7. LEAST SQUARES PROBLEMS

least_squares_2013.docx

"This principle - taking many measurements to smooth out measurement errors or other random variations - is almost universal in observational and experimental sciences." ([p. 105] Heath 2002)

The material in this chapter follows the presentations of Heath (2002) and Ascher and Grief (2011).

,

7.1 PRELIMINARIES

This chapter investigates solutions to the algebraic problem

$$\min_{\beta} \left\| X \cdot \beta - y \right\|_2 ,$$

where $\|\ \|_2$, represents the Euclidean norm, both X, a m-by-n real matrix, and y, a m-by-1 real vector, are known, the unknown β is an n-by-1 vector, and $m \geq n$. Invariably in the *overdetermined* systems, where $m > n$, there exist no exact solution vectors, β, even in the absence of floating-point roundoff errors. Such algebraic problems often occur when observations are compared to models.

Random errors unavoidably crop up in all measurements. Often repeated measurements are performed to reduce *statistically* the contribution of these inevitable errors. When the number of parameters in a model, *n*, used to describe a phenomenon is less than *m*, the number of measurements performed, we call such model systems *overdetermined*. Often such model systems are linear in terms of the model parameters and are represented here in matrix-vector notation -

$$X \cdot \beta = y,$$

where X is a known m-by-n matrix ($m > n$), representing the model (to be approximated), β is an unknown n-element vector of model coefficients, and y is a known m-element vector. Usually the vector β is unlikely to reproduce the right-hand-side vector, y to floating-point accuracy, `eps` as we found for non-singular n-by-n linear systems in Chapter 4. *Least square* is the term we use to identify the category of algorithms for solving *approximately* such overdetermined linear systems. Here the modifier *approximately* means that solutions are determined by *minimizing* the *Euclidean* norms of the residual vectors, $r \equiv X \cdot \beta - y$, as functions of β. To represent the approximations inherent in least-squares solutions, usually the = symbol is replaced by ≈ symbol -

$$X \cdot \beta \approx y.$$

Existence of a least-square solution is always *guaranteed* but all solutions are not necessarily unique ([p. 109] Heath 2002). The rectangular system, $X \cdot \beta \approx y$ possesses *unique* solution if and only if X has full column rank[1], rank(X) = n ([p. 109] Heath 2002).

The least-squares approximation is frequently employed in deriving curve fits to experimental data. For example, assume that you have *m* observations of y(t),

$$(t_k, y_k) \text{ for } k = 1{:}m.$$

[1] If the rank(X) < n, one has a rank-deficient solution to $X \cdot \beta \approx b$, but the solution is not unique.

The fundamental idea of the *linear* least-squares approach is that y(t) is approximated by of a *linear* combination of n distinct basis[2] functions, here depicted by $\varphi_k(t)$, For example, a least-squares problem can be expressed in a matrix format,

$$y(t) \approx \beta_1\phi_1(t)+\beta_2\phi_2(t) + \cdots \beta_n\phi_n(t),$$

where X, a m-by-n rectangular matrix, represents the application of the basis functions at t_k for k = 1:n,

$$X_{ij} \equiv \phi_j(t_i), \text{ and } y \approx X \cdot \beta.$$

Note that the basis functions themselves can be, and usually are, *nonlinear* functions of t, but the y equations are linear in the *unknown* coefficients, β.

7.2 NORMAL EQUATIONS

How does one solve these systems of equations? Assume for the moment that matrix, X, has a full column rank. Then one approach, derived by Gauss, minimizes the *sum* of the squares of the residual, *r*,

$$X \cdot \beta - y \equiv r,$$

where

$$\beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, \; X = \begin{pmatrix} X_{11} & \cdots & X_{1n} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ \vdots & & \vdots \\ X_{m1} & \cdots & X_{mn} \end{pmatrix}, \; y = \begin{pmatrix} y_1 \\ \vdots \\ \vdots \\ \vdots \\ y_m \end{pmatrix}, \; r = \begin{pmatrix} r_1 \\ \vdots \\ \vdots \\ \vdots \\ r_m \end{pmatrix}.$$

The *square* of the Euclidean norm of the residual, Φ, a scalar, is defined as

$$\Phi \equiv \sum_{i=1}^{i=m}\left(y_i - \sum_{j=1}^{j=n} X_{ij}\beta_j \right)^2$$

The minimum of this scalar Φ is determined by setting its derivative, relative to β_k, equal to zero,

$$\frac{\partial\Phi(\beta)}{\partial\beta_k} = \sum_{i=1}^{m}\left[\left(y_i - \sum_{j=1}^{j=n} X_{ij}\beta_j\right)\left(-X_{jk}\right)\right] = 0, \quad \text{for } k = 1:n.$$

Rewriting

$$\sum_{i=1}^{i=m} X_{ik} \sum_{j=1}^{j=n} X_{ij}\beta_j = \sum_{i=1}^{i=m} X_{ik} y_i \qquad \text{for } k=1:n.$$

[2] The concept of linear independence is intimately related to the concept of a *basis*. S is termed the spanning set for a space V if every vector in V is a linear combination of the vectors in S. Thus a linear independent spanning set for the vector space V is called a *basis* for V. For example, the unit vectors {e_1, e_2, e_3} in Cartesian coordinate space ([p. 194] Meyer 2000).

In our familiar matrix-vector format this expression is expressed as

$$(X^T \cdot X) \cdot \beta = X^T \cdot y,$$

where $X^T \cdot X$ is a *non-singular square* n-by-n matrix[3] and $X^T * y$ is an n-by-1 column vector. Such equations, derived first by Gauss, are termed *normal* equations. This is the "*most widely used*" algorithm for solving full-rank least squares problems ([p. 238] Golub & van Loan 1996). When the m-by-n matrix, X, has full column rank (rank(X) = n), then $X^T \cdot X$ is *positive definite* (e.g. [p. 110], Heath 2002; [p. 143] Ascher & Chen 2011). To solve such normal equations can be solved using the techniques presented previously in chapter 4.

```
Algorithm 11.1: Least Squares via Normal Equation
                          ([p. 82] Trefthen & Bau 1997)

    •  Create matrix C = X^T·X and vector d = X^T·y
    •  Cholesky factorization L^T·L = C
    •  Solve lower triangular system, L·w = d for w
    •  Solve upper triangular system, L^T·β = w, for β
```

```
% Example 6.1 [p. 145] Ascher & Grief (2011)
>> X = [1 0 1; 2 3 5; 5 3 -2; 3 5 4; -1 6 3];
>> y = [4 -2 5 -2 1]';
>> size(X)
ans = 5     3     % 5 rows and 3 columns
>> rank(X)
ans = 3   → X has full column rank
>> C = X'*X
B = 40    30    10
    30    79    47
    10    47    55
>> d = X'*y
d = 18
     5
   -21
>> L = chol(C, 'lower')
L =     6.3246           0            0
        4.7434      7.5166            0
        1.5811       5.255       4.9885
>> w = forward_sub(L, d)
w =     2.846
       -1.1308
       -3.9205
>> beta = back_sub(L', w)
beta = 0.347226173541963
        0.399004267425320
       -0.785917496443812
>> r = (X*beta - y)'
r = -4.4387   -0.0381   -0.4950    1.8930   -1.3110
```

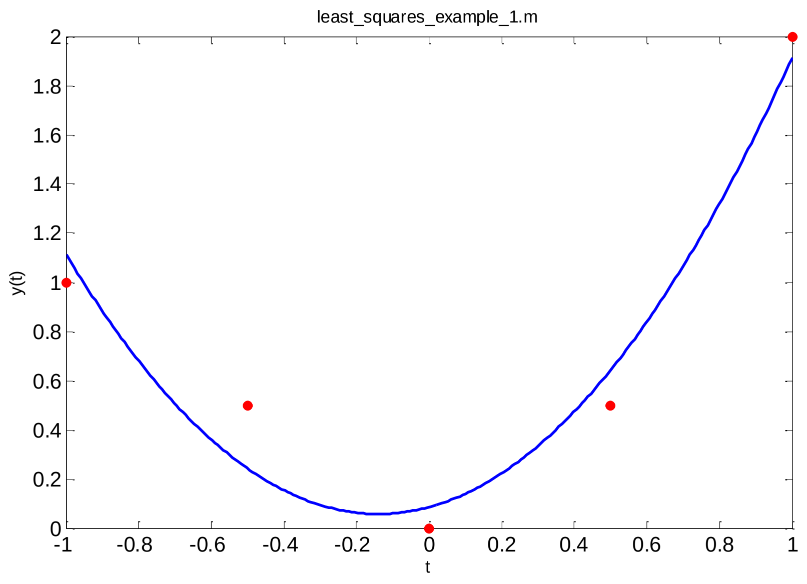
Alternatively **`linsolve`** can be used[4]. This normal-equation approach for solving a m-by-n least squares system requires a *total* of $(m + n/3)n^2$ floating-point operations (e.g. [p. 238] Golub & van Loan 1996).

[3] The n-by-n matrix, $X^T * X$, is now positive definite since it has been assumed that the *columns* of X are linear independent, rank(X) = n ([p. 110], Heath 2002).

[4] `opts.SYM = true;  opts.POSDEF = true; beta = linsolve(X'*X, X'*y, opts);`

In `least_squares_example1.m` we fit the following five data points by a three-term polynomial (three basis vectors: $\varphi_1 = 1$, $\varphi_2 = t$, $\varphi_3 = t^2$) using normal-equation approach ([p. 106] Heath 2002)

```
>> t = [-1.0 -0.5 0.0 0.5 1.0]';
>> y = [1.0   0.5 0.0 0.5 2.0]';
>> X = [ones(size(t)) t   t.^2];    % basis vectors
>> rank(X) % Unique approximate solution can be found if rank(X) = n
ans = 3    % Hence the least-squares solution is "unique"
>> Xt = X';
>> C = Xt*X
C =    5            0           2.5
       0          2.5             0
     2.5            0         2.125
>> w = Xt*y
w =    4
       1
    3.25
>> L    = chol(X'*X, 'lower');
>> w    = forward_sub(L, b);
>> beta = back_sub(L', w)
beta = 0.0857
       0.4000
       1.4286
>> norm(X*beta - y, 2)    % Euclidean norm of the least squares fit
ans = 3.3806e-001
>> tt = linspace(min(t), max(t), 250);
>> yy = @(t) (beta(1) + beta(2)*t + beta(3)*t.^2);
>> plot(t, y, 'ro', tt, yy(tt));
```



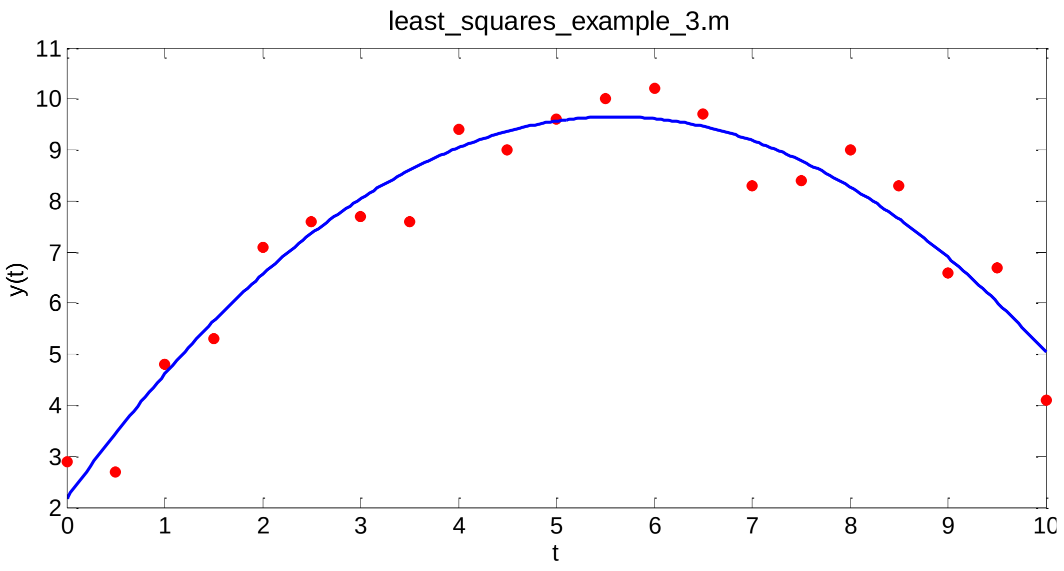
`least_squares_example2.m` Another Example ([p. 106], Heath, 2002):

A surveyor is to determine the heights of three hills above some arbitrary reference point. Sighting first from the reference point the surveyor measures x_1 = 1237 m, x_2 = 1941 m, and x_3 = 2417 m. To confirm these measurements the surveyor climbs the first hill and measures the height of the second hill, *relative* to the first, $x_2 - x_1$ = 711 m and the third hill relative the first to be $x_3 - x_1$ = 1177 m. Finally, he climbs to the top of the second hill and measures the height of the third hill above the second, $x_3 - x_2$ = 475 m.

```
  >> X = [ 1   0    0;
           0   1    0;
           0   0    1;
          -1   1    0;
          -1   0    1;
           0  -1    1];
  >> y = [1237; 1941; 2417; 711;  1177; 475];
  >> B = X'*X; w = X'*y;
>> beta = linsolve(B, w);
  beta =   1236
           1943
           2416
```

In `least_squares_example3.m`, the third example ([p. 108], Heath 2002): data is fit by a quadratic model -

```
>> t = (0:0.5:10)';
>> y = [2.9 2.7 4.8 5.3 7.1 7.6 7.7 7.6 9.4 9 9.6 10 10.2 9.7 8.3 8.4 9
        8.3 6.6 6.7 4.1]';
>> X = [ones(size(t)) t t.^2];
>> B     = X'*X;
>> w     = X'*y;
>> beta = linsolve(B, w);
         2.1757
         2.6704
       -0.23844
>> tt = linspace(min(t), max(t), 250);
>> yy = @(t) beta(1) + beta(2)*t + beta(3)*t.^2;
>> plot(tt, yy(tt), t, y, 'ro', 'MarkerSize', 10)
```



Understanding how the above fit (blue) is constructed from the noisy data points (red) is the primary goal of this chapter.

7.3 QR ALGORITHM VIA HOUSEHOLDER TRANSFORMATIONS

"a problem transformation that is legitimate theoretically is not always advisable numerically" ([p. 117] Heath 2002).

The following presentation follows closely that of Heath (2002).

Potential computational issues affect the use of the normal-equation technique. Consequently we pursue more robust algorithms to solve least square problems. Here we must employ techniques which preserve the Euclidean norms of our model systems. Gaussian elimination, the subject of chapter 4, does not, but *orthogonal* transformations do preserve Euclidean norms. Note a *square* (n-by-n) matrix, Q, is called *orthogonal* if $Q \cdot Q^T = I$, where I represents the identity matrix. Transformations produced by such orthogonal matrices conserve Euclidean norms, as can be seen when an orthogonal matrix Q multiplies an arbitrary vector u -

$$\|Q\cdot u\|_2^2 = (Q\cdot u)^T\cdot(Q\cdot u) = u^T\cdot(Q^T\cdot Q)\cdot u = u^T\cdot I\cdot u = u^T\cdot u = \|u\|_2^2.$$

Orthogonal matrices modify vectors via rotations or reflections but the Euclidean length of the vectors remain unchanged.

As with our study of n-by-n linear systems in chapter 4 the ultimate goal of the orthogonal transformations of an overdetermined m-by-n system, $X \cdot \beta \approx y$, is an n-by-n upper triangular matrix, R, such that

$$\begin{bmatrix} R \\ O \end{bmatrix}\cdot\beta \approx \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$$

Consequently, the residual vector becomes

$$\|r\|_2^2 = \|c_1 - R\cdot\beta\|_2^2 + \|c_2\|_2^2.$$

Note that the first element in the above sum can be required to be zero by letting β be the solution of the triangular linear system,

$$R\cdot\beta = c_1.$$

The second element of the above sum does not depend on β and, consequently, the minimum sum of the square residuals becomes

$$\|r\|_2^2 = \|c_2\|_2^2.$$

Orthogonal transformation of an m-by-n matrix X to an n-by-n upper triangular matrix R is performed by what is termed a QR factorization -

$$X = Q\cdot\begin{bmatrix} R \\ O \end{bmatrix},$$

where Q is an m-by-m orthogonal matrix, and O is an (m-n)-by-n matrix of zeros. Thus $X \cdot \beta \approx y$ becomes

$$\|r\|_2^2 = \|y - X \cdot \beta\|_2^2 = \left\| y - Q\begin{bmatrix} R \\ O \end{bmatrix} \cdot \beta \right\|_2^2 = \left\| Q^T \cdot y - \begin{bmatrix} R \\ O \end{bmatrix} \cdot \beta \right\|_2^2 = \|c_1 - R \cdot \beta\|_2^2 + \|c_2\|_2^2,$$

where

$$Q^T \cdot y = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix},$$

and c_1 is n-by-1 sized vector and c_2 is (m-n)-by-1 sized vectors. Thus β is the solution of the n-by-n sized triangular system, $R \cdot \beta = c_1$ and the residual norm, $||r||_2$ becomes $||c_2||_2$. Three orthogonalization approaches (QR factorizations) are ordinarily employed:

- Householder reflections
- Givens rotations
- Gram-Schmidt orthogonalization

7.3.1 Householder Transformations (elementary reflections)

To generate the n-by-n upper triangular matrix, R, the Householder algorithm applies a sequence of orthogonal matrices applied to the m-by-n matrix X

$$Q_n \cdot Q_{n-1} \cdots Q_2 \cdot Q_1 \cdot X = R,$$

where the succession of orthogonal matrices is itself a orthogonal matrix -

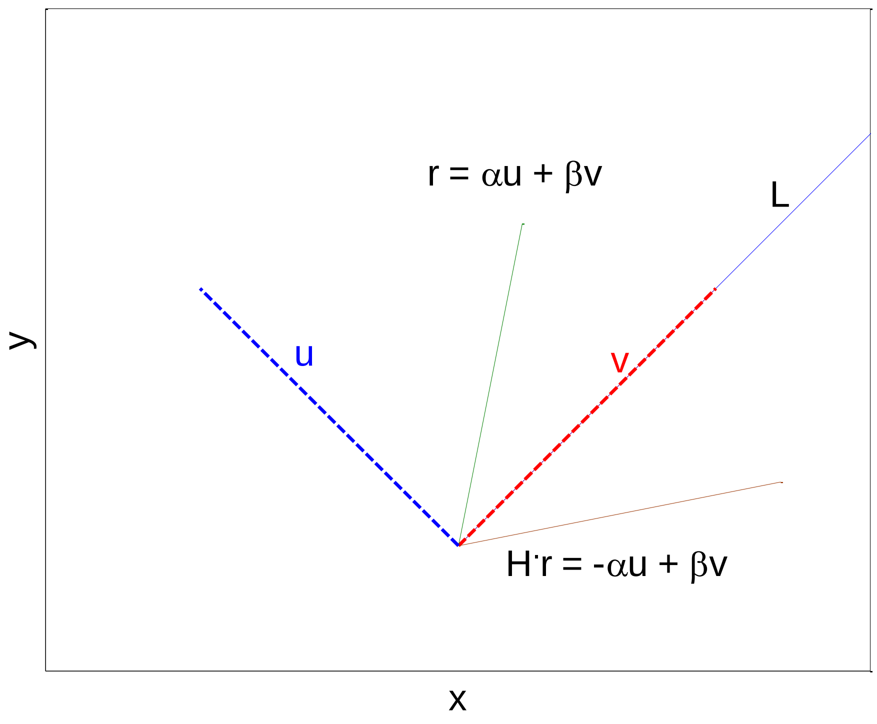
$$Q^T = Q_n \cdot Q_{n-1} \cdots Q_2 \cdot Q_1 \;\Rightarrow\; Q = Q_1^T \cdot Q_2^T \cdots Q_{n-1}^T \cdot Q_n^T,$$

and, consequently, $X = Q \cdot R$. The action of each Q_k creates zeros in the kth column below the diagonal while maintaining all prior zero elements (e.g., [p. 69] Trefethen & Bau 1997). Following Trefethen and Bau the result of a sequence of Q_k on 5-by-3 sized matrix, X, is illustrated below where x represents a nonzero element, $\color{red}{x}$ represents a modified value -

$$\underset{X}{\begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{bmatrix}} \overset{Q_1}{\rightarrow} \underset{Q_1 \cdot X}{\begin{bmatrix} \color{red}{x} & \color{red}{x} & \color{red}{x} \\ 0 & \color{red}{x} & \color{red}{x} \\ 0 & \color{red}{x} & \color{red}{x} \\ 0 & \color{red}{x} & \color{red}{x} \\ 0 & \color{red}{x} & \color{red}{x} \end{bmatrix}} \overset{Q_2}{\rightarrow} \underset{Q_2 \cdot Q_1 \cdot X}{\begin{bmatrix} \color{red}{x} & \color{red}{x} & \color{red}{x} \\ 0 & \color{red}{x} & \color{red}{x} \\ 0 & 0 & \color{red}{x} \\ 0 & 0 & \color{red}{x} \\ 0 & 0 & \color{red}{x} \end{bmatrix}} \overset{Q_3}{\rightarrow} \underset{Q_3 \cdot Q_2 \cdot Q_1 \cdot X}{\begin{bmatrix} \color{red}{x} & \color{red}{x} & \color{red}{x} \\ 0 & \color{red}{x} & \color{red}{x} \\ 0 & 0 & \color{red}{x} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}$$

The action of Q_1 creates zeros at locations: (2, 1), (3, 1), (4, 1), and (5, 1); Q_2 creates zeros at: (3, 2), 4, 2), and (5, 2); Q_3 generates zeros at (4, 3) and (5, 3) without affecting the previously generated zero values.

How does a Householder reflection create zeros in a column while preserving its length (Euclidean norm)? This is best seen through the action of a vector reflection in a two dimensions. Following Watkins ([p. 202]



2002) let L be a non-zero vector passing through the origin and v is a unit vector aligned along v and u is a unit vector transverse to v. These vectors {v, u} form a Cartesian basis such that every two-dimensional rector, r, can be represented as a linear combination of the vectors, {u, v} and $r \equiv \alpha u + \beta v$ and the reflection of the vector r through L is $-\alpha u + \beta v$. Thus the reflection matrix H must satisfy $H \cdot r = -\alpha u + \beta v$ for all α and β and thus H matrix must satisfy

$$H \cdot u = -u \qquad \text{and} \quad H \cdot v = v$$

([p. 194] Watkins 2002). Note[5] that the use of a matrix, $P = u \cdot u^T$, $P \cdot u = (u \cdot u^T) \cdot u = u \cdot (u^T \cdot u) = u \|u\|_2^2 = u$ and $P \cdot v = 0$ creates a result similar to what is required, but setting $H = I - 2 \cdot P$, where I is the identity matrix one now does produce the required matrix, $H = I - 2u \cdot u^T$. When $\|u\|_2^2 \neq 1$, for n-by-1 vectors, u, the n-by-n Householder matrix becomes

$$H = I - \frac{2u \cdot u^T}{\|u\|_2}.$$

([p. 209] Golub & van Loan 1996). The matrix H is orthogonal and symmetric: $H = H^T = H^{-1}$.

Vector lengths are unchanged, when multiplied by Householder matrices, *H*, then such multiplications can used to eliminate sub-diagonal elements of matrix column while preserving its Euclidean norm. Given a vector, *x*, we can determine a corresponding vector *u* which forms a Householder matrix, *H*, such that all the *x* components, but the first element are "annihilated" by the multiplication of *x* by the Householder matrix. The appropriate value for *u* follows from given a column vector x one requires that $H \cdot x$ is a multiple of e_1, where x(2:n) gets zeroed out, -

[5] Observe for an n-by-1 vector that $u \cdot u^T$ is an *outer* product, creating an n-by-n matrix, but $u^T \cdot u$ is inner product generating a scalar.

$$\alpha e_1 \equiv H\cdot x = \left(I - 2\frac{u\cdot u^T}{u^T\cdot u}\right)\cdot x = x - \frac{2u^T\cdot x}{u^T\cdot u}u,$$

Setting $u = x + \alpha e_1$ implies $\Rightarrow u^T\cdot x = x^T\cdot x + \alpha x_1$ and

$$u^T\cdot u = x^T\cdot x + 2\alpha x_1 + \alpha^2 \Rightarrow$$

$$H\cdot x = x - \frac{2u^T\cdot x}{u^T\cdot u}u = x - \frac{2u^T\cdot x}{u^T\cdot u}(x+\alpha e_1) = x - \frac{2u^T\cdot x}{u^T\cdot u}x - 2\alpha\frac{u^T\cdot x}{u^T\cdot u}e_1$$

$$H\cdot x = \left(1 - 2\frac{x^T\cdot x + \alpha x_1}{x^T\cdot x + 2\alpha x_1 + \alpha^2}\right)x - 2\alpha\frac{u^T\cdot x}{u^T\cdot u}e_1$$

(e.g., [p. 209] Golub & van Loan 1996). The coefficient of x, (the parenthesis above) must be zero, in order for $H\cdot x$ be a multiple of e_1. This results if $\alpha = \pm||x||_2$ and $x^T\cdot x = \alpha^2$, then

$$1 - 2\frac{x^T\cdot x + \alpha x_1}{x^T\cdot x + 2\alpha x_1 + \alpha^2} = 1 - 2\frac{\alpha^2 + \alpha x_1}{2\left(\alpha^2 + \alpha x_1\right)} = 0,$$

and, consequently,

$$u = x \pm \|x\|_2 e_1 \Rightarrow H\cdot x = (I - 2\frac{u\cdot u^T}{u^T\cdot u})x = \pm\|x\|_2 e_1.$$

Several "*important practical details*" occur in computing Householder reflections ([p. 210] Golub & van Loan). Care must be taken in choosing the sign of $||x||_2$, otherwise severe subtractive cancellation ensues. Moreover, in practice it is useful to introduce u(1) ≡ 1.0. Consequently, we employ Golub van Loan's Algorithm 5.1.1, **house**, to compute the Householder transformation

Illustrated below is an example Householder reflection (Example 3.7, [p. 122] Heath 2002) -

```
>> x = [2 1 2]';
>> norm(x, 2)              % Euclidean norm
ans = 3
>> [u, β] = house(x); % Employing Golub & van Loan (1996) H format
Here β = 2/(u'*u)
>> H = eye(length(x)) - 2*u*u'/(u'*u)
H =    0.66667      0.33333      0.66667
       0.33333      0.66667     -0.66667
       0.66667     -0.66667     -0.33333
>> H*x
ans =                  % equals -3 to within few eps
            3
  2.2204e-016
  1.1102e-016
>> isequal(H, H')                    % example H is symmetric
ans = 1
>> isequal(H*H', eye(length(x)))     % example H is orthogonal
ans =  0                             % NOTE
>> max(max(abs(H*H' - eye(length(x)))))
ans =  1.1102e-016                   % difference from orthogonal < 0.5*eps
```

```
   function [u, beta] = house(x)
 % function [u, beta] = house(x)
 % INPUT
```

```
%           x  = (n by 1) column vector
% OUTPUT
%           u    = (n by 1) column vector
%           beta = 2/(uT*u)
% Given vector, x,  house computes u such that H = In - beta*u*u^T and
% Px = |||x||_2*e_1
% Matrix Computations, 3rd Ed.
% Golub, J H & van Loan, C F
% Algorithm 5.1.1
% (Johns Hopkins University Press, 1996), p. 210

    if ~iscolumn(x);
      disp('input is NOT column vector');
      u = inf; beta = nan;
      return;
    end
  n      = length(x);
  sigma  = x(2:n)'*x(2:n);
  u      = [1; x(2:n)];
    if sigma == 0.0
        beta = 0.0;
    else
        mu   = sqrt(x(1)^2 + sigma);
          if x(1) <= 0.0
              u(1) = x(1) - mu;
          else
            u(1) = -sigma/(x(1) + mu);
          end
        beta = (2.0*u(1)^2)/(sigma + u(1)^2);
        u    = u/u(1);
    end
  end
```

The QR algorithm for the factorization of an m-by-n matrix, *A*, by successive Householder transformations following the illustration on pg. 9 is illustrated below -

```
      function [Q, R] =  qr_brute_force(A)
        % A     m-by-n matrix
        % Q     m-by-m orthogonal matrix
  function [Q, R] = house_QR_brute_force(A)
 %function [Q, R] = house_QR_brute_force(A)
 % INPUT:
 %        A     m-by-n matrix
 % OUTPUT:
 %        Q     m-by-m orthogonal matrix
 %        R     m-by-n upper triangular matrix
 %
 % Applies Householder transformations to remove
 %   subdiagonals Hn*Hn-1 ...H3*H2*H1*A = R
 %   Q^T = Qn*Qn-1 ... Q3*Q2*Q1 => Q = (Q^T)^T
 % Contains essential features of qr factorization BUT
 %   is very inefficient: ~nm^3 operations

      [m,n]  = size(A);
      I      = eye(m);
```

```
        Qt      = I;
            for j = 1:n
              H               = I;
              [u, beta]       = house( A(j:m, j) );
              H(j:m, j:m)     = eye(length(u)) - beta*u*u';
              A               = H*A;
              Qt              = H*Qt;
            end
        Q = Qt';
        R = A;
  end
```

"It is usually not necessary to compute Q explicitly" ([p. 212] Golub & van Loan 1996).

Thus an algorithm, more efficient that the above one, can be implemented which, first, overwrites the upper triangular portion of the m-by-n matrix A by the n-by-n upper triangular matrix, R ([p. 224] Golub & van Loan 1996). Second, the nonzero elements (j+1:m) of the jth Householder vector,

$$u^{(j)} = [\underbrace{0, \ldots .0,}_{j-1} 1, u^{(j)}_{j+1}, u^{(j)}_{j+2}, \ldots, u^{(j)}_{m}]^T$$

overwrites the jth subdiagonal column of A, A(j+1:m, j). This algorithm is implemented below -

```
       function [A, beta] = house_QR(A)
     % function [A, beta] = house_QR(A)
      % INPUT
      %          A   = (m-by-n) matrix
      % OUTPUT
      %          A    = m-by-n) matrix OVERWRITTEN
      % upper triangular part of A overwritten by upper triangle R
      % j+1:m components of jth House vector stored in A(j+1:m, j) j < m
      %          beta = (n-by-1) vector
      % Matrix Computations, 3rd Ed.
      % Golub, J H & van Loan, C F
      % Algorithm 5.2.1
      % (Johns Hopkins University Press, 1996), p. 224
          [m, n] = size(A);
          beta   = zeros(n, 1);
              for j = 1:n
                 [u, beta(j)]   = house(A(j:m, j));
                  A(j:m, j:n)   = (eye(m - j + 1) - beta(j)*u*u')* A(j:m,
     j:n);
                  if j < m; A(j + 1:m, j) = u(2:m - j + 1);  end
              end
       end
```

Upon the completion of this algorithm a 6-by-4 rectangular matrix, A, is overwritten by

$$A = \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{14} \\ u_2^{(1)} & R_{22} & R_{23} & R_{23} \\ u_3^{(1)} & u_3^{(2)} & R_{33} & R_{34} \\ u_4^{(1)} & u_4^{(2)} & u_4^{(4)} & R_{44} \\ u_5^{(1)} & u_5^{(2)} & u_5^{(4)} & u_5^{(5)} \\ u_6^{(1)} & u_6^{(2)} & u_6^{(4)} & u_6^{(5)} \end{bmatrix}.$$

This algorithm requires $2n^2(m - n/3)$ floating-point operations; if Q is to be computed explicitly, a factor of two more floating-point operations, $4(m^2n - mn^2 + n^3/3)$ are necessary ([p. 225] Golub & van Loan 1996). Below is an extension of the above **`house_QR.m`**, which computes both Q and R.

```
  function [Q, R] = house_full_QR(A)
% function [Q, R] = house_full_QR(A)
% INPUT
%          A    m-by-n matrix
% OUTPUT:
%         Q     m-by-m orthogonal matrix
%         R     m-by-n upper triangular matrix

% Matrix Computations, 3rd Ed.
% Golub, J H & van Loan, C F

 [m, n]     = size(A);
 [A, beta] = house_QR(A);  % jch version of GvL 5.2.1
 R          = triu(A);
 I          = eye(m,m);
 Q          = I;

 % jch version of GvL equation 5.1.5
     for j = n:-1:1
         u(j:m, 1)  = [1.0; A(j+1:m, j)];
        Q(j:m, j:m) = (I(j:m, j:m) - beta(j)*u(j:m)*u(j:m)')*Q(j:m, j:m);
     end
  end
```

We have computed the QR factorization of A. To solve the general least squares problem one must transform the right-hand-side, b, by an identical sequence of Householder transformations. Thus

```
  function beta = solution_QR(A, b)
% function beta = solution_QR(A, b)
% INPUT:
%         A = [m by n] array
%         b   [m x 1 ] vector
% OUTPUT:
%         beta = [n x 1 ] vector
```

```
%
% QR factorization of a using Householder Transformations

      [m, n] = size(A);
      [Q, R] = qr_brute_force(A);
      b      = Q'*b;
      beta   = back_sub(R(1:n, 1:n), b(1:n));
 end


   function beta = house_least_squares_sol(A, b)
 % function beta = house_least_squares_sol(A, b)
  % INPUT
  %         A  = (m-by-n) matrix
  % OUTPUT
  %         A   = m-by-n) matrix OVERWRITTEN
  % upper triangular part of A overwritten by upper triangle R
  % j+1:m components of jth House vector stored in A(j+1:m, j) j < m
  %         beta = (n-by-1) vector
  % Matrix Computations, 3rd Ed.
  % Golub, J H & van Loan, C F
  % Algorithm 5.3.2
  % (Johns Hopkins University Press, 1996), p. 240
   [A, beta] = house_QR(A);
   [m, n]    = size(A);
       for j = 1:n
          u(j, 1)       = 1;
          u(j + 1:m, 1) = A(j + 1:m, j);
          b(j:m)        = (eye(m-j+1)-beta(j)*u(j:m)*u(j:m)')*b(j:m);
       end
    beta = back_sub(A(1:n, 1:n), b(1:n));
 end
```

For the full rank problem **house_least_squares_sol.m** requires $2n^2(m - n/3)$ floating-point operations; the O(nm) operations associated with updating b and the $O(n^2)$ operations associated with back substitution are not significant with the number of operations related to factoring the m-by-n sized matrix A ([p. 240] Golub & van Loan 1996).

```
>> solution_QR(X, y)
ans =0.085714
          0.4
       1.4286
>> house_least_squares_sol(X, y)
ans =0.085714
          0.4
       1.4286
```

least_squares_example2.m:

$$A\bullet x = \begin{bmatrix} +1 & +0 & +0 \\ +0 & +1 & +0 \\ +0 & +0 & +1 \\ -1 & +1 & +0 \\ -1 & +0 & +1 \\ +0 & -1 & +1 \end{bmatrix} \bullet \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1237 \\ 1941 \\ 2417 \\ 711 \\ 1177 \\ 475 \end{bmatrix} = b.$$

```
>> A = [ 1 0 0; 0 1 0; 0 0 1; -1 1 0; -1 0 1; 0 -1 1]
A =   1      0      0
      0      1      0
      0      0      1
     -1      1      0
     -1      0      1
      0     -1      1
>> b =[1237 1941 2417 711 1177 475]'
b =     1237
        1941
        2417
         711
        1177
         475
>> x = solution_QR(A, b)
x =      1236
         1943
         2416
```

`least_squares_example3.m`

```
>> t = (0:0.5:10)';
>> y = [2.9 2.7 4.8 5.3 7.1 7.6 7.7 7.6 9.4 9 9.6 10 10.2 9.7 8.3 8.4 9 8.3 6.6 6.7
4.1]';
>> X = [ones(size(t)) t t.^2];
>> x = solution_QR(X, y)
x =  2.1757
     2.6704
    -0.23844      % Agrees with normal equation calculation
```

7.4 BUILT-IN MATLAB PROCEDURES

7.4.1 **`polyval`** and **`polyfit`**

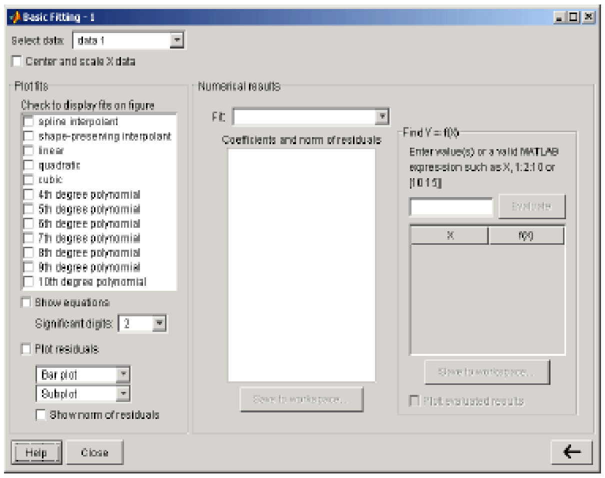
Built-in MATLAB functions - *Least squares* polynomial fits created through **`polyfit & polyval`**

```
>> help polyfit
POLYFIT Fit polynomial to data.
P = polyfit(X, Y, N) finds the coefficients of a polynomial P(X) of degree N that
fits the data Y best in a least-squares sense.   P is a row vector of length N+1
containing the  polynomial coefficients in descending powers, P(1)*X^N + P(2)*X^(N-1) +
...+ P(N)*X + P(N+1).

>>  help polyval
POLYVAL Evaluate polynomial.
Y  =  polyval(P,X) returns the value of a polynomial P evaluated at X. P is a vector
of length N+1 whose elements are the coefficients  of the  polynomial in  descend-
```

```
ing powers;  Y = P(1)*X^N + P(2)*X^(N-1) + ... + P(N)*X + P(N+1).  If X is a matrix or
vector, the polynomial is evaluated at all points in X.

% Previous least_squares_example1.m:
>> t = [-1.0 -0.5 0.0 0.5 1.0]'; y = [1.0 0.5 0 0.5 2.0]';
>> polyfit(t, y, 2)    % Assume again the model three-term polynomial
ans =  1.4286    0.4000    0.0857  % Same value as x in the first example
```

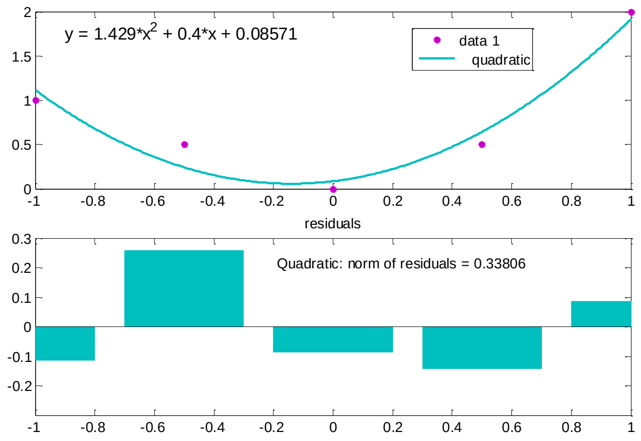
7.4.2 Basic Fitting Interface *Most important*

Basic Fitting tab exists under Tools icon on plot header/banner:
Fit data as least-squares polynomials up to order 10.
Can plot *multiple* fits for single data set.
Plots residuals.
Fit with cubic splines[6] & shape-preserving cubics (PCHIP).
Examine numerical values of the fits.
Saves fit parameters to workspace.



```
>> t = [-1.0 -0.5 0.0 0.5 1.0]';  % Create vectors in command window
>> y = [1.0   0.5 0.0 0.5 2.0]';
>> plot(t, y, 'mo');
      Under Tools tab → click Basic Fitting
      Under Check to display fits on figures → click quadratic
      Click box → show equations
      Significant digits  → click 4
      Click → Plot residuals
      Click box → show norm of residuals
```

[6] A spline is a piecewise polynomial subject to constraints at boundaries. For a cubic spline, the values of the polynomial, its first derivative, and second derivative are continuous between segments.



7.5 MATLAB's BACKSLASH OPERATOR

"The fundamental tool for solving linear systems of equations is the backslash operator, \." ([p. 123] Higham & Higham 2005)

MATLAB's backslash operator can solve three types of linear systems, A·x = b, for given matrices A and b. The three types of systems are delineated by the structure of A: square (n-by-n), overdetermined (m-by-n, with m > n), and underdetermined (m-by-n, with n < m).

Square Systems

If A is an n-by-n non-singular matrix `A\b` produces the solution vector, x, calculated via LU factorization with partial pivoting. During the factorization the backslash operator computes the reciprocal of the condition number, `rcond(A)`, and generates a warning if `rcond` ≤ `eps`.

However MATLAB identifies the following specific matrices and employs additional algorithms to increase computational efficiency.

- Triangular matrices (upper or lower) and their permutations. Such systems are computed via substitution.
- Positive definite matrices. Cholesky factorization (`chol`) is employed instead of LU factorization. If a symmetric matrix, *A*, has positive diagonal elements, MATLAB initiates a Cholesky factorization. If it fails, a LU factorization is employed.
- Upper Hessenberg matrix. An LU factorization, tailored to the Hessenberg matrix structure is used to factor A.

Overdetermined Systems

If A is an m-by-n matrix with m > n matrix `A\b` computes a least-squares solution vector, x via an orthogonal-triangular factorization, via Householder transformations –

$$A \cdot P = Q \cdot R,$$

where *P* is a permutation matrix of the identity matrix, *Q* is orthogonal, and *R* is upper triangular matrix. The least-squares solution x is calculated via

$$x = P \bullet (R \backslash (Q' \bullet B)).$$

Using the parameters from **`least_squares_example1.m`**, the backslash operator reproduces the solution derived earlier by the normal[7] equation:

```
    >> t = [-1.0 -0.5 0.0 0.5 1.0]';
    >> y = [1.0   0.5 0.0 0.5 2.0]';
    >> X = [ones(size(t)) t   t.^2];
    >> x = X\y
    x =  0.085714
              0.4
           1.4286
```

Underdetermined Systems

If A is an m-by-n matrix with m < n matrix **`A\b`** computes a solution vector, x, with j nonzero terms where j is the **`rank(A)`** -

```
>> A = randi(100, 2, 3)
A =
    82    13    64
    91    92    10
>> b = [3; 4]
b =  3
     4
>> A\b
ans =
     0.035215
    0.0086464
            0
```

7.7 REFERENCES:

Ascher, U. M., & Greif, C.2011, *A First Course in NUMERICAL METHODS* (SIAM: Philadelphia).

Datta, B. S. 1995, *Numerical Linear Algebra and Applications* (Brooks/Cole: Pacific Grove, CA).

Golub, G. H., & van Loan, C. F. 1996, *Matrix Computations, 3rd Ed.* (Johns Hopkins: Baltimore).

Heath, M. T. 2002, *Scientific Computing: An Introductory Survey*, 2nd Ed. (McGraw-Hill: NY).

Higham, D. J. & Higham, N. J. 2005, *MATLAB Guide, 2nd Ed.* (SIAM: Philadelphia).

Meyer, C. D. 2000, Matrix Analysis and Applied Linear Algebra (SIAM: Philadelphia).

Moler, C. 2004, *Numerical Computing with MATLAB* (SIAM: Philadelphia).

Quarteroni, A., & Saleri, F. 2003, *Scientific Computing with MATLAB* (Springer-Verlag: Berlin).

[7] Higham & Higham ([p. 125]) note that the least-squares solution can be computed via **`pinv(X)*y`**.

Sauer, T. 2006, *Numerical Analysis* (Pearson, Addison-Wesely: Boston).

Trefethen, L. N. & Bau, D. 1997, *Numerical Linear Algebra* (SIAM: Philadelphia).

Van Loan, C. F. 2000, *Introduction to Scientific Computing: A Matrix-Vector Approach Using MATLAB*, (Prentice-Hall: Upper Saddle River, NJ).

5.8 QUESTIONS

1. True or false: A linear least squares problem always has a solution.

2. True or false: Fitting a straight line to a set of data points is a linear least squares problem, whereas fitting a quadratic polynomial to the data is a nonlinear least squares problem.

3. True or false: An overdetermined linear least squares problem $Ax \approx b$ always has a unique solution x that minimizes the Euclidean norm of the residual vector $r = b - Ax$.

4. True or false: In solving a linear least squares problem $Ax \sim b$, if the residual is 0, then the solution x must be unique.

5. True or false: If the n x n matrix Q is a Householder transformation, and x is an arbitrary n-vector, then the last k components of the vector Q^*x are zero for some $k < n$.

6. True or false: Methods based on orthogonal factorization are generally more expensive computationally than methods based on the normal equations for solving linear least squares problems.

7. In a linear least squares problem $Ax \sim b$, where A is an m X n matrix, if rank(A) < n, then which of the following situations are possible?
(a) There is no solution.
(b) There is a unique solution.

(c) There is a solution, but it is not unique.

8. In an overdetermined linear least squares problem with model function $f(t, x) = x_1\varphi_1(t) + x_2\varphi_2(t) + x_3\varphi_3(t)$, what will be the rank of the resulting least squares matrix A if we take $\varphi_1(t) = 1$, $\varphi_2(t) = t$, and $\varphi_3(t) = 1 - t$?

9. What is the system of normal equations for the linear least squares problem $Ax \sim b$?

10. List two ways in which use of the normal equations for solving linear least squares problems may suffer loss of numerical accuracy.

11. Let A be an m x n matrix. Under what conditions on the matrix A is the matrix A^T*A
(a) Symmetric?
(b) Nonsingular?
(c) Positive definite?

12. Which of the following properties of an m x n matrix A, with m > n, indicate that the minimum residual solution of the least squares problem $Ax \sim b$ is *not* unique?
(a) The columns of A are linearly dependent.
(b) The rows of A are linearly dependent.
(c) The matrix A^T*A is singular.

13. (a) Can Gaussian elimination with pivoting be used to compute an LU factorization of a rectangular m x n matrix A, where L is an m X k matrix whose entries above its main diagonal are

all zero, U is a k x n matrix whose entries below its main diagonal are all zero, and k = min{m, n}?
(b) If this were possible, would it provide a way to solve an overdetermined least squares problem $Ax \sim b$, where m > n? Why?

14. (a) What is meant by two vectors x and y being orthogonal to each other?
(b) Prove that if two nonzero vectors are orthogonal to each other, then they must also be linearly independent.
(c) Give an example of two nonzero vectors in the plane that are orthogonal to each other.

15. What is meant by an orthogonal projector? How is this concept relevant to linear least squares?

16. (a) Why are orthogonal transformations, such as Householder or Givens, often used to solve least squares problems?
(b) Why are such methods not often used to solve square linear systems?
(c) Do orthogonal transformations have any advantage over Gaussian elimination for solving square linear systems? If so, state one.

17. Which of the following matrices are orthogonal?

a) $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$,

b) $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$,

c) $\begin{bmatrix} 2 & 0 \\ 0 & 1/2 \end{bmatrix}$,

d) $\begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 \\ -\sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$.

18. Which of the following properties does an n x n orthogonal matrix necessarily have?
   (a) It is nonsingular.
   (b) It preserves the Euclidean vector norm when multiplied times a vector.
   (c) Its transpose is its inverse.
   (d) Its columns are orthonormal.
   (e) It is symmetric.
   (f) It is diagonal.
   (g) Its Euclidean matrix norm is 1.
   (h) Its Euclidean condition number 1.

19. Which of the following types of matrices are necessarily orthogonal?
   (a) Permutation
   (b) Symmetric positive definite
   (c) Householder transformation
   (d) Givens rotation
   (e) Nonsingular
   (f) Diagonal.

20. If a vertical beam has a downward force applied at its lower end, the amount by which it stretches will be proportional to the magnitude of the force. Thus, the total length *y* of the beam is given by the equation

$$y = x_1 + x_2 t,$$

where x_1 is its original length, t is the force applied, and x_2 is the proportionality constant. Suppose that the following measurements are taken:

t	10	15	20
y	11.60	11.85	12.25

(a) Set up the overdetermined 3 x 2 system of linear equations corresponding to the data collected.
(b) Is this system consistent? If not, compute each possible pair of values for (x_1, x_2) obtained by selecting any two of the equations from the system. Is there any reason to prefer anyone of these results?
(c) Set up the system of normal equations and solve it to obtain the least squares solution to the overdetermined system. Compare your result with those obtained in part b.

21. Suppose you are fitting a straight line to the three data points (0,1), (1,2), (3,3).
   (a) Set up the overdetermined linear system for the least squares problem.
   (b) Set up the corresponding normal equations.

(c) Compute the least squares solution by Cholesky factorization.

22. Set up the linear least squares system *Ax*~ *b* for fitting the model function $f(t, x) = x_1t+x_2e^t$ to the three data points (1,2), (2,3), (3,5).

23. In fitting a straight line $y = x_0 +x_1t$ to the three data points (t_i, Y_i) = (0,0), (1,0), (1,1), is the least squares solution unique? Why?

24. (a) What is the Euclidean norm of the minimum residual vector for the following linear least squares problem? (b) What is the solution vector x for this problem?

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cong \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}.$$

25. In Section 5.2 we observed that the cross-product matrix A^T*A is exactly singular in floating-point arithmetic if

$$A = \begin{bmatrix} 1 & 1 \\ \varepsilon & 0 \\ 0 & \varepsilon \end{bmatrix},$$

where ε is a positive number smaller than √eps in a given floating-point system. Show that if A = QR is the QR factorization for this matrix for this matrix A*,* then *R* is *not* singular, even in floating-point-arithmetic.

26. 3.1. For *n* = 0, 1, ... ,5, fit a polynomial of degree *n* by least squares to the following data:

t|	0.0	1.0	2.0	3.0	4.0	5.0
y|	1.0	2.7	5.8	6.6	7.5	9.9

Make a plot of the original data points along with each resulting polynomial curve. Which polynomial would you say captures the general trend of the data better? Obviously this is a subjective question and its answer depends both on the nature of the given data (e.g. the uncertainty of the data values) and the purpose of the fit. Explain your assumptions in answering.

26) (a) Solve the following least squares problem using any method you like:

$$\begin{bmatrix} 0.16 & 0.10 \\ 0.17 & 0.11 \\ 2.02 & 1.29 \end{bmatrix} \bullet \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cong \begin{bmatrix} 0.26 \\ 0.28 \\ 3.31 \end{bmatrix}.$$

(b) Now solve the same least squares problem again, but this time use the slightly perturbed right-hand side

$$\begin{bmatrix} 0.27 \\ 0.25 \\ 3.33 \end{bmatrix}$$

(c) Compare your results from parts *a* and *b.* Can you explain this difference?

27) (a) Evaluate the function

$$f(t) = .05 \sin(l000t) + 0.5 \cos(\pi t) - .4\sin(10t)$$

at the 101 points given by 0:.01:1. Plot the result broken line.

(b) In order to study the slow scale *trend* of this function, we wish to find a low degree polynomial (degree at most 6) that best approximates *f* in the least squares sense at the above 101 data points. By studying the figure from part (a) find out the smallest *n* that would offer a good fit in that sense. (c) Find the best approximating polynomial of degree *n* and plot it together with *f.* What are your observations?

28) Let us synthesize data in the following way. We start with the cubic polynomial

$$q(t) = -11 + \frac{53}{3}t - \frac{17}{2}t^2 + \frac{7}{6}t^3.$$

Thus, *n* = 4. Sample q(t) at 33 equidistant points between 0.9 and 4.1. Then we add to these values 30% noise using the random number generator `randn` in MATLAB, to obtain "*the data*" which the approximations that you will construct "see." From here on, no knowledge of *q(t),* its properties, or its values anywhere is assumed: we pretend we don't know.

Your programming task is to pass through these data two approximations:

(a) An interpolating polynomial of degree 32. This can be done using the MATLAB function `polyfit`.

(b) An interpolating cubic spline using the MATLAB function `spline`. The corresponding method is described later in the course, but you don't need to rush and study that right now.

(c) Plot the data and the obtained approximations. Which of these approximations make sense? Discuss.