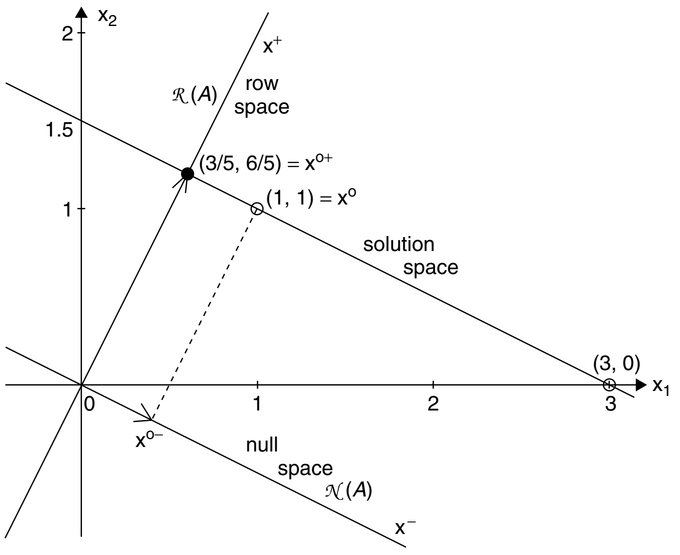

Figure 2.1 A minimum-norm solution.

and any vector in the null space of A can be expressed by Eq. (2.1.4) as

$$A\mathbf{x}^- = [\,1 \quad 2\,]\begin{bmatrix} x_1^- \\ x_2^- \end{bmatrix} = 0; \qquad x_2^- = -\frac{1}{2}x_1^- \tag{E2.1.4}$$

We use Eq. (2.1.7) to obtain the minimum-norm solution

$$\mathbf{x}^{o+} = A^T[AA^T]^{-1}\mathbf{b} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}\left([\,1 \quad 2\,]\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)^{-1} 3 = \frac{3}{5}\begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 1.2 \end{bmatrix} \tag{E2.1.5}$$

Note from Fig. 2.1 that the minimum-norm solution $\mathbf{x}^{o+}$ is the intersection of the solution space and the row space and is the closest to the origin among the vectors in the solution space.

2.1.3 The Overdetermined Case ($M > N$): LSE Solution

If the number (M) of (independent) equations is greater than the number (N) of unknowns, there exists no solution satisfying all the equations strictly. Thus we try to find the LSE (least-squares error) solution minimizing the norm of the (inevitable) error vector

$$\mathbf{e} = A\mathbf{x} - \mathbf{b} \tag{2.1.8}$$

Then, our problem is to minimize the objective function

$$J = \tfrac{1}{2}\|\mathbf{e}\|^2 = \tfrac{1}{2}\|A\mathbf{x} - \mathbf{b}\|^2 = \tfrac{1}{2}[A\mathbf{x} - \mathbf{b}]^T[A\mathbf{x} - \mathbf{b}] \tag{2.1.9}$$

whose solution can be obtained by setting the derivative of this function (2.1.9) with respect to **x** to zero.

$$\frac{\partial}{\partial \mathbf{x}} J = A^T[A\mathbf{x} - \mathbf{b}] = \mathbf{0}; \qquad \mathbf{x}^o = [A^T A]^{-1} A^T \mathbf{b} \tag{2.1.10}$$

Note that the matrix A having the number of rows greater than the number of columns ($M > N$) does not have its inverse, but has its left pseudo (generalized) inverse $[A^T A]^{-1} A^T$ as long as A is not rank-deficient—that is, all of its columns are independent of each other (see item 2 in Remark 1.1). The left pseudo-inverse matrix can be computed by using the MATLAB command `pinv()`.

The LSE solution (2.1.10) can be obtained by using the `pinv()` command or the backslash (\) operator.

```
>>A = [1; 2]; b = [2.1; 3.9];
>>x = pinv(A)*b %A\b or x = (A'*A)^-1*A'*b, equivalently
  x = 1.9800
```

```
function  x = lin_eq(A,B)
%This function finds the solution to Ax = B
[M,N] = size(A);
if size(B,1) ~= M
  error('Incompatible dimension of A and B in lin_eq()!')
end
if M == N, x = A^-1*B; %x = inv(A)*B or gaussj(A,B); %Eq.(2.1.1)
 elseif M < N %Minimum-norm solution (2.1.7)
   x = pinv(A)*B; %A'*(A*A')^-1*B; or eye(size(A,2))/A*B
 else %LSE solution (2.1.10) for M > N
   x = pinv(A)*B; %(A'*A)^-1*A'*B or x = A\B
end
```

The above MATLAB routine `lin_eq()` is designed to solve a given set of equations, covering all of the three cases in Sections 2.1.1, 2.1.2, and 2.1.3.

(cf) The power of the `pinv()` command is beyond our imagination as you might have felt in Problem 1.14. Even in the case of $M < N$, it finds us a LS solution if the equations are inconsistent. Even in the case of $M > N$, it finds us a minimum-norm solution if the equations are redundant. Actually, the three cases can be dealt with by a single `pinv()` command in the above routine.

2.1.4 RLSE (Recursive Least-Squares Estimation)

In this section we will see the so-called RLSE (Recursive Least-Squares Estimation) algorithm, which is a recursive method to compute the LSE solution. Suppose we know the theoretical relationship between the temperature $t[^\circ]$ and the resistance $R[\Omega]$ of a resistor as

$$c_1 t + c_2 = R$$

and we have lots of experimental data $\{(t_1, R_1), (t_2, R_2), \dots, (t_k, R_k)\}$ collected up to time k. Since the above equation cannot be satisfied for all the data with any value of the parameters c_1 and c_2, we should try to get the parameter estimates that are optimal in some sense. This corresponds to the overdetermined case dealt with in the previous section and can be formulated as an LSE problem that we must solve a set of linear equations

$$A_k \mathbf{x}_k \approx \mathbf{b}_k, \quad \text{where } A_k = \begin{bmatrix} t_1 & 1 \\ t_2 & 1 \\ \cdot & \cdot \\ t_k & 1 \end{bmatrix}, \ \mathbf{x}_k = \begin{bmatrix} c_{1,k} \\ c_{2,k} \end{bmatrix}, \text{ and } \mathbf{b}_k = \begin{bmatrix} R_1 \\ R_2 \\ \cdot \\ R_k \end{bmatrix}$$

for which we can apply Eq. (2.1.10) to get the solution as

$$\mathbf{x}_k = [A_k^T A_k]^{-1} A_k^T \mathbf{b}_k \tag{2.1.11}$$

Now, we are given a new experimental data (t_{k+1}, R_{k+1}) and must find the new parameter estimate

$$\mathbf{x}_{k+1} = [A_{k+1}^T A_{k+1}]^{-1} A_{k+1}^T \mathbf{b}_{k+1} \tag{2.1.12}$$

with

$$A_{k+1} = \begin{bmatrix} t_1 & 1 \\ \cdot & \cdot \\ t_k & 1 \\ t_{k+1} & 1 \end{bmatrix}, \quad \mathbf{x}_{k+1} = \begin{bmatrix} c_{1,k+1} \\ c_{2,k+1} \end{bmatrix}, \quad \text{and} \quad \mathbf{b}_{k+1} = \begin{bmatrix} R_1 \\ \cdot \\ R_k \\ R_{k+1} \end{bmatrix}$$

How do we compute this? If we discard the previous estimate $\mathbf{x}_k$ and make direct use of Eq. (2.1.12) to compute the next estimate $\mathbf{x}_{k+1}$ every time a new data pair is available, the size of matrix A will get bigger and bigger as the data pile up, eventually defying any powerful computer in this world.

How about updating the previous estimate by just adding the correction term based on the new data to get the new estimate? This is the basic idea of the RLSE algorithm, which we are going to trace and try to understand. In order to do so, let us define the notations

$$A_{k+1} = \begin{bmatrix} A_k \\ \mathbf{a}_{k+1}^T \end{bmatrix}, \quad \mathbf{a}_{k+1} = \begin{bmatrix} t_{k+1} \\ 1 \end{bmatrix}, \quad \mathbf{b}_{k+1} = \begin{bmatrix} \mathbf{b}_k \\ R_{k+1} \end{bmatrix}, \quad \text{and} \quad P_k = [A_k^T A_k]^{-1} \tag{2.1.13}$$

and see how the inverse matrix P_k is to be updated on arrival of the new data (t_{k+1}, R_{k+1}).

$$P_{k+1} = [A_{k+1}^T A_{k+1}]^{-1} = \left[[\, A_k^T \quad \mathbf{a}_{k+1} \,] \begin{bmatrix} A_k \\ \mathbf{a}_{k+1}^T \end{bmatrix} \right]^{-1}$$

$$= [A_k^T A_k + \mathbf{a}_{k+1}\mathbf{a}_{k+1}^T]^{-1} = [P_k^{-1} + \mathbf{a}_{k+1}\mathbf{a}_{k+1}^T]^{-1} \tag{2.1.14}$$

(Matrix Inversion Lemma in Appendix B)

$$P_{k+1} = P_k - P_k\mathbf{a}_{k+1}[\mathbf{a}_{k+1}^T P_k \mathbf{a}_{k+1} + 1]^{-1}\mathbf{a}_{k+1}^T P_k \tag{2.1.15}$$

It is interesting that $[\mathbf{a}_{k+1}^T P_k \mathbf{a}_{k+1} + 1]$ is nothing but a scalar and so we do not need to compute the matrix inverse thanks to the Matrix Inversion Lemma (Appendix B). It is much better in the computational aspect to use the recursive formula (2.1.15) than to compute $[A_{k+1}^T A_{k+1}]^{-1}$ directly. We can also write Eq. (2.1.12) in a recursive form as

$$\mathbf{x}_{k+1} \overset{(2.1.12,\ 14)}{=} P_{k+1}A_{k+1}^T\mathbf{b}_{k+1} \overset{(2.1.13)}{=} P_{k+1}[A_k^T \ \mathbf{a}_{k+1}]\begin{bmatrix} \mathbf{b}_k \\ R_{k+1} \end{bmatrix}$$

$$= P_{k+1}[A_k^T\mathbf{b}_k + \mathbf{a}_{k+1}R_{k+1}] \overset{(2.1.11)}{=} P_{k+1}[A_k^T A_k \mathbf{x}_k + \mathbf{a}_{k+1}R_{k+1}]$$

$$\overset{(2.1.13)}{=} P_{k+1}[(A_{k+1}^T A_{k+1} - \mathbf{a}_{k+1}\mathbf{a}_{k+1}^T)\mathbf{x}_k + \mathbf{a}_{k+1}R_{k+1}]$$

$$\overset{(2.1.13)}{=} P_{k+1}[P_{k+1}^{-1}\mathbf{x}_k - \mathbf{a}_{k+1}\mathbf{a}_{k+1}^T\mathbf{x}_k + \mathbf{a}_{k+1}R_{k+1}]$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + P_{k+1}\mathbf{a}_{k+1}(R_{k+1} - \mathbf{a}_{k+1}^T\mathbf{x}_k) \tag{2.1.16}$$

We can use Eq. (2.1.15) to rewrite the gain matrix $P_{k+1}\mathbf{a}_{k+1}$ premultiplied by the 'error' to make the correction term on the right-hand side of Eq. (2.1.16) as

$$K_{k+1} = P_{k+1}\mathbf{a}_{k+1} \overset{(2.1.15)}{=} [P_k - P_k\mathbf{a}_{k+1}[\mathbf{a}_{k+1}^T P_k\mathbf{a}_{k+1} + 1]^{-1}\mathbf{a}_{k+1}^T P_k]\mathbf{a}_{k+1}$$

$$= P_k\mathbf{a}_{k+1}[I - [\mathbf{a}_{k+1}^T P_k\mathbf{a}_{k+1} + 1]^{-1}\mathbf{a}_{k+1}^T P_k\mathbf{a}_{k+1}]$$

$$= P_k\mathbf{a}_{k+1}[\mathbf{a}_{k+1}^T P_k\mathbf{a}_{k+1} + 1]^{-1}\{[\mathbf{a}_{k+1}^T P_k\mathbf{a}_{k+1} + 1] - \mathbf{a}_{k+1}^T P_k\mathbf{a}_{k+1}\}$$

$$K_{k+1} = P_k\mathbf{a}_{k+1}[\mathbf{a}_{k+1}^T P_k\mathbf{a}_{k+1} + 1]^{-1} \tag{2.1.17}$$

and substitute this back into Eq. (2.1.15) to write it as

$$P_{k+1} = P_k - K_{k+1}\mathbf{a}_{k+1}^T P_k \tag{2.1.18}$$

The following MATLAB routine "`rlse_online()`" implements this RLSE (Recursive Least-Squares Estimation) algorithm that updates the parameter estimates by using Eqs. (2.1.17), (2.1.16), and (2.1.18). The MATLAB program "`do_rlse.m`" updates the parameter estimates every time new data arrive and compares the results of the on-line processing with those obtained by the off-line (batch job) processing—that is, by using Eq.(2.1.12) directly. Noting that

- the matrix $[A_k^T A_k]$ as well as $\mathbf{b}_k$ consists of information and is a kind of squared matrix that is nonnegative, and
- $[A_k^T A_k]$ will get larger, or, equivalently, $P_k = [A_k^T A_k]^{-1}$ will get smaller and, consequently, the gain matrix K_k will get smaller as valuable information data accumulate,

one could understand that P_k is initialized to a very large identity matrix, since no information is available in the beginning. Since a large/small P_k makes the correction term on the right-hand side of Eq. (2.1.16) large/small, the RLSE algorithm becomes more conservative and reluctant to learn from the new data as the data pile up, while it is willing to make use of the new data for updating the estimates when it is hungry for information in the beginning.

```
function [x,K,P] = rlse_online(aT_k1,b_k1,x,P)
K = P*aT_k1'/(aT_k1*P*aT_k1'+1); %Eq.(2.1.17)
x = x +K*(b_k1-aT_k1*x); %Eq.(2.1.16)
P = P-K*aT_k1*P;  %Eq.(2.1.18)
```

```
%do_rlse
clear
xo = [2  1]'; %The true value of unknown coefficient vector
NA = length(xo);
x = zeros(NA,1); P = 100*eye(NA,NA);
for k = 1:100
   A(k,:) = [k*0.01 1];
   b(k,:) = A(k,:)*xo +0.2*rand;
   [x,K,P] = rlse_online(A(k,:),b(k,:),x,P);
 end
 x   % the final parameter estimate
 A\b % for comparison with the off-line processing (batch job)
```

2.2 SOLVING A SYSTEM OF LINEAR EQUATIONS

2.2.16 Gauss Elimination

For simplicity, we assume that the coefficient matrix A in Eq. (2.0.1) is a nonsingular 3×3 matrix with $M = N = 3$. Then we can write the equation as

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \tag{2.2.0a}$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \tag{2.2.0b}$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \tag{2.2.0c}$$