The code below generated the figure, and produced values of the infinity norm of the error, $ME$, (maximum absolute value) and of the $L^2$ norm of the error, $NE$.

```matlab
hold off;
ME=zeros(9,4);
NE=ME;
for nu=4:10
    N=2^nu;
    x=(-N:N)'/N;
    A=[x.^0 x.^1 x.^2 x.^3];
    [Q,R]=qr(A,0);
    scale=Q(2*N+1,:);
    Q=Q*diag(1./scale);
    P=[1+0*x 1 1.5*x.^2-.5 2.5*x.^3-1.5*x];
    E=Q-P;
    ME(nu,:)=max(abs(E));
    NE(nu,:)=sqrt(2^(-nu)*sum(abs(E).^2));
    plot(x,E)
    hold on;
end
xlabel('x');
ylabel('Error');
```

The code below generated the figure, and produced values of the infinity norm of the error, $ME$, (maximum absolute value) and of the $L^2$ norm of the error, $NE$. 

```matlab
hold off;
ME=zeros(9,4);
NE=ME;
for nu=4:10
    N=2^nu;
    x=(-N:N)'/N;
    A=[x.^0 x.^1 x.^2 x.^3];
    [Q,R]=qr(A,0);
    scale=Q(2*N+1,:);
    Q=Q*diag(1./scale);
    P=[1+0*x 1 1.5*x.^2-.5 2.5*x.^3-1.5*x];
    E=Q-P;
    ME(nu,:)=max(abs(E));
    NE(nu,:)=sqrt(2^(-nu)*sum(abs(E).^2));
    plot(x,E)
    hold on;
end
xlabel('x');
ylabel('Error');
```
title('Error due to discretization for Legendre Polynomials')
hold off;

The values of \( ME \) and \( NE \) show that the errors for the first two Legendre polynomials are effective zero (near machine epsilon) and that the errors in the quadratic and cubic polynomials scale with \( \Delta x \). They are (roughly) halved every time \( \Delta x \) is halved.

```matlab
>> format short
>> ME

ME =

    0     0     0     0
    0     0     0     0
    0     0     0     0
    0 0.0000  0.0484  0.0972
  0.0000  0.0000  0.0238  0.0468
  0.0000  0.0000  0.0118  0.0230
  0.0000  0.0000  0.0059  0.0114
  0.0000  0.0000  0.0029  0.0057
  0.0000  0.0000  0.0015  0.0028
  0.0000  0.0000  0.0007  0.0014

>> NE

NE =

    0     0     0     0
    0     0     0     0
    0     0     0     0
    0 0.0000  0.0500  0.0986
  0.0000  0.0000  0.0246  0.0475
  0.0000  0.0000  0.0122  0.0233
  0.0000  0.0000  0.0061  0.0115
  0.0000  0.0000  0.0030  0.0057
  0.0000  0.0000  0.0015  0.0029
  0.0000  0.0000  0.0008  0.0014
```

9.2 a) Since \( A \) is upper triangular, its eigenvalues are its diagonal entries which are all 1. Its determinant is the product of the diagonals, so that is also 1. Of course since the determinant is nonzero, the rank of the matrix is m.
b) \[ A^{-1} = \begin{pmatrix}
1 & -2 & 4 & -8 & \ldots & -2^{m-1} \\
0 & 1 & -2 & 4 & \ldots & -2^{m-2} \\
0 & 0 & 1 & -2 & \ldots & -2^{m-3} \\
0 & 0 & 0 & 1 & \ldots & -2^{m-4} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & \ldots & 1
\end{pmatrix} \]
or more compactly \[ a_{ij} = \begin{cases} 
0 & \text{if } i > j \\
-2^{i-1} & \text{otherwise}
\end{cases} \]

To prove that this is the correct formula for \( A^{-1} \), we write \( A = I + 2N \), where \( N \) is an \( m \times m \) matrix with all entries zero except on the super-diagonal (i.e. \( n_{ij} = \delta_{i,j-1} \)). The linear transformation represented by \( N \) is a right shift. It is straightforward to check that the \( k \)th power of \( N \) has zeroes except for entries in the \( k \)th super-diagonal, and so in particular, \( N^m = 0 \). Using the distributive law together with that fact we have

\[
(I + 2N) \left( \sum_{k=0}^{m-1} (-2N)^k \right) = \sum_{k=0}^{m-1} (-2N)^k - \sum_{k=0}^{m-1} (-2N)^{k+1} \\
= \sum_{k=0}^{m-1} (-2N)^k - \sum_{k=1}^{m} (-2N)^k \\
= (-2N)^0 - (-2N)^m = I
\]

Thus we have shown \( A^{-1} = \sum_{k=0}^{m-1} (-2N)^k \).

\( c \) If we take the singular value decomposition of \( A = U\Sigma V^* \), then \( A^{-1} = V\Sigma^{-1}U^* \), and the singular values of \( A^{-1} \) are the inverses if those of \( A \). (Of course \( A^{-1} = V\Sigma^{-1}U^* \) is not a singular value decomposition since the diagonals are in reversed order. To get a singular value decomposition, use the Flip matrix \( F \) from problem 6.2, and write \( A^{-1} = (VF)F\Sigma^{-1}F(UF)^* \).) The singular values of \( A^{-1} \) are the inverses of the singular values of \( A \). Thus \( 1/\sigma_m \) is the largest singular value, and \( 1/\sigma_m = ||A^{-1}||_2 \). We need an upper bound on \( \sigma_m \), which translates into a lower bound on \( ||A^{-1}||_2 = \max_{||v||=1} ||A^{-1}v||_2 \). So in particular, if we take \( v = e_m \), we have \( A^{-1}e_m \) is the \( m \)th column of \( A^{-1} \), whose norm is \( \sqrt{\sum_{j=0}^{m-1} 2^{2j}} = \sqrt{2^m - 1} \). Thus \( 1/\sigma_m \geq (4^m - 1)/3 \) and \( \sigma_m \leq 1/\sqrt{(4^m - 1)/3} \). When I checked this, I found that this was not a sharp estimate, but, it seemed to be close to the actual values.

Reproduce figure 9.1 etc. In last week’s homework we already produced the listings for \( \text{mgs}(A) \) and for \( \text{clgs}(A) \).

To reproduce experiment 2 we run

```matlab
[U,X]=qr(randn(80));
[V,X]=qr(randn(80));
S=diag(2.~(-1:-1:-80));
A=U*S*V; [QC,RC]=clgs(A);
[QM,RM]=mgs(A);
semilogy(1:80,diag(RC),'o',1:80,diag(RM),'x',1:80,2.~(-1:-1:-80));
xlabel('j');
```
ylabel(’r_{jj}’);

yielding the plot:

To reproduce experiment 3, run

```matlab
A=[.7 .70711;.70001 .70711];
[Q,R]=qr(A);
norm(Q’*Q-eye(2))
[Q,R]=mgs(A);
norm(Q’*Q-eye(2))
```

and we get

```
>>
ans =
  2.3515e-016
ans =
  2.3014e-011
```

Figure 9.1 shows that the classical Gram-Schmidt is encountering errors which are of order $\sqrt{\epsilon_m}$ whereas the modified Gram-Schmidt algorithm seems to hit errors of order $\epsilon_m$. This is assuming that the correct results should continue to follow the curve $2^{-j}$ as “explained” at the bottom of page 65. This explains why the results last week were so close for the two algorithms. The differences seem to show up only when the test arrays are nearly rank deficient with very small singular values. Picking arrays at random, the chances of getting such a matrix are vanishingly small.
10.1 We can write the Householder reflector as \( F = I - 2vv^*/(v^*v) \), or equivalently, if we let \( u = v/\|v\|_2 \) so that \( \|u\|_2 = 1 \), \( F = I - 2uu^* \). \( Fu = u - 2uu^*u = -u \). Therefore \(-1\) is an eigenvalue with eigenvector \( u \). Pick an orthonormal basis \( \{w_1, w_2, \ldots, w_{m-1}\} \) for \( u^\perp \), the space of vectors orthogonal to \( u \). \( Fw_j = w_j - 2uu^*w_j = w_j \), so each of the \( w_j \) are independent eigenvectors with eigenvalue 1. Geometrically, \( F \) is reflection through a hyperplane, so it leaves the plane perpendicular to \( v = x - \|x\|e_1 \) invariant (\( m - 1 \) eigenvalues), and flips \( v \) to \(-v\) (the \(-1\) eigenvalue). Thus we have that the eigenvalues are \(-1\) and 1 with multiplicity \( m - 1 \). We can get the singular values as the absolute values of the eigenvalues, i.e. 1 since the matrix is Hermitian, or directly since the singular values of any unitary matrix must be 1. The determinant is the product of the eigenvalues, so it is \(-1\).

10.2 Here is a listing of my routine for Householder QR:

```matlab
function [W,R]=house(A)
[m,n]=size(A);
W=zeros(m,n); %initialize W
R=A; % work on a copy of A
for k=1:n
    x=R(k:m,k);
    s=sign(x(1));
    if(s == 0) % the built in sign function returns a zero for 0 input
        s=1; % this sets the value to 1.
    end
    v=x; % it is a bit more efficient to just change the first entry than to
    v(1)=s*norm(x)+v(1); % add a multiple of e1
    v=v/norm(v);
    W(k:m,k)=v;
    R(k:m,k:n)=R(k:m,k:n)-2*v*(v'*R(k:m,k:n));
end
R=R(1:n,1:n); % the bottom rows are just 0's, so clip them out
```

Here is the listing for formQ:

```matlab
function Q=formQ(W)
[m,n]=size(W);
Q=eye(m); % initialize
for j=1:m
    for k=n:-1:1
        Q(k:m,j)=Q(k:m,j)-2*W(k:m,k)*(W(k:m,k)'*Q(k:m,j));
    end
end
```

Here is a sample test:

```matlab
>> A=randn(6,4)+i*randn(6,4);
```
\[ [W, R] = \text{house}(A); \]
\[ Q = \text{formQ}(W); \]
\[ RR = [R; \text{zeros}(2, 4)]; \]
\[ \text{norm}(Q * RR - A) \]
\[ \text{ans} = \]
\[ 2.918385327668342 \times 10^{-15} \]
\[ \text{norm}(Q * Q' - \text{eye}(6)) \]
\[ \text{ans} = \]
\[ 9.786992491369182 \times 10^{-16} \]

10.4 a) Left multiplication by the matrix \( J \) is a clockwise rotation through \( \theta \). We can write \( F = J \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \) so that \( F \) is a reflection through the \( y \)-axis followed by a rotation. That composition is itself a reflection. To see which reflection it is, we can find the eigenvector corresponding to the eigenvalue 1. It is \((s, c + 1)^*\). The orthogonal direction is spanned by \((c + 1, -s)^*\) and it is an eigenvector corresponding to the eigenvalue \(-1\). Using our geometric understanding of Householder reflections, that tells us we can reconstruct \( F \) as a Householder reflection \( I - 2v v^*/(v^* v) \) where \( v = (c + 1, -s)^* \). Note, that if we let \( c' = \cos(\theta/2) \) and \( s' = \sin(\theta/2) \) then we can rewrite \((s, c + 1) = (2c's', 2c'^2 - 1 + 1) = 2c'(s', c')\). Thus we can describe the reflection in terms of the half angle. Perhaps the best way to describe \( F \) though is as the householder reflection that takes a vector making an angle of \( \theta \) with the \( x \)-axis and reflects it onto the \( x \)-axis.

b) **Givens QR** Given \( A \in \mathbb{R}^{m \times n} \) with \( m \geq n \) the following algorithm overwrites \( A \) with \( Q^* A = R \) where \( R \) is upper triangular and \( Q \) is unitary (orthogonal):

```plaintext
for j=1:n % loop over the columns
    for i=m:-1:j+1 % loop over the rows below the diagonal, % starting at the bottom
        [c, s] = givens(A(i-1,j), A(i,j)); % compute the givens rotation % if we need to be able to reconstruct Q, % we would need to store the sequence of [c, s] pairs % (that is not done in this alg.)
        A(i-1:i, j:n) = [c -s; s c] * A(i-1:i, j:n); Apply the rotation
    end
end
```

where the function \( \text{givens} \) is defined by

```plaintext
function [c, s] = givens(a, b)
% this extracts the info needed to make the givens rotation.
r = sqrt(a^2 + b^2); % if a or b are large, this is % dangerous because there might be an overflow
if a == 0 % special case when a is zero
    c = 1;
else
    c = a / r;
end
```

6
s=-b/r;
end

This more complex algorithm avoids overflow.

function [c,s]=givens(a,b)
if (b==0)
    c=1;
    s=0;
else
    if(|b|>|a|) % choose a case so that |tau|<=1
        % to avoid possibility of overflow
        tau=-a/b;
        s=1/sqrt(1+tau^2);
        c=s*tau;
    else
        tau=-b/a;
        c=1/sqrt(1+tau^2);
        s=c*tau;
    end
end

the first givens routines take 4 flops plus a square root, call it 4 flops. The second routine is similar trading one fewer division for two comparisons.

c) The inner most loop in the main algorithm is the multiplication of the givens rotation by the portion of the two rows \( A(i-1:i, j:n) = [c-s; sc] * A(i-1:i, j:n) \). Each entry that is modified in this step involves 3 flops. Each entry will be “visited” twice – once when it is in the upper row of two, and a second time when it is in the lower row. Thus the cost is 6 flops per entry. Looking at the shape of the regions being modified we get a volume estimate (leading term) of \( n^2(m-n/3)/2 \) (same shape as for Householder) for a total cost with leading term \( 3n^2(m-n/3) \).