5. Least squares problems and orthogonalization

Figure 5: Linear regression - best linear approximation from measurements in a least squares sense.

5.7 Appendix

5.8 Linear Regression Example

Example 19 (A classic example: best linear approximation from measurements. Skipped). Suppose that you are given with $n$ measurements $(x_i, y_i)$ and wish to find the line that defines \{y_i\} given \{x_i\}. That is, find scalars $a, b$ so that $ax_i + b \approx y_i$ for all $i$ in an optimal way. This can be written in matrix form

$$
\begin{bmatrix}
    x_1 & 1 \\
    x_2 & 1 \\
    \vdots & \vdots \\
    x_n & 1 \\
\end{bmatrix}
\begin{bmatrix}
    a \\
    b \\
\end{bmatrix}
\approx
\begin{bmatrix}
    y_1 \\
    y_2 \\
    \vdots \\
    y_n \\
\end{bmatrix}.
$$

A common choice for the optimality measure is the $\ell_2$ norm. We get a problem

$$
\min_{a,b} \sum (ax_i + b - y_i)^2,
$$
5. Least squares problems and orthogonalization

or

\[
\min_{a,b} \left\| \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right\|_2.
\]

This problem is also called the linear regression, and boils to the above least squares minimization. Figure 5 illustrates the example.

**Example 20** (Linear regression continued). Assume again that we are given with \( m \) measurements \((x_i, y_i), x_i \in [0, 1]\), and wish to find the line that defines \( \{y_i\} \) given \( \{x_i\} \). We have the following minimization

\[
\arg \min_{a,b} \left\| \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right\|_2.
\]

Denote this problem as \( \arg \min \|Ax - b\| \), then the minimizer of this problem will be \((A^\top A)^{-1}A^\top b\) or:

\[
\begin{bmatrix} a \\ b \end{bmatrix} = \left( \frac{\sum_i x_i^2}{\sum_i x_i} \sum_i x_i \right)^{-1} \left[ \frac{\sum_i x_i y_i}{n} \right].
\]

The following code shows an example of linear regression. The output image appears in Fig. 6.
Example 21 (Linear regression with weights). Assume again that we are given with \( m \) measurements \((x_i, y_i)\) and wish to find the line that defines \( \{y_i\} \) given \( \{x_i\} \). This time, suppose that we know that the measurements \( y_i \) are noisier as \( x_i \) is smaller. That is - the magnitude of the errors in \( y_i \) are proportional to \( \frac{1}{\sqrt{x_i}} \). Obviously, the measurements \( y_i \) that correspond to the higher values of \( x_i \) are much more informative than the others. How will we treat that numerically? We will use a diagonal weight matrix \( W \), such that \( w_{ii} = x_i \). This will give more emphasis to the measurements with high values of \( x_i \). The code below...
5. Least squares problems and orthogonalization

Figure 7: Weighted linear regression code example.

shows this example of (weighted) linear regression. The output image appears in Fig. 7, and it shows that for this example, the reconstructed line with weights is better than without weights.

```python
x = linspace(0.01,1.0,200);
a = 0.8;
b = 0.4;
epsilon = 0.3.*randn(length(x))./sqrt(x);
W = diagm(x);
y = a.*x + b + epsilon;
A = [x ones(length(x))];
B = copy(y);
opt_ab_no_w = (A'*A) \ A'*B;
opt_ab = (A'*W*A) \ A'*W*B);
using PyPlot
close("all"); plot(x,y,".b");
plot(x,a*x + b,"-r");
plot(x,opt_ab_no_w[1]*x + opt_ab_no_w[2],"-k");
plot(x,opt_ab[1]*x + opt_ab[2],"-g");
title("Weighted Least Squares: linear regression example");
legend(["Measurements","True line","Est. line w/o weights","Est. line w weights"]);```

Figure 7: Weighted linear regression code example.
5. Least squares problems and orthogonalization

5.8.1 Image de-blurring regularized least squares example

**Example 22** (Image deblurring using Tikhonov regularization). *The next example is one of the simplest and yet interesting examples for Tikhonov regularization. Assume that we have an image $\mathbf{x}$ that undergoes a blurring operation by some matrix $\mathbf{A}$, and is corrupted by additive white Gaussian noise. That is, we have the vector $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n}$, where $\mathbf{A}$ is a blurring matrix, $\mathbf{n}$ is a noisy image. We wish to recover $\mathbf{x}$ from $\mathbf{y}$. Figure 8 shows a clean image and a corrupted one.*

In the simple case of deblurring, we assume that we know $\mathbf{A}$, and we assume that the noise is small. Our first attempt at the reconstruction would be to solve $\hat{\mathbf{x}} = \arg \min \| \mathbf{A}\mathbf{x} - \mathbf{y} \|_2^2$, which in this case would be just $\mathbf{A}^{-1}\mathbf{y}$ since $\mathbf{A}$ is square and full rank. Another attempt will be to add regularization, and solve $\hat{\mathbf{x}} = \arg \min \| \mathbf{A}\mathbf{x} - \mathbf{y} \|_2^2 + \lambda \| \mathbf{x} \|_2^2$ for some $\lambda$. The resulting reconstructions appear in Fig. 9. The first attempt led to essentially nothing, while the second attempt indeed led to a reasonable reconstruction. How did that happen? Apparently, in the first attempt, in addition to sharpen the image we also amplified the noise so much that it ruined the image - this amplification is due to small eigenvalues of $\mathbf{A}$. In the second...*
Figure 9: Deblurring example - reconstructions using LS and regularized LS.

attempt we added an identity, which erased the influence of the small eigenvalues and left the large ones more or less as they are. This prevented the amplification of noise. Below is the code of the program.
### EXAMPLE - LS with regularization using MAT

```matlab
using Matplotlib; close("all")
# Read the image and downsize it
a = matread("lena512.mat");
a = a["lena512"]; n1 = size(a,1); n2 = size(a,2);
blur1 = spdiagsm((0.25*ones(n1-1), 0.5*ones(n1), 0.25*ones(n1-1)), (-1, 0, 1), n1, n1);
blur2 = spdiagsm((0.25*ones(n1-1), 0.5*ones(n1), 0.25*ones(n1-1)), (-1, 0, 1), n2, n2);
A = kron(blur2,blur1);
a = reshape(A*a[:], n1,n2);
a = a[1:2:end,1:2:end];
n1 = 256; n2 = 256;
# Define the blurring operator A
blur1 = spdiagsm((0.25*ones(n1-1), 0.5*ones(n1), 0.25*ones(n1-1)), (-1, 0, 1), n1, n1);
blur2 = spdiagsm((0.25*ones(n1-1), 0.5*ones(n1), 0.25*ones(n1-1)), (-1, 0, 1), n2, n2);
A = kron(blur2,blur1);
A = A^2;
# Define the noisy image
noise = randn(n1,n2);
b = reshape(A*a[:], n1,n2) + 10.0*noise;
# First figure
figure()
subplot(1,2,1)
imshow(a,cmap = "gray")
title("original image")
subplot(1,2,2)
imshow(b,cmap = "gray")
title("noisy blurred image")

# Reconstructions and second figure
a_ls = reshape(A\b[:], n1,n2);
a_rls = reshape((A'*A + 0.5*speye(n1*n2))\(A'*b[:]), n1,n2);
figure()
subplot(1,2,1)
imshow(a_ls,cmap = "gray")
title("LS reconstructed image")
subplot(1,2,2)
imshow(a_rls,cmap = "gray")
title("regularized LS reconstructed image")
```
5.8.2 Proof for the SVD existence Theorem

**Theorem 21.** Let $A \in \mathbb{R}^{m \times n}$, then there exist orthogonal matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$, such that $V^T V = I_n$, $U^T U = I_m$ and

$$A = U \Sigma V^T,$$

where $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \ldots \geq \sigma_p \geq 0$. This is called the full SVD factorization of $A$.

**Proof.** Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ be two unit norm vectors, such that $Ax = \sigma y$ with $\sigma = \|A\|_2$. Let $V_1 \in \mathbb{R}^{n \times (n-1)}$ and $U_1 \in \mathbb{R}^{m \times (m-1)}$ be any orthogonal matrices that complement $x$ and $y$ to a full space, such that $V = [x|V_1] \in \mathbb{R}^{n \times n}$ and $U = [y|U_1] \in \mathbb{R}^{m \times m}$ are also orthogonal matrices. The matrix $U^T A V$ has the following structure:

$$U^T A V = U^T [\sigma y|AV_1] = \begin{bmatrix} \sigma & w^T \\ 0 & B \end{bmatrix} = A_1,$$

where $w \in \mathbb{R}^{n-1}$ is an unknown vector. We will now show that $w = 0$, and this will allow to complete the proof by induction argument. We have that

$$\|A_1\|_2^2 \geq (\sigma^2 + w^T w)^2,$$

so $\|A_1\|_2^2 \geq \sigma^2 + w^T w$. But $\|A_1\|_2 = \|A\|_2 = \sigma$, and so we must have that $w = 0$. It is also easy to show that $\sigma \geq \|B\|_2$, otherwise this would contradict that $\|A_1\|_2 = \sigma$.

The rest of the proof can be completed by induction, assuming that

$$U^T A V = \begin{bmatrix} \Sigma & 0 \\ 0 & B \end{bmatrix}$$

for $\Sigma_k = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k) \in \mathbb{R}^{k \times k}$, and $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \ldots \geq \sigma_k \geq 0$, are the $k$ largest singular values and $\sigma_k \geq \|B\|_2$. Applying the steps above to the matrix $B \in \mathbb{R}^{(m-k) \times (n-k)}$, we can find orthogonal matrices $U_2 \in \mathbb{R}^{(m-k) \times (m-k)}$ and $V_2 \in \mathbb{R}^{(n-k) \times (n-k)}$ such that

$$U_2^T B V_2 = \begin{bmatrix} \sigma_{k+1} & 0 \\ 0 & C \end{bmatrix},$$
for some matrix $C$ and $\sigma_{k+1} = \|B\|_2$. Now $U$ and $V$ in Eq. (49) can be multiplied by the two orthogonal matrices $\text{blockdiag}(I_{k\times k}, U_2)$, and $\text{blockdiag}(I_{k\times k}, V_2)$ respectively, resulting in new orthogonal matrices $U, V$ that lead to $U^T AV = \begin{bmatrix} \Sigma_{k+1} & 0 \\ 0 & C \end{bmatrix}$.

5.8.3 The Householder orthogonal matrices

A Householder matrix is a square matrix of the form

$$Q = I - 2ww^T,$$

where $w$ is a normalized vector. It’s easy to show that any Householder matrix is orthogonal, i.e., $Q^T Q = I$. Householder matrices are usually used to zero entries in vectors while keeping the $\ell_2$ norm of the vector intact. That is, given a vector $v$, it’s easy to generate a matrix $Q$ such that $Qv = \|v\|_2e_1$, where $e_1$ is the first unit vector. To find the vector $w$ that will be used for that we will write $Qv = v - 2ww^Tv = \|v\|_2e_1$. Marking $\beta = \|v\|_2$, and realizing that $w^Tv$ is just a scalar, we can see that

$$w = \frac{v - \beta e_1}{\|v - \beta e_1\|_2} \triangleq \text{Householder}(u).$$

To show that this is indeed what we want we’ll notice that $(v - \beta e_1)^Tv = \beta^2 - \beta v_1$, and similarly, $\|v - \beta e_1\|^2_2 = v^Tv - 2\beta v_1 + \beta^2 = 2\beta^2 - 2\beta v_1$. Using these

$$v - 2ww^Tv = v - 2(v - \beta e_1)\frac{(v - \beta e_1)^Tv}{\|v - \beta e_1\|^2_2}$$

$$= v - 2(v - \beta e_1)\frac{\beta^2 - \beta v_1}{2\beta^2 - 2\beta v_1}$$

$$= v - (v - \beta e_1) - \beta e_1.$$

We will not show this, but using these matrices one can form a QR factorization. At the first iteration $\beta$ will be $R_{11}$, and we will form a matrix $Q_1$ such that $Q_1a_1$ will be $R_{11}e_1$. Then we consider only the $n - 1$ bottom entries and repeat this process, zeroing the bottom $n - 2$ entries of the second column of the matrix $Q_1A$.

We will instead consider a somewhat similar process, called Householder Bi-diagonalization.
that is a key step in computing SVD decomposition.

**SVD computation via householder bi-diagonalization**  In this section we will give the concept of the computation of SVD and eigendecomposition, through a process that is called householder bi/tri-diagonalization.

Assume that we have \( A \in \mathbb{R}^{m \times n} \), and assume without loss of generality that \( m \geq n \) (otherwise we can work with \( A^\top \)). We wish to find orthogonal matrices \( U \in \mathbb{R}^{m \times n} \) and \( V \in \mathbb{R}^{n \times n} \), such that \( A = UBV^\top \) and \( B \in \mathbb{R}^{n \times n} \) is bi-diagonal.

The motivation for doing this is simple: having an SVD decomposition is similar to having an eigen-decomposition - the SVD is actually achieved through the eigendecomposition of \( A^\top A \). To have an eigendecomposition, one must at some point calculate the roots of polynomial \( p(\lambda) = 0 \), that can be calculated through a determinant. Working with bi/tri-diagonal matrices enables us to compute the determinant of such matrices efficiently, using the Thomas algorithm for example.
The bi-diagonalization process can be illustrated using the following diagram:

5.4.3 Bidiagonalization

Suppose $A \in \mathbb{R}^{m \times n}$ and $m \geq n$. We next show how to compute orthogonal $U_B$ ($m$-by-$m$) and $V_B$ ($n$-by-$n$) such that

$$U_B^T AV_B = \begin{bmatrix} d_1 & f_1 & 0 & \cdots & 0 \\ 0 & d_2 & f_2 & 0 \\ \vdots & \ddots & \ddots & \ddots \\ 0 & \cdots & d_{n-1} & f_{n-1} \\ 0 & \cdots & 0 & d_n \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \end{bmatrix}$$

(5.4.6)

$U_B = U_1 \cdots U_n$ and $V_B = V_1 \cdots V_{n-2}$ can each be determined as a product of Householder matrices:

$$\begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix} \xrightarrow{v_1} \begin{bmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} \xrightarrow{v_2} \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & x & 0 & 0 \end{bmatrix} \xrightarrow{v_3}$$

$$\cdots$$

$$\begin{bmatrix} x & x & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & 0 & x & 0 \\ 0 & 0 & 0 & x \end{bmatrix} \xrightarrow{v_4} \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & 0 & 0 \\ 0 & 0 & x & 0 \\ 0 & 0 & 0 & x \end{bmatrix}.$$
The algorithm for the bidiagonalization process is given below.

**Algorithm: Householder bi-diagonalization**

# Input: $A \in \mathbb{R}^{m \times n}$
# Output: $B \in \mathbb{R}^{m \times n}$ bi-diagonal. $U, V$ orthogonal s.t $A = U B V^T$

Initialize: $B = A$, $U = I_m$, $V = I_n$.

for $k = 1, \ldots, \min(m, n)$ do
  if $k < m$ then
    $w = \text{Householder}(B_{k:m,k})$
    $B_{k:m,k:n} = B_{k:m,k:n} - 2w(w^T B_{k:m,k:n})$
    $U_{k:m,;} = U_{k:m,;} - 2w(w^T U_{k:m,;})$
  end

  if $k < n - 1$ then
    $w = \text{Householder}(B_{k:k+1:n})$
    $B_{k:m,k+1:n} = B_{k:m,k+1:n} - 2(B_{k:m,k+1:n}w)w^T$
    $V_{i,k+1:n} = V_{i,k+1:n} - 2(V_{i,k+1:n} * w)w^T$
  end

end

$U = U^T$

Algorithm 9: Householder bi-diagonalization

Once we apply a bidiagonalization process and have $A = U B V^T$, or $B = U^T A V$, we can rather easily compute the SVD decomposition of the bidiagonal matrix $B = \hat{U} \Sigma \hat{V}^T$, and then have $A = (U \hat{U}) \Sigma (V^T \hat{V})^T$, and essentially have the SVD decomposition of $A$ with $U \hat{U}$ and $V \hat{V}$ as the orthogonal matrices. The SVD decomposition of $B$ can be computed by the eigendecomposition of the tridiagonal $B^T B$. We will first have to find its eigenvalues using an iterative method like Bisection, Newton-Raphson, or others. We will not demonstrate this further in this course.

The code and a simple example for the bidiagonalization process are given below.
function house(v)
    beta = norm(v);
    e1 = zeros(eltype(v),length(v)); e1[1] = 1.0;
    w = v-beta*e1;
    w = w/norm(w);
    return w,beta;
end

function BiDiag(A)
    (m,n) = size(A);
    B = copy(A);
    U = eye(m);
    V = eye(n);
    for k=1:min(n,m)
        if k < m
            w, = house(vec(B[k:end,k]));
            B[k:end,k:end] = B[k:end,k:end] - 2*w*(w'*B[k:end,k:end]);
            U[k:end,:) = U[k:end,:] - 2*w*(w'*U[k:end,:]);
        end
        if k < n-1
            w, = house(vec(B[k,k+1:end]));
            B[k:end,k+1:end] = B[k:end,k+1:end] - 2*(B[k:end,k+1:end]*w)*w';
            V[:,k+1:end] = V[:,k+1:end] - 2*(V[:,k+1:end]*w)*w';
        end
    end
    U = U';
    return B,U,V;
end
Example 23 (Bi-diagonalization).

```julia
julia> A = [1.0  2.0  3; 4  5  6; 7  8  9; 10 11 12 ]
43 Array{Float64,2}:
1.0  2.0  3.0
4.0  5.0  6.0
7.0  8.0  9.0
10.0 11.0 12.0

julia> B,U,V = BiDiag(A);

julia> B
43 Array{Float64,2}:
12.8841  21.8764  0.0
0.0     2.24624 -0.613281
0.0    -2.22045e-16  1.82449e-15
0.0    -3.33067e-16  6.90253e-31

julia> U
44 Array{Float64,2}:
0.0776151  0.833052  -0.537377  0.105953
0.31046   0.451237   0.63753   -0.541807
0.543305  0.069421   0.33707   0.765757
0.776151 -0.312395   -0.437224  -0.329902

julia> V
33 Array{Float64,2}:
1.0   0.0  0.0
0.0   0.667002  0.745056
0.0   0.745056  -0.667002

julia> vecnorm(A - U*B*V')
1.2074409138593832e-14

julia> vecnorm(U'*U - eye(4))
1.090537212533903e-15

julia> vecnorm(V'*V - eye(3))
5.026748538604307e-16
```