank-1 update 相乘应该有一个等价的 rank-$k$ update. 因此我们可以合并若干个 Householder 矩阵得到一个正交矩阵 $I - WY^T$. 下面的算法给出了使用 WT representation 计算 Householder QR 的方法，同样来自 [VL4]:

```
1  function VR = house_qr_WY(A, bs)
2  % Blocked Householder QR using WY representation
3  % Van Loan 4th edition Algorithm 5.2.2
4  % Input parameters:
5  %   A  : Size m * n, m >= n
6  %   bs : Block size
7  % Output parameter:
8  %   VR : Same size as A, triu(VR(1:n, 1:n)) is R, tril(VR, -1) contains Householder vectors
9      [m, n] = size(A);
10     for j = 1 : bs : n
11         % Columns in the current panel: [j, k]
12         k = min(n, j + bs - 1);
13         curr_bs = k - j + 1;
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
        % QR factorize A(j : m, j : k), in practice the results overwrite A(j : m, j : k)
        A(j : m, j : k) = house_qr(A(j : m, j : k));
        % Build W and Y from V
        % We only store and use the (j : m)-th non-zeros rows in Y and W
        Y = tril(A(j : m, j : k), -1) + eye(m-j+1, curr_bs);
        W = gen_W_from_Y(Y);
        % Update columns right to the current panel
        % Q_j * Q_{j+1} * ... * Q_k = eye(m) - W * Y'
        % A(j : m, k+1 : n) = Q_k * ... * Q_{j+1} * Q_j * A(j : m, k+1 : n)
        % A(j : m, k+1 : n) = (eye(m) - Y * W') * A(j : m, k+1 : n)
        WTA = W' * A(j : m, k+1 : n);
        A(j : m, k+1 : n) = A(j : m, k+1 : n) - Y * WTA;
    end
    VR = A;
end
```

```
function W = gen_W_from_Y(Y)
% Generate the WY matrix from Householder vectors
% Van Loan 4th edition Algorithm 5.1.2
% Y : Size m * r, m > r, each column is a Householder vector v
% W : Size m * r, generated W matrix
    [m, r] = size(Y);
    b = 2 / (norm(Y(:, 1)).^2);
    W = zeros(m, r);
    W(:, 1) = b * Y(:, 1);
    for j = 2 : r
        vj = Y(:, j);
        b = 2 / (norm(vj).^2);
        W(:, j) = b * vj - b * W(:, 1 : j-1) * (Y(:, 1 : j-1)' * vj);
    end
end
```

对应地，用 WY representation 计算 $Q$ 左乘任意矩阵 $X$ 的算法如下：

```
function Z = apply_house_Q_WY_bwd(V, X, bs)
% Backward accumulation of Householder matrices using WY representation
% Input parameters:
%   V  : Size m * n, its strict lower part contains the Householder vectors
%   X  : Input vectors, size m * k
%   bs : Block size, default is 8
% Output parameter:
%   Z : Z = Q1 * Q2 * ... * Qn * X
    [m, n] = size(V);
    if (nargin < 3), bs = 8; end
    n1 = min(m-1, n);  % The last column of a square V has no Householder vector
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
    Z = X;
    for k = n1 : -bs : 1
        % Columns in the current panel: [j, k]
        j = max(1, k - bs + 1);
        curr_bs = k - j + 1;
        % Build W and Y from V
        % We only store and use the (j : m)-th non-zeros rows in Y and W
        Y = tril(V(j : m, j : k), -1) + eye(m-j+1, curr_bs);
        W = gen_W_from_Y(Y);
        % Q_j * Q_{j+1} * ... * Q_k = eye(m) - W * Y'
        % Z = Q_j * Q_{j+1} * ... * Q_k * Z = (eye(m) - W * Y') * Z
        YTZ = Y' * Z(j : m, :);
        Z(j : m, :) = Z(j : m, :) - W * YTZ;
    end
end
```

## 3. 基于对称 Householder 变换的三对角化

理解单向量版的 Householder QR 后，基于对称 Householer 变换的三对角化就不难理解了。我刚开始看三对角化的时候还有过一个幼稚的想法：为什么是三对角化？直接把 Householder QR 里面的 Householder 矩阵再右乘一下 $A$ 会怎么样？按了两行代码我就发现我的 naive 之处了：$H^TA$ 将 $A(2:end,1)$ 清零了，但是 $(H^TA)H$ 将 $(H^TA)(1,2:end)$ 清零了的同时又把 $(H^TA)(2:end,1)$ 填上了非零元。由于特征值分解总是需要同时在矩阵两边乘以正交矩阵及其转置，那么我们就要找一个正交矩阵使得它左乘和右乘 $A$ 的时候不能『先挖土后填土』。

**性质2**提醒我们，可以构造一个不满长度的 $v=(0,\cdots,0,1,a,b,c,\cdots)^T$, 这样其对应的 Householder 变换在左乘和右乘 $A$ 的时候不会改变 $A$ 的最上面几行和最左边几行，$H^TA$ or $HA$ 清零的列 or 行不会被 $(H^TA)H$ or $H^T(AH)$ 重新填入非零元。因此，我们可以固定一个带宽 $b=2\times bs+1$, 通过一系列对称 Housholder 变换将 $A$ 变成一个带宽为 $b$ 的带状矩阵 $B$. 这样的 Householder 变换每次都不会改变当前待处理矩阵的上面/左侧 $bs$ 行/列。

因为对称三对角矩阵的特殊性质，我们可以将 $A$ 变换成一个对称三对角矩阵。取更大的带宽的情况将在下一节讨论。下面的三对角化算法依然来自 [VL4]:

```
function [Q, T] = house_tridiag(A)
% Householder Tridiagonalization, Van Loan book 4th edition, Algorithm 8.3.1
% Output parameters:
%   Q : Product of Householder matrices
%   T : Tridiagonal matrix, A = Q * T * Q^T
    n = size(A, 1);
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
    Q = eye(n);
    for k = 1 : n-2
        [v, b] = house_vec(A(k+1 : n, k));
        % Q_k = eye(m) - b * v * v', apply Q_k' * A * Q_k
        p = b * A(k+1 : n, k+1 : n) * v;
        w = p - (0.5 * b * p' * v) * v;
        A(k+1, k) = norm(A(k+1 : n, k));
        A(k, k+1) = A(k+1, k);
        vwT = v * w';
        A(k+1 : n, k+1 : n) = A(k+1 : n, k+1 : n) - vwT - vwT';
        % v(2 : end) can be stored in A(k+2 : n, k)
        A(k+2 : n, k) = 0;
        A(k, k+2 : n) = 0;
        % Accumulate Q = Q * Q_k = Q - b * (Q * v) * v';
        t = Q(:, k+1 : n) * v;
        Q(:, k+1 : n) = Q(:, k+1 : n) - (b .* t) * v';
    end
    T = A;
end
```

值得注意的是，这里没有用类似前面几个算法的方式依次计算 $H^TA(k+1:n,k+1:n)$ 和 $[H^TA(k+1:n,k+1:n)]H$.
$(I-\beta vv^T)A(I-\beta vv^T)=A-\beta Avv^T-\beta vv^TA+\beta^2vv^TAvv^T$, 利用对称性可以引入中间变量 $p,w$, 使得最后只计算 $A-vw^T-wv^T$. 在这里我偷了一个懒，显式生成了 Q 矩阵。如果不显式生成，Householder vectors 同样可以存储在三对角以外的区域并且用 `apply_house_Q_bwd()` 生成 $Q$ 或者 apply $Q$.

到这里，很自然地我们可以问两个问题：

1. 能否像 Householder QR 的 WY 表示法一样合并几个变换，用 WY 表示法或者类似的方法来更新 trailing matrix?
2. 变换到三对角矩阵而不是带宽更大的矩阵是最好的选择吗?

这两个问题我们暂时放一放。既然我们得到了对称三对角矩阵，我们先看看如何快速求出其特征值和特征向量。

## 4. 对称三对角矩阵的特征值分治求解算法

这部分参见 [VL4] Section 8.4.3 8.4.4 和 [ECNU-MC] Section 5.4. 这里我偷懒一点，直接截两本书里的图：

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC



考虑不可约对称三对角矩阵

$$
\begin{aligned}
T &= \left[\begin{array}{ccccc|ccccc}
a_1 & b_1 & & & & & & & & \\
b_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & a_{m-1} & b_{m-1} & & & & & & \\
 & & b_{m-1} & a_m & b_m & & & & & \\
\hline
 & & & b_m & a_{m+1} & b_{m+1} & & & & \\
 & & & & b_{m+1} & \ddots & \ddots & & & \\
 & & & & & \ddots & \ddots & b_{n-1} & & \\
 & & & & & & b_{n-1} & a_n &
\end{array}\right] \\
&= \left[\begin{array}{ccccc|ccccc}
a_1 & b_1 & & & & & & & & \\
b_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & a_{m-1} & b_{m-1} & & & & & & \\
 & & b_{m-1} & a_m - b_m & & & & & & \\
\hline
 & & & & a_{m+1} - b_m & b_{m+1} & & & & \\
 & & & & b_{m+1} & \ddots & \ddots & & & \\
 & & & & & \ddots & \ddots & b_{n-1} & & \\
 & & & & & & b_{n-1} & a_n &
\end{array}\right] + \left[\begin{array}{ccccc|ccccc}
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & b_m & b_m & & & \\
\hline
 & & & & & b_m & b_m & & & \\
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & & & & &
\end{array}\right] \\
&= \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + b_m v v^{\mathsf{T}},
\end{aligned}
$$

其中 $v = [0, \ldots, 0, 1, 1, 0, \ldots, 0]^{\mathsf{T}}$. 假定 $T_1$ 和 $T_2$ 的特征值分解已经计算出来了, 即 $T_1 = Q_1 \Lambda_1 Q_1^{\mathsf{T}}$, $T_2 = Q_2 \Lambda_2 Q_2^{\mathsf{T}}$, 下面考虑 $T$ 的特征值分解.

Rainmaker's Notebook
『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

Now suppose that we have $m$-by-$m$ orthogonal matrices $Q_1$ and $Q_2$ such that $Q_1^T T_1 Q_1 = D_1$ and $Q_2^T T_2 Q_2 = D_2$ are each diagonal. If we set

$$U = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix},$$

then

$$U^T T U = U^T \left( \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + \rho v v^T \right) U = D + \rho z z^T$$

where

$$D = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}$$

is diagonal and

$$z = U^T v = \begin{bmatrix} Q_1^T e_m \\ \theta Q_2^T e_1 \end{bmatrix}.$$

Comparing these equations we see that the effective synthesis of the two half-sized Schur decompositions requires the quick and stable computation of an orthogonal $V$ such that

$$V^T (D + \rho z z^T) V = \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$$

which we discussed in §8.4.3.

**Theorem 8.4.3.** *Suppose $D = \operatorname{diag}(d_1, \ldots, d_n) \in \mathbb{R}^{n \times n}$ and that the diagonal entries satisfy $d_1 > \cdots > d_n$. Assume that $\rho \neq 0$ and that $z \in \mathbb{R}^n$ has no zero components. If $V \in \mathbb{R}^{n \times n}$ is orthogonal such that*

$$V^T (D + \rho z z^T) V = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$$

*with $\lambda_1 \geq \cdots \geq \lambda_n$ and $V = [\, v_1 \,|\, \cdots \,|\, v_n \,]$, then*

**(a)** *The $\lambda_i$ are the $n$ zeros of $f(\lambda) = 1 + \rho z^T (D - \lambda I)^{-1} z$.*

**(b)** *If $\rho > 0$, then $\lambda_1 > d_1 > \lambda_2 > \cdots > \lambda_n > d_n$.*
*If $\rho < 0$, then $d_1 > \lambda_1 > d_2 > \cdots > d_n > \lambda_n$.*

**(c)** *The eigenvector $v_i$ is a multiple of $(D - \lambda_i I)^{-1} z$.*

下面的代码基本是对上面图片内容的直接翻译：

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
function [V, D] = tridiag_eig_dc(T)
% Divide and conquer method, see Van Load 4th edition Section 8.4.3, 8.4.4
% T is a symmetric tridiagonal matrix, V * D * V' = T
    n = size(T, 1);
    if (n <= 4)
        [V, D] = eig(T);
        return;
    end
    % T = blkdiag(T1, T2) + rho * v * v';
    % v = [zeros(m-1, 1); 1; 1; zeros(n-m-1, 1)];
    m = floor(n / 2);
    rho = T(m+1, m);  % == beta_m
    T1 = T(1 : m, 1 : m);
    T1(m, m) = T1(m, m) - rho;
    [Q1, D1] = tridiag_eig_dc(T1);
    T2 = T(m+1 : n, m+1 : n);
    T2(1, 1) = T2(1, 1) - rho;
    [Q2, D2] = tridiag_eig_dc(T2);
    % U = blkdiag(Q1, Q2); D = blkdiag(D1, D2);
    % U' * T * U = U' * (blkdiag(T1, T2) + rho * v * v') * U = D + rho * z * z';
    % z = U' * v = [Q1(m, :), Q2(1, :)]';
    % Solve [V1, L1] = eig(D + rho * z * z')
    z  = [Q1(m, :), Q2(1, :)]';
    d0 = [diag(D1); diag(D2)];
    d  = sort(d0, 'descend');
    L1 = zeros(n);
    V1 = zeros(n);
    % Theorem 8.4.3
    % (a) l_i are the n zeros of 1 + rho * z' * inv(D - l * I) * z
    f = @(lambda)(1 + rho * sum( z .* z ./ (d0 - lambda)));
    for i = 1 : n
        if (rho > 0)
            % (b-1) l_1 > d_1 > l_2 > d_2 > ... > l_n > d_n
            d_i = d(i) + 10 * eps;
            if (i == 1)
                h = 1;
                d_im1 = d(1) + h;
                f_d1 = f(d(1) + 10 * eps);
                while (f(d_im1) * f_d1 > 0)
                    h = h * 2;
                    d_im1 = d(1) + h;
                end
            else
                d_im1 = d(i - 1) - 10 * eps;
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
45            end
46            l_i_range = [d_i, d_im1];
47        else
48            % (b-2) d_1 > l_1 > d_2 > l_2 > ... > d_n > l_n
49            d_i = d(i) - 10 * eps;
50            if (i == n)
51                h = 1;
52                d_ip1 = d(n) - h;
53                f_dn = f(d(n) - 10 * eps);
54                while (f(d_ip1) * f_dn > 0)
55                    h = h * 2;
56                    d_ip1 = d(n) - h;
57                end
58            else
59                d_ip1 = d(i + 1) + 10 * eps;
60            end
61            l_i_range = [d_ip1, d_i];
62        end
63        l_i = fzero(f, l_i_range);
64        L1(i, i) = l_i;
65        % (c) Eigenvector v_i is a multiple of inv(D - l_i * I) * z
66        v_i = z ./ (d0 - l_i);
67        V1(:, i) = v_i ./ norm(v_i);
68    end
69    % U' * T * U = V1 * L1 * V1';
70    % T = (U * V1) * L1 * (V1' * U');
71    V = [Q1 * V1(1 : m, :); Q2 * V1(m+1 : n, :)];
72    D = L1;
73 end
```

这个代码在数值上是不太稳定的。[ECNU-MC] 列出了一些现在使用的提高数值稳定性和加快计算的方法，这里略过。

## 5. 稠密对称到多对角矩阵变换 (sy2sb) 算法

第 3、4 节给了我们一个完整的特征值和特征向量求解器。然而，第 3 节最后还有两个问题尚未解答。实际上这两个问题是相互关联的。回顾一下 Householder QR 的分析：

> 给定一个整数 $b$, 那么第 9 行更新任何 $A(j:m,l), j<l<j+b$ 的时候，并不需要来自 $A(:,j+b:end)$ 的信息。也就是说，对 $A(:,j+b:end)$ 的更新可以被延迟。

Rainmaker's Notebook
『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home
Archives
About
SiteXC

对比一下 `house_tridiag()`. 上一节的分析中，$H^TAH$ 展开后的第四项 $\beta^2vv^TAvv^T$ 可以看作 $\beta^2v(v^TAv)v^T$, 这个矩阵的任一行和列都依赖于 $v^TAv$ 这个标量值，而这个标量的计算需要用到整个 $A$ 的信息。这意味着我们无法推迟更新 $A$ 的任何行或者列，也就无法合并任何 Householder 变换。这就是 1-stage 三对角化的缺点：只能用 BLAS-2 操作，无法使用 BLAS-3 操作，性能受限于内存带宽。

下面我们考察将 sy2sb. 选择半带宽 $bs > 1$, 并且将矩阵分块：

$$A = \begin{bmatrix} A_{1,1} & A_{2,1}^T & \cdots & A_{k,1}^T \\ A_{2,1} & A_{2,2} & \cdots & A_{k,2}^T \\ \vdots & \vdots & \ddots & \vdots \\ A_{k,1} & A_{k,2} & \cdots & A_{k,k} \end{bmatrix},$$

矩阵块 $A_{1:k-1,1:k-1}$ 大小均为 $bs \times bs$, 最下面一行和最右边一侧的矩阵块大小可能小于 $bs \times bs$. 考虑第一个 Householder vector $v_1 = (zeros(bs, 1), 1, x_1, \cdots, x_{n-bs-1})^T$, 其对应的 Householder 变换 $H$ 在左乘和右乘 $A$ 的时候会改变 $A(bs+1:n,:)$ 和 $A(:,bs+1:n)$. 这样一来，$A_{:,1}$ 和 $A_{1,:}$ 中的列和行进行更新的时候，就像 Householder QR 一样仅仅依赖于当前行的值和 $\beta, v$ 的值，不依赖于任何 $A_{p,q},\ p,q \geq 2$ 的信息。因此，对于右下角的矩阵块

$$\tilde{A} = \begin{bmatrix} A_{2,2} & \cdots & A_{k,2}^T \\ \vdots & \ddots & \vdots \\ A_{k,2} & \cdots & A_{k,k} \end{bmatrix}$$

的更新，可以被推后。类似地，第二个 Householder vector $v_2 = (zeros(bs+1, 1), 1, x_1, \cdots, x_{n-bs-2})^T$ 对应的 Householder 变换对 $\tilde{A}$ 的更新也可以被推迟。前 $bs$ 个 Householder 变换对 $\tilde{A}$ 的更新都可以推迟并且可以被合并用 WY 表示法和 BLAS-3 操作。因此，sy2sb 的外循环第一步就是将 $A_1 = [A_{1,1}; A_{2,1}; \cdots; A_{k,1}]$ 用 `house_qr()` 进行分解, 生成对应的 $W, Y$, 然后更新 $\tilde{A}$. 之后用同样的方法不断处理 $\tilde{A}$ 直至得到多对角矩阵。

最后我们要处理一下最下面一行和最右边一侧的矩阵块不是正方形的情形。这个边角情况在 [Grimes1988] 被略过了。考虑一个简单的例子：$n = 7, bs = 3$, 对应的矩阵分块为

$$A = \begin{bmatrix} A(1:4, 1:4) & A(1:4, 5:7) \\ A(5:7, 1:4) & A(5:7, 5:7) \end{bmatrix}.$$

Rainmaker's Notebook
『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

$A$ 在 sy2sb 以后变成了一个七对角矩阵 $B$，$B(7,1:3)=0$, 但是 $B(7,4:7)$ 不是 0. 我们需要三个 Householder vectors 对应的三个 Householder 变换来完成清零。因此，$M=A(5:7,1:3)$, 但 WY 表示法更新只更新 $A(5:7,5:7)$, 留下了 $A(5:7,4)$ 未被处理。记 $[Q_M,R_M]=qr(M)$, 由分块矩阵乘法可知

$$B=\begin{bmatrix} A(1:4,1:4) & A(1:4,5:7)Q_M \\ Q_M^TA(5:7,1:4) & Q_M^TA(5:7,5:7)Q_M \end{bmatrix}.$$

其中，$Q_M^TA(5:7,1:3)=R_M^T$, 则有 $B(5:7,4)=Q_M^TA(5:7,4)$.

综合上面的分析，我们可以得到如下 sy2sb 算法：

```
function VB = house_sy2sb(A, bs)
% Reduces a real symmetric matrix A to real symmetric band-diagonal
% form VB using blocked Householder transforms
% Reference: 10.1145/44128.44130, Section 3
% Input parameter:
%   A  : Size n * n, symmetric matrix
%   bs : Semi bandwidth of T, >= 2, default is 8
% Output parameters:
%   VB : Size n * n, tril(VT, -bs-1) stores the Householder vectors in sy2sb,
%        the middle is a symmetric band-diagonal matrix with bandwidth 2*bs+1
    n = size(A, 1);
    if (nargin < 2), bs = 8; end
    n1 = n - bs - 1;  % The last column in VB that its last row is 0
    for j = 1 : bs : n1
        % Columns in the current panel: [j, k]
        k = min(n1, j + bs - 1);
        curr_bs = k - j + 1;
        % Note: the first row of A in the panel is l = j + bs, not k + 1
        % QR factorize M = A(l : n, j : k), size(M, 1) >= size(M, 2) always holds
        l = j + bs;
        A(l : n, j : k) = house_qr(A(l : n, j : k));
        R = triu(A(l : l+curr_bs-1, j : k));
        A(j : k, l : l+curr_bs-1) = R';
        if (k < l-1)
            % Last block, where l > k, need to handle columns [k+1, l-1]
            A(l : n, k+1 : l-1) = apply_house_QT_bwd(A(l : n, j : k), A(l : n, k+1 : l-1));
            A(k+1 : l-1, l : n) = A(l : n, k+1 : l-1)';
        end
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
29          % Build W and Y from V
30          Y = tril(A(l : n, j : k), -1) + eye(n-l+1, curr_bs);
31          W = gen_W_from_Y(Y);
32          % Q_k = eye(n) - W * Y', apply Q_k' * A * Q_k
33          S = A(l : n, l : n) * W;
34          V = W' * S;
35          T = S - 0.5 * Y * V;
36          YTT = Y * T';
37          A(l : n, l : n) = A(l : n, l : n) - YTT - YTT';
38      end
39      VB = A;
40  end
```

其中 `apply_house_QT_bwd` 对应 $Q_M^T X$, 与 `apply_house_Q_bwd` 的区别仅仅是第 11 行。

```
1   function Y = apply_house_QT_bwd(V, X)
2   % Backward accumulation of the Householder matrices in reversed order
3   % Van Loan book 4th edition Section 5.1.6
4   % Input parameters:
5   %   V : Size m * n, its strict lower part contains the Householder vectors
6   %   X : Input vectors, size m * k
7   % Output parameter:
8   %   Y : Y = Qn * ... * Q2 * Q1 * X
9       [m, n] = size(V);
10      Y = X;
11      for j = 1 : n
12          v = [1; V(j+1 : m, j)];
13          b = 2 / (norm(v).^2);
14          % Y = Q_j * Y = (I - b * v * v') * Y
15          t = v' * Y(j : m, :);
16          Y(j : m, :) = Y(j : m, :) - (b .* v) * t;
17      end
18  end
```

将 sy2sb 得到的 $Q$ 左乘 $X$ 的算法和 `apply_house_Q_WY_bwd()` 类似：

```
1   function Z = apply_sy2sb_Q_WY(VB, bs, C)
2   % Apply the Q matrix from sy2sb to C (calculate Q * C)
3   % Input parameters:
4   %   VB : Size n * n, tril(VT, -bs-1) stores the Householder vectors in sy2sb
5   %   bs : Semi-bandwidth of the B matrix
6   %   C  : Size n * k, matrix to be applied with Q
7   % Output parameter:
8   %   Z : Size n * k, == Q * C
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
    n = size(VB, 1);
    n1 = n - bs - 1;  % The last column in B that its last row is 0
    n2 = 1 + floor((n1 - 1) / bs) * bs;
    for j = n2 : -bs : 1
        % Columns in the current panel: [j, k]
        k = min(n1, j + bs - 1);
        curr_bs = k - j + 1;
        l = j + bs;
        % Extract Y from V and build W
        Y = tril(VB(l : n, j : k), -1) + eye(n-l+1, curr_bs);
        W = gen_W_from_Y(Y);
        % Q * C(l : n, :) = (I - W * Y') * C(l : n, :)
        YTC = Y' * C(l : n, :);
        C(l : n, :) = C(l : n, :) - W * YTC;
    end
    Z = C;
end
```

## 6. 多对角到三对角矩阵变换 (sb2st) 算法

求解对称多对角矩阵仍然不是一件容易的事情。ELPA 和 SLATE 的 2nd stage 将对称多对角矩阵进一步变换到对称三对角矩阵，然后复用对称三对角矩阵的分治求解。有一个日本人的特征值求解器 EigenExa 不走寻常路，在 sy2sb 以后直接求解对称多对角矩阵的特征值和特征向量。他们宣称这样比变换到三对角再求解更快，但是我没有验证过这个说法。不过 EigenExa 这么做的原因是很合理的，因为本节涉及的算法比前面的都更难从数学上严格论证，更难以直观理解。

sb2st 依然基于 Householder 变换。这就带来了一个问题：如何一边破坏矩阵的稀疏结构，一边利用矩阵的稀疏结构。

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

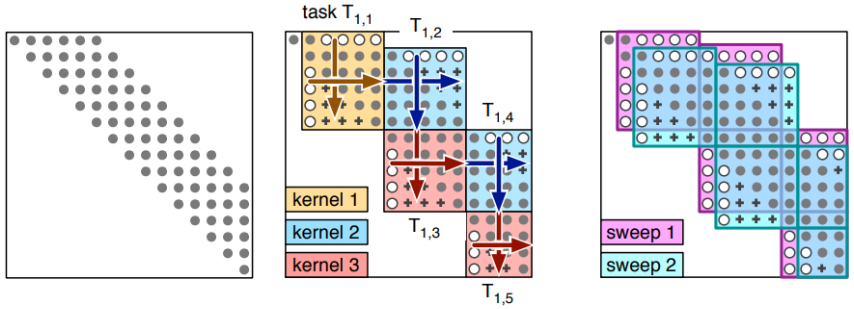



**Figure 2.3:** Bulge-chasing algorithm. "o" indicates eliminated elements; "+" indicates fill. Arrows show application of Householder reflector on left $(\rightarrow)$, which update a block row, and on right $(\downarrow)$, which update a block column.

上面这张图来自 [SLATE-EVD]。我们先考虑第一步，将第一行（列）除三对角以外的元素清零。此时，Householder vector 的长度等于半带宽 $bs = 5$. 在两侧 apply Householder 变换（记为变换 (1, 1)）后，$A(1, 3:6)$ 被清零了（图 (b) 中黄色方框里第一行的空心圆圈），但是 $A(2:6, :)$ 生成了一些新的非零元（图 (b) 中 $T_{1,2}$ 下方蓝色方框中的空心圆圈和加号）。下一步，我们先去把这些新填入的非零元中的第一行 $A(2, 8:11)$ （图 (b) 中 $T_{1,2}$ 下方蓝色方框中的空心圆圈）进行清零，这个变换记为 (1, 2)。$A(2, 4:7)$ 中的非零元暂时先不管。这一步清零又在 $A(7:9, :)$ 引入了新的非零元（图 (b) 中 $T_{1,4}$ 下方蓝色方框中的空心圆圈和加号）。这时我们放过 $A(3:6, :)$ 中由变换 (1,1) 引入的非零元，而是追着新引入的非零元的第一行 $A(7, 13:15)$ 进行消除，这个消除的变换记为 (1,3). (1,3) 没有再引入新的非零元。到此我们暂时告一段落，并将变换 (1, 1:3) 称为 sweep 1.

虽然 sb2st 还远远没有完成，但是让我们先看一下 sweep 1 的规律：

- 将第 1 行变换成三对角并引入新的非零元，
- 把上一步引入的非零元中的第 1 行（即整个矩阵的第 $1 + 1$ 行）中引入的新非零元消除掉并再次在第 $1 + 1 + bs$ 行开始引入新非零元，
- 把上一步引入的非零元中的第 1 行（即整个矩阵的第 $1 + 1 + bs$ 行）中引入的新非零元消除掉。

如果 bs 小一点，sweep 后面几步都会变成『把上一步引入的非零元中的第 1 行（即整个矩阵的第 $1 + 1 + j \cdot bs$ 行）中引入的新非零元消除掉并再次在第 $1 + 1 + (i + 1) \cdot bs$ 行开始引入新非零元』，直到没有新非零元引入。

接下来我们看第二行。sweep 1 中的第二步使得矩阵的第二行依然保留着半带宽 $bs$, 因此我们可以得到类似 sweep 1 的变换序列：

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

- 将第 2 行变换成三对角并引入新的非零元，
- 把上一步引入的非零元中的第 1 行（即整个矩阵的第 $2+1$ 行）中引入的新非零元消除掉并再次在第 $2+1+bs$ 行开始引入新非零元，
- 重复：把上一步引入的非零元中的第 1 行（即整个矩阵的第 $2+1+j\cdot bs$ 行）中引入的新非零元消除掉，并再次在第 $2+1+(i+1)\cdot bs$ 行开始引入新非零元，直到没有新非零元引入。

基于这些观察，我们可以得到一个看起来并不复杂的 naive sb2st 算法，其中 `k` 是 sweep index，`j` 是每一次清零的列，(k, l) 指定了一个特定的变换：

```
function [Q, T] = house_sb2st_v0(B, bs)
% Reduce a symmetric band matrix to symmetric tridiagonal matrix
% Reference: DOI 10.1109/SHPCC.1994.296622, bulge chasing algorithm
% Input parameters:
%   B  : Size n * n, symmetric matrix with bandwidth 2*bs+1
%   bs : Semi-bandwidth of B
% Output parameters:
%   Q : Size n * n, orthogonal matrix, Q * T * Q' == B
%   T : Size n * n, symmetric tridiagonal matrix
    n = size(B, 1);
    Q = eye(n);
    % Sweep from 1st column to (n-2)-th column
    for k = 1 : n-2
        j_list = [k, k+1 : bs : n-bs-1];
        for l = 1 : length(j_list)
            % Householder transform (k, l) in the paper
            % When transforming the j-th column, [r1, r2] are nonzeros, [r2+1, n] are zeros
            % [r1, r2]s in the same k do not overlap with each other
            j = j_list(l);
            if (l == 1)
                r1 = j + 1;
            else
                r1 = j + bs;
            end
            r2 = min(r1 + bs - 1, n);
            [v, b] = house_vec(B(r1 : r2, j));
            % The full Householder vector is [zeros(r1-1, 1); v; zeros(n-r2, 1)];
            % (H' * B) * H will first update B(r1 : r2, :), then update B(:, r1 : r2)
            H = eye(r2-r1+1) - (b * v) * v';
            B(r1 : r2, :) = H' * B(r1 : r2, :);
            B(:, r1 : r2) = B(:, r1 : r2) * H;
            % Accumulate Q = Q * Q_k = Q - b * (Q * v) * v';
            % Each j reads and updates the same Q columns, and different j in
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
34            % the same k reads and updates different Q columns (can be parallelized)
35            t = Q(:, r1 : r2) * v;
36            Q(:, r1 : r2) = Q(:, r1 : r2) - (b .* t) * v';
37        end
38    end
39    T = B;
40 end
```

为了方便对比和理解，上面这个算法在对 $B$ 进行变换的同时也在生成 $Q$. 要注意的是，这个算法并没有利用 $B$ 的稀疏性质：第 30, 31, 36 行使用的 $B$ 和 $Q$ 某些行/列的全部列/行。很不幸，[Bischof1994, Auck2011] 都没有具体提及利用如何利用稀疏性质，[Lang1993] 似乎也没有提及（此文从第三节开始就在讨论如何并行）。我观察了 `house_sb2st_v0()` 运行中间的 $B$ 矩阵变化，自己摸索出了下面这个新的版本：

```
1  function VT = house_sb2st_v2(B, bs)
2  % Reduce a symmetric band matrix to symmetric tridiagonal matrix
3  % Reference: DOI 10.1109/SHPCC.1994.296622, bulge chasing algorithm
4  % Input parameters:
5  %   B  : Size n * n, symmetric matrix with bandwidth 2*bs+1
6  %   bs : Semi-bandwidth of B
7  % Output parameters:
8  %   VT : Size n * n, its tridiagonal part is T, tril(VT, -2) contains all
9  %        Householder vectors in sb2st
10    n = size(B, 1);
11    % Sweep from 1st column to (n-2)-th column
12    v_tmp = zeros(n, 1);
13    for k = 1 : n-2
14        j_list = [k, k+1 : bs : n-bs-1];
15        for l = 1 : length(j_list)
16            % Householder transform (k, l) in the paper
17            % When transforming the j-th column, [r1, r2] are nonzeros, [r2+1, n] are zeros
18            % [r1, r2]s in the same k do not overlap with each other
19            j = j_list(l);
20            if (l == 1)
21                r1 = j + 1;
22            else
23                r1 = j + bs;
24            end
25            r2 = min(r1 + bs - 1, n);
26            [v, b] = house_vec(B(r1 : r2, j));
27            % The full Householder vector is [zeros(r1-1, 1); v; zeros(n-r2, 1)];
28            % H' * B updates B(r1 : r2, j : r3), B * H updates B(j : r3, r1 : r2)
29            if (l == 1)
```

Rainmaker's Notebook
『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》



```
30                r3 = min(n, r1 + 2 * bs);
31            else
32                r3 = min(n, r1 + 3 * bs - 1);
33            end
34            t = v' * B(r1 : r2, j : r3);
35            B(r1 : r2, j : r3) = B(r1 : r2, j : r3) - (b .* v) * t;
36            t = B(j : r3, r1 : r2) * v;
37            B(j : r3, r1 : r2) = B(j : r3, r1 : r2) - (b .* t) * v';
38            v_tmp(r1 : r2) = v;
39        end
40        % v_tmp(k) is [zeros(1, k), 1, x, ..., x];
41        B(k+2 : n, k) = v_tmp(k+2 : n);
42    end
43    VT = B;
44 end
```

在 `house_sb2st_v2 ()` 中，每次更新的 $B$ 矩阵块大小不超过 $4(bs)^2$, 符合 [Bischof1994] 里面讲 sb2st 部分提到的 “a Householder update involving $b$ rows of $A$ requires only $O(b^2)$ flops”. 此外，sweep k 中的所有 Householder vectors 存储到了 $B(:,k)$, 可以之后用来计算 $Q$ 左乘 $X$.

进一步观察可知，sb2st 中的对 $B$ 应用各个变换的顺序并不是完全固定的。[Bischof1994] 中给出了三个要求，a < b 表示 b 要在 a 后面：(a) (k, l) < (k+1, l), (b) (k,l) < (k, l+1); (c) (k, l) < (k+1, l-1). 观察本节第一张图可以发现，(1, 1) 和 (1, 2) 完成以后，(1, 3) 不会再触碰 (2, 1) 需要的数据，因此 (1, 3) 和 (2, 1) 可以同时计算。而对于 $Q$ 左乘 $X$, 同一个 sweep 中的不同变换更新的是不同的 $Q$ 的列，故 $Q$ 的生成和使用不再受前述三个顺序要求中 (b) 的限制。

下图来自 [Auck2011], 给出了 sb2st 变换过程和计算 $Q$ 左乘 $X$ 的前后依赖顺序，其中 $U_l^{(k)}$ 对应前述变换 (k, l). 两侧的图各自构成了一个 DAG，可以根据此 DAG 进行并行。

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

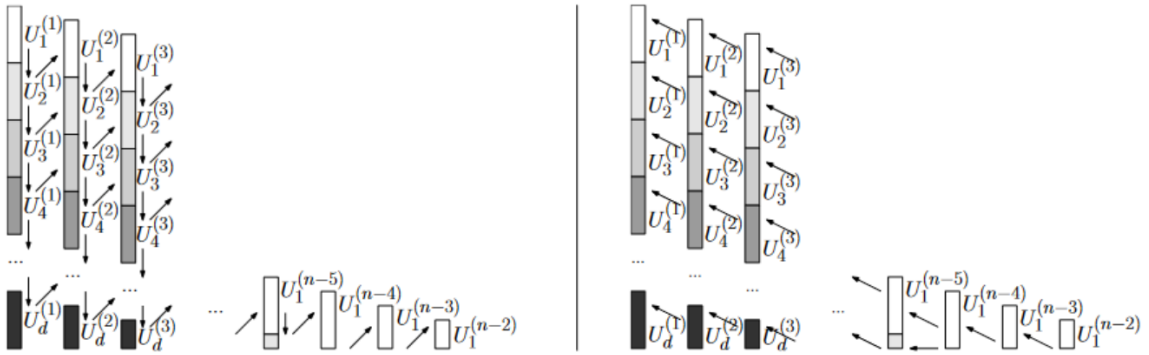


Figure 2: Householder vectors from the bulge-chasing algorithm for the reduction from banded to tridiagonal form [44]. The arrows indicate the order of execution during reduction (left) and back transformation of eigenvectors (right).

对于计算 $Q$ 左乘 $X$, 我们可以选择 DAG 中的若干个节点，合并其 Householder 变换并应用 WY 表示法进行更新。我使用的合并顺序是沿着对角线斜向上，下面是我的代码：

```
function Z = apply_sb2st_Q_WY(VT, bs, nv, C)
% Apply the Q matrix from sb2st to C (calculate Q * C)
% Based on DOI:10.1016/j.parco.2011.05.002 Figure 2 right part and
% DOI:10.1109/SHPCC.1994.296622 Section 3
% Input parameters:
%   VT : Size n * n, tril(VT, -2) stores the Householder vectors in sb2st
%   bs : Semi-bandwidth of the AB matrix entering sb2st, == length of each
%        Householder vectors in sb2st
%   nv : Number of Householder vectors to combine
%   C  : Size n * k, matrix to be applied with Q
% Output parameter:
%   Z : Size n * k, == Q * C
    n = size(VT, 1);
    l_num = zeros(n-2, 1);
    for k = 1 : n-2
        j_list = [k, k+1 : bs : n-bs-1];
        l_num(k) = length(j_list);
    end
    for l = 1 : l_num(1)
        for k1 = n-2 : -1 : 1
            if (l_num(k1) >= l), break; end
        end
        for ke = k1 : -nv : 1
            % Combine transforms (ke : -1 : ks, l)
```

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
            ks = max(1, ke - nv + 1);
            curr_nv = ke - ks + 1;
            Y = zeros(bs + curr_nv - 1, curr_nv);
            min_r1 = 1e100;
            max_r2 = 0;
            for k = ke : -1 : ks
                % Calculate j, r1, r2 from (k, l) and extract corresponding v
                if (l == 1)
                    j = k;
                    r1 = j + 1;
                else
                    j = k + 1;
                    if (l > 2), j = j + (l - 2) * bs; end
                    r1 = j + bs;
                end
                r2 = min(r1 + bs - 1, n);
                v = VT(r1 : r2, k);
                if (l == 1), v(1) = 1; end
                % Record min_r1, max_r2; put v in Y
                min_r1 = min(min_r1, r1);
                max_r2 = max(max_r2, r2);
                Y_r1 = k - ks + 1;
                Y_r2 = Y_r1 + (r2 - r1);
                Y_col = ke - k + 1;
                Y(Y_r1 : Y_r2, Y_col) = v;
            end
            % length(v) might be smaller than bs, truncate empty rows in Y
            Y = Y(1 : max_r2-min_r1+1, :);
            W = gen_W_from_Y(Y);
            % Q' * C(min_r1 : max_r2, :) = (I - Y * W') * C(min_r1 : max_r2, :)
            % Why we are using Q' * C instead of Q * C here, but Q * C in apply_sy2sb_Q_WY?
            WTC = W' * C(min_r1 : max_r2, :);
            C(min_r1 : max_r2, :) = C(min_r1 : max_r2, :) - Y * WTC;
        end
    end
    Z = C;
end
```

## 7. 代码的组合使用

1-stage 方法：

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

```
[Q, T] = house_tridiag(A);
[V1, D] = tridiag_eig_dc(T);
V = Q * V1;
% Check error norm: norm(V * D * V' - A, 'fro') / norm(A, 'fro')
```

2-stage 方法：

```
 B_nnz_pattern = toeplitz([ones(1, 1+bs), zeros(1, n-bs-1)]);
 VB = house_sy2sb(A, bs);
 B = VB .* B_nnz_pattern;

 T_nnz_pattern = toeplitz([ones(1, 2), zeros(1, n-2)]);
 VT = house_sb2st_v2(B, bs);
 T = VT .* T_nnz_pattern;

 [V1, D] = tridiag_eig_dc(T);
V2 = apply_sb2st_Q_WY(VT, bs, nv, V1);
V = apply_sy2sb_Q_WY(VB, bs, V2);
% Check error norm: norm(V * D * V' - A, 'fro') / norm(A, 'fro')
```

目前 `tridiag_eig_dc` 不是很稳定，可以换成 MATLAB 的 `eig` 来验证其他各个函数的正确性。

## 参考文献

1. (VL4) Maxtrix Computation 4th Edition
2. (Marek2014) The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science
3. (Auck2011) Parallel solution of partial symmetric eigenvalue problems for electronic structure calculations
4. (Bischof1994) Parallel tridiagonalization through two-step band reduction
5. (Lang1993) A Parallel Algorithm for Reducing Symmetric Banded Matrices to Tridiagonal Form
6. (Grimes1988) Solution of large, dense symmetric generalized eigenvalue problems using secondary storage
7. (ECNU-MC) 华东师范大学潘建瑜教授矩阵计算讲义
8. (SLATE-EVD) SLATE working note 13: Implementing Singular Value and Symmetric/Hermitian Eigenvalue Solvers

发布于 2023-01-14

tags: { LinearAlgebra } { Householder } { QR } { Eigensolver }

Rainmaker's Notebook

『求雨巫师的神奇之处在于他总是躲着不见你，却总说刚下完的雨是拜他所赐。』——《天真的人类学家》

Home

Archives

About

SiteXC

0 comments

Anonymous

Leave a comment

Markdown is supported

Login with GitHub

Preview

Be the first person to leave a comment!