Systems of Bilinear Equations [1]

Scott Cohen Carlo Tomasi

{scohen, tomasi}@cs.stanford.edu
Computer Science Department
Stanford University
Stanford, CA 94305

[1]This research was sponsored in part by the National Science Foundation under contract IRI-9509149.

Abstract

How hard is it to solve a system of bilinear equations? No solutions are presented in this report, but the problem is posed and some preliminary remarks are made. In particular, solving a system of bilinear equations is reduced by a suitable transformation of its columns to solving a homogeneous system of bilinear equations. In turn, the latter has a nontrivial solution if and only if there exist two invertible matrices that, when applied to the tensor of the coefficients of the system, zero its first column. MATLAB code is given to manipulate three-dimensional tensors, including a procedure that finds one solution to a bilinear system often, but not always.

Contents

1 Introduction

One of the most common types of systems of equations that arise in science and engineering is a linear system

$$Ax = b \tag{1}$$

where $A \in \mathbf{R}^{m\times n}$, $x \in \mathbf{R}^n$, and $b \in \mathbf{R}^m$. The system (1) consists of m equations

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \qquad \text{for } \ i = 1, \ldots, m \tag{2}$$

in n unknowns $x_1, \ldots, x_n$. The left-hand side of the ith equation in (2) is the linear form $a_i^T x$, where a_i^T is the ith row of A.

Adding one dimension to the linear system (2) yields a *bilinear* system of equations

$$\sum_{j=1}^{m}\sum_{k=1}^{n} t_{ijk}y_j z_k = d_i \qquad \text{for } \ i = 1, \ldots, l \tag{3}$$

where the t_{ijk} and the d_i are known real numbers. This system consists of l equations in $m + n$ unknowns $y_1, \ldots, y_m$ and $z_1, \ldots, z_n$. Here we have two unknown vectors

$$\begin{aligned} y &= [\ y_1 \ \cdots \ y_m \ ]^T \in \mathbf{R}^m \quad \text{ and} \\ z &= [\ z_1 \ \cdots \ z_n \ ]^T \in \mathbf{R}^n \ , \end{aligned}$$

an order-3 tensor of coefficients

$$T = [\ t_{ijk}\ ] \in \mathbf{R}^{l\times m\times n} \ ,$$

and a right-hand side vector

$$d = [\ d_1 \ \cdots \ d_l \ ]^T \in \mathbf{R}^l \ .$$

A shorthand notation for the bilinear system (3) is

$$yTz = d.$$

A homogeneous bilinear system is a bilinear system with right-hand side $d = 0$:

$$\sum_{j=1}^{m}\sum_{k=1}^{n} t_{ijk}y_j z_k = 0 \qquad \text{for } \ i = 1, \ldots, l \ , \tag{4}$$

or, in our shorthand notation, $yTz = 0$.

The system (3) can be written in three alternative forms that make use of matrix notation:

$$\begin{aligned} y^T A_i z &= d_i \qquad \text{for } \ i = 1, \ldots, l & (5) \\ B(y)z &= d & (6) \\ C(z)y &= d \ , & (7) \end{aligned}$$

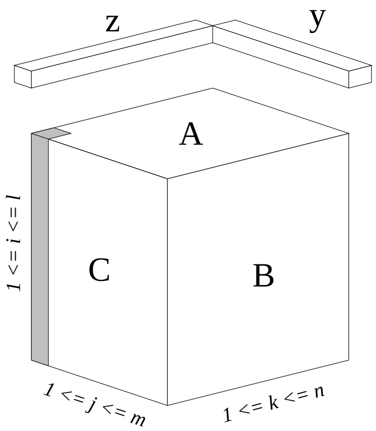

Figure 1: The tensor of the coefficients of a system of bilinear equations. The two vectors y and z are placed along the dimensions that they contract. The shaded area is the first column of the tensor (j=k=1).

where

$$B(y) = \sum_{j=1}^{m} y_j B_j \tag{8}$$

$$C(z) = \sum_{k=1}^{n} z_k C_k \ , \tag{9}$$

and where the matrices

$$A_i = \begin{bmatrix} t_{i11} & \cdots & t_{i1n} \\ \vdots & \ddots & \vdots \\ t_{im1} & \cdots & t_{imn} \end{bmatrix} \in \mathbf{R}^{m \times n} \qquad \text{for } \ i = 1, \ldots, l \tag{10}$$

$$B_j = \begin{bmatrix} t_{1j1} & \cdots & t_{1jn} \\ \vdots & \ddots & \vdots \\ t_{lj1} & \cdots & t_{ljn} \end{bmatrix} \in \mathbf{R}^{l \times n} \qquad \text{for } \ j = 1, \ldots, m \tag{11}$$

$$C_k = \begin{bmatrix} t_{11k} & \cdots & t_{1mk} \\ \vdots & \ddots & \vdots \\ t_{l1k} & \cdots & t_{lmk} \end{bmatrix} \in \mathbf{R}^{l \times m} \qquad \text{for } \ k = 1, \ldots, n \tag{12}$$

represent "slices" of the tensor T along its three dimensions (see figure 1). The representation (5) makes it clear that the left-hand side of the ith bilinear equation is the bilinear form $y^T A_i z$.[1] The

[1] A more general system of equations includes strictly linear terms in y and z in the left-hand side of each equation. If the ith equation were $y^T A_i z + e_i^T y + f_i^T z = d_i$, then we would have a system of *biaffine* equations. As in the bilinear case, holding one of the unknown vectors fixed results in a linear system to solve for the other vector. This is clear if we write the system as $B(y)z + Ey + Fz = d$, where e_i^T and f_i^T are the ith rows of the matrices E and F, respectively. At the present time, we only consider systems in which the left-hand side of each equation is a bilinear function of y and z.

representation (6) makes it clear that fixing the vector y results in a linear system to solve for z. Similarly, the representation (7) shows that fixing the vector z yields a linear system to solve for y. Therefore, fixing one of the unknown vectors leaves a linear system to solve for the other vector.

A very simple example of a bilinear system is the single equation of a rectangular hyperbola in the plane:

$$yz = d, \qquad y \in \mathbf{R}, \;\; z \in \mathbf{R}, \;\; d \in \mathbf{R}. \tag{13}$$

Our definition of a bilinear system also includes the matrix form of equation (13):

$$YZ = D, \qquad Y \in \mathbf{R}^{m\times p}, \;\; Z \in \mathbf{R}^{p\times n}, \;\; D \in \mathbf{R}^{m\times n}. \tag{14}$$

In (14) there are mn equations, one for each entry d_{ij} of D, and $(m+n)p$ unknowns, one for each entry in Y and Z. Let $y \in \mathbf{R}^{mp}$ and $z \in \mathbf{R}^{pn}$ be the vectors formed by the row major and column major order of the elements in the matrices Y and Z, respectively. Also, let $A_{ij} \in \mathbf{R}^{mp\times pn}$ be the matrix which has the identity matrix I_p in its (i,j)th $p \times p$ block, and zeros elsewhere. Then (14) is equivalent to the bilinear system

$$y^T A_{ij} z = d_{ij}, \qquad \text{for} \;\; i = 1, \ldots, m, \;\; j = 1, \ldots, n. \tag{15}$$

The matrix A_{ij} "picks out" the elements in Y and Z that are multiplied together in forming the matrix product element d_{ij}. Obviously, we can unroll the 2D index into a 1D index to make (15) look exactly like the form (5).

We would like to understand bilinear systems as well as we understand linear systems. The following problems arise:

1. Give conditions on the coefficient tensor T and right-hand side vector d under which the system (3) admits a solution. In the homogeneous case $d = 0$, give conditions on T under which the homogeneous bilinear system admits a nontrivial solution (y, z), $y \neq 0$, $z \neq 0$.

2. Describe the set of solutions to a bilinear system.

3. Compute the set of solutions to a bilinear system.

4. If the equations in a bilinear system are incompatible, determine a least squares solution; that is, find a pair (y, z) that minimizes the squared error

$$e^2(y,z) = \sum_{i=1}^{l} \left(\sum_{j=1}^{m} \sum_{k=1}^{n} t_{ijk} y_j z_k - d_i \right)^2 .$$

All these problems have been completely solved for linear systems.

This report is organized as follows. In section 2, we study bilinear systems in which one of the unknown vectors y, z is a scalar ($m = 1$ or $n = 1$). Such bilinear systems are thinly veiled linear systems. In section 3, we characterize the solutions to general homogeneous and non-homogeneous systems. The zeros of a certain degree $2n$ polynomial in m variables $y_1, \ldots, y_m$ are exactly the vectors y for which there is a solution (y, z) to the homogeneous bilinear system (4); the zeros of a certain degree $2m$ polynomial in n variables $z_1, \ldots, z_n$ are exactly the vectors z for which there is a solution (y, z) to (4). In section 4, we present some simple results on the existence of solutions (nontrivial solutions in the case of homogeneous systems). In section 5, we give a technique for reducing a non-homogeneous problem to a homogeneous one.

Given that non-homogeneous bilinear systems are no harder than homogenous ones, we address the solution of homogenous systems in section 6. First we show that finding a nontrivial solution to a homogeneous bilinear system is equivalent to finding invertible matrices which zero the coefficient tensor's first column ($j = k = 1$ — see figure 1) when applied to its j-rows and k-rows. We then give a heuristic method for zeroing the first column of a homogeneous system with Householder matrices. The method has always succeeded in several random trials when $l >> m, n$, but fails often when l is not much larger than m and n. In section 7, we show that homogeneous bilinear systems generalize the eigenvalue problem to more than two, possibly non-square matrices. With this fact in mind, we review two matrix eigenvalue problems in section 8, including the case when the matrices are not square.

In practice, the equations in system (3) usually come from noisy data, and are therefore inconsistent. In section 9, we discuss the problem of finding a least squares solution to an inconsistent system. In section 10, we pose two problems which are related to, but harder than the problem of solving bilinear systems. Our concluding remarks in section 11 are followed by two appendices. Appendix A lists some MATLAB code to work with tensors of order-3 (that is, three-dimensional tensors). This code includes a program that in many cases (but not always) zeros the first column of a tensor by Householder reflections. Appendix B contains MATLAB code that finds a basis of solution functions $x_1(\lambda), \ldots, x_p(\lambda)$ for the eigenvalue problem $(A - \lambda B)x = 0$ when for every $\lambda \in \mathbf{C}$ $A - \lambda B$ has linearly dependent columns.

2 Bilinear Systems with $m = 1$

We begin our study of bilinear systems with the simple case $m = 1$ or $n = 1$. We shall see in the next few sections that solving such bilinear systems reduces to solving linear systems. Without loss of generality, only the $m = 1$ case is considered.

2.1 Homogeneous Systems with $m = n = 1$

When $m = n = 1$, the two unknown vectors y and z, as well as the matrices $A_i = a_i$, are actually scalars. The homogenous bilinear system is

$$a_i y z = 0 \qquad \text{for } \ i = 1, \ldots, l. \tag{16}$$

It is trivial to see that the solution set is

$$S = \{\ (y, z)\ :\ y = 0 \ \text{ or } \ z = 0\ \} \qquad \text{if } \ \exists i\ a_i \neq 0$$

or

$$S = \{\ (y, z)\ :\ y \in \mathbf{R},\ z \in \mathbf{R}\ \} \qquad \text{if } \ \forall i\ a_i = 0.$$

In the former case, the solution set is the union of the y-axis and the z-axis. In the latter case, the solution set is the entire yz-plane. See figure 2. If we let $a = [\ a_1 \ \cdots \ a_l\ ]^T \in R^l$, then the solution set is one-dimensional if $\text{rank}(a) = 1$ and two-dimensional if $\text{rank}(a) = 0$.

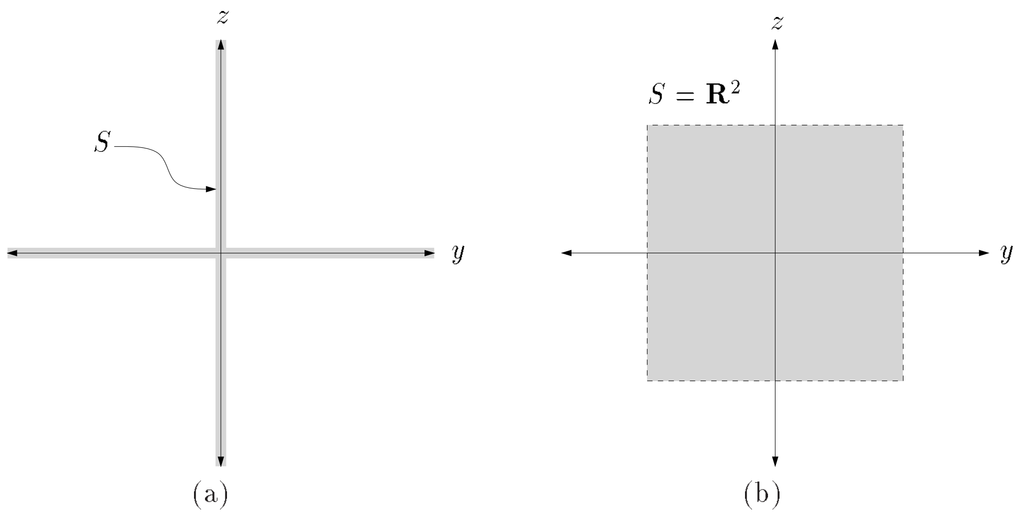

Figure 2: Solution Set for a Homogeneous Bilinear System with $m = n = 1$. (a) The solution set S to (16) is the union of the y-axis and z-axis when $\exists i\ a_i \neq 0$. (b) The solution set S is the entire yz-plane when $\forall i\ a_i = 0$.

2.2 Homogeneous Systems with $m = 1$

The case $m = 1$ with general n is almost as easy as the case $m = n = 1$ considered in the previous section. When $m = 1$, the vector y is a scalar, while the matrices $A_i = a_i^T \in \mathbf{R}^{1\times n}$ are row vectors. The homogenous bilinear system is

$$y a_i^T z = 0 \qquad \text{for} \;\; i = 1, \ldots, l. \tag{17}$$

When $y = 0$, any vector $z \in \mathbf{R}^n$ satisfies (17). When $y \neq 0$, the vector z must be in the nullspace of the matrix $A \in \mathbf{R}^{l\times n}$ whose rows are the a_i^T. Hence the solution set is

$$S = \{\ (0, z)\ :\ z \in \mathbf{R}^n\ \} \ \bigcup\ \{\ (y, z)\ :\ y \in \mathbf{R},\ z \in \mathrm{null}(A)\ \},$$

where

$$A = [\ a_1\ \cdots\ a_l\ ]^T.$$

If we let

$$\begin{aligned} P &= \{\ (0, z)\ :\ z \in \mathbf{R}^n\ \} \equiv \{0\} \times \mathbf{R}^n \qquad \text{and} \\ Q &= \{\ (y, z)\ :\ y \in \mathbf{R},\ z \in \mathrm{null}(A)\ \} \equiv \mathbf{R} \times \mathrm{null}(A), \end{aligned}$$

then

$$S = P \bigcup Q.$$

Now consider the case $m = 1$, $n = 2$. Then P is the plane through the origin in (y, z_1, z_2)-space which is perpendicular to the y-axis, and $\dim(\mathrm{null}(A)) \in \{0, 1, 2\}$. If $\dim(\mathrm{null}(A)) = 0$, then $Q = \mathbf{R} \times \{(0, 0)\}$ is the y-axis, and the solution set $S = P \bigcup Q$ is the union of a plane and a line.

If $\dim(\text{null}(A)) = 1$ with $\text{null}(A) = \text{span}(v)$, then Q is a plane through the origin spanned by the vectors $(0, v)$ and $(1, 0, 0)$, and $S = P \bigcup Q$ is the union of two planes. If $\dim(\text{null}(A)) = 2$, then $Q = \mathbf{R} \times \mathbf{R}^2$, and the solution set is $S = Q = \mathbf{R} \times \mathbf{R}^2$, the whole (y, z_1, z_2)-space. The three qualitatively different solution sets S are shown in figure 3.

2.3 Non-Homogeneous Systems with $m = 1$

In the previous section we saw that solving a homogeneous bilinear system with $m = 1$ reduces to solving a homogeneous linear system $Az = 0$. We now show that solving a non-homogeneous bilinear system with $m = 1$ reduces to solving a non-homogeneous linear system.

Consider the bilinear system

$$y a_i^T z = d_i \qquad \text{for } \; i = 1, \ldots, l. \tag{18}$$

If $y = 1$, then (18) becomes the linear system

$$Az = d. \tag{19}$$

If z is a solution to (19), then obviously $(1, z)$ is a solution to (18) and, more generally, $(c, z/c)$ is a solution for any $c \neq 0$. The solution set to (18) is

$$S = \{ \, (c, z/c) \; : \; Az = d, \; c \neq 0 \, \} \qquad \text{if } \; d \neq 0.$$

Of course, the solution set S will be empty if there is no solution to the linear system (19).

If S is empty, we can still ask for a least squares solution to (18). By this we mean a pair (y, z) that minimizes the squared error

$$e^2(y, z) = ||yAz - d||_2^2.$$

Note that

$$e^2(c, z/c) = e^2(1, z) = ||Az - d||_2^2 \qquad \forall c \neq 0$$

and

$$\min_z ||Az - d||_2^2 \leq ||d||_2^2 = e^2(0, z).$$

Thus a least squares solution z to the linear system (19) yields least squares solutions $(c, z/c)$ $\forall c \neq 0$ to the bilinear system (18).

3 Characterizing the Solutions

If (y, z) is a solution to the general bilinear system (3), then so is $(cy, z/c)$ for any nonzero constant c. If $d \neq 0$, then $(y, 0)$ and $(0, z)$ are not solutions for any y or z. Consequently, solutions to a non-homogeneous system can be characterized by, say, $||z|| = 1$ without loss of generality. If (y, z) is a solution to the homogeneous system (4), then so is $(c_1 y, c_2 z)$ for any constants c_1, c_2. Therefore, the nontrivial solutions to a homogeneous system can be characterized by $||y|| = ||z|| = 1$ without loss of generality.

Let us focus for the moment on the homogeneous system

$$B(y)z = 0.$$

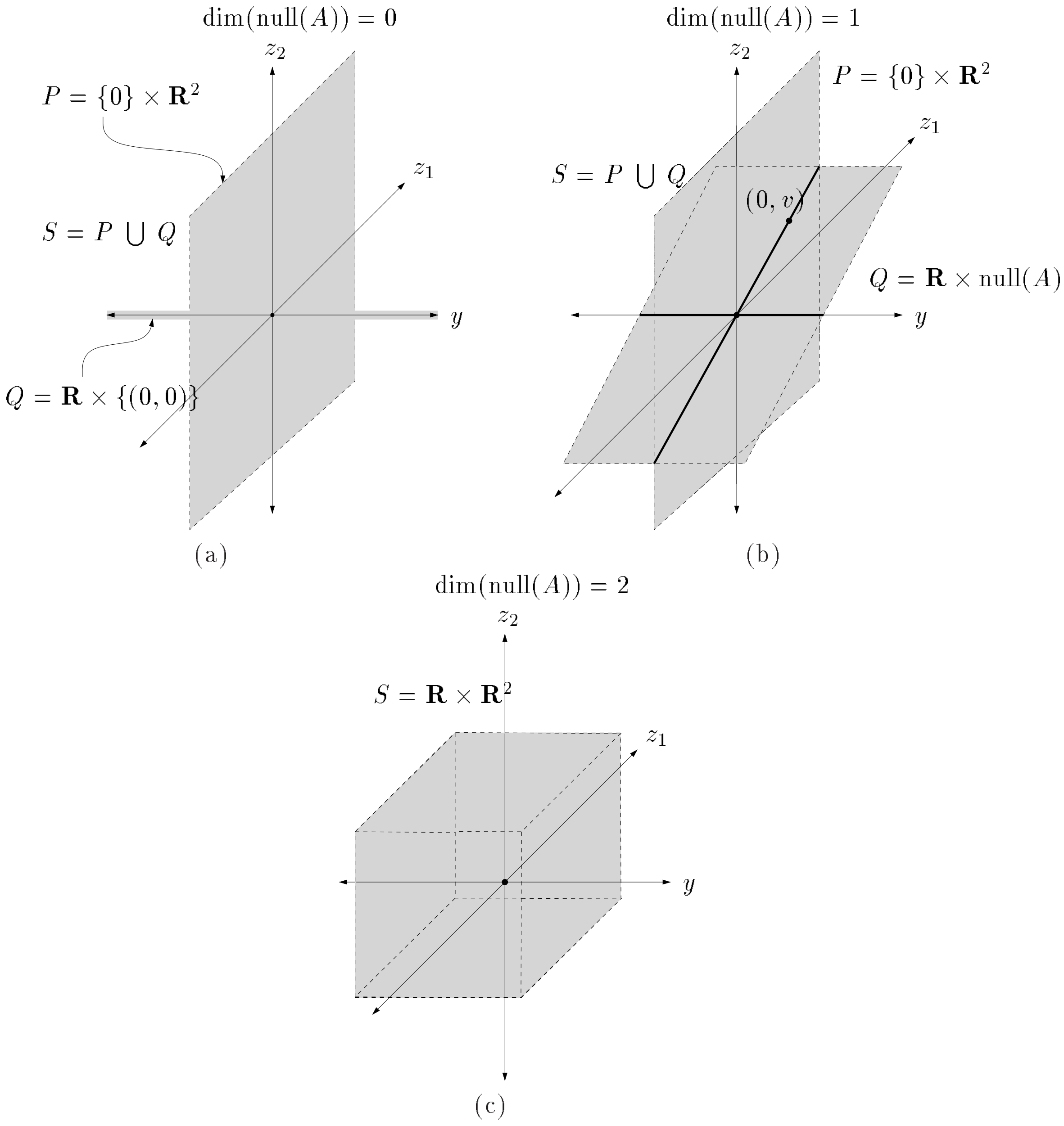

Figure 3: Solution Set for a Homogeneous Bilinear System with $m = 1$, $n = 2$. (a) The solution set S to (17) is the union of the plane $P = \{0\} \times \mathbf{R}^2$ and the y-axis when $\dim(\mathrm{null}(A)) = 0$. (b) The solution set S is the union of the two planes P and $Q = \mathbf{R} \times \mathrm{null}(A)$ when $\dim(\mathrm{null}(A)) = 1$. (c) The solution set S is the entire (y, z_1, z_2)-space when $\dim(\mathrm{null}(A)) = 2$.

Recall that $B(y) \in \mathbf{R}^{l \times n}$. It follows that nontrivial solutions to the homogeneous system exist iff there exists $y \neq 0$ such that

$$\operatorname{rank}(B(y)) < n \tag{20}$$

or, equivalently,

$$\det(B^T(y)B(y)) = 0. \tag{21}$$

For each such y that satisfies (21), any

$$z \in \operatorname{null}(B(y))$$

completes a solution (y, z).

Similar observations can be made with the role of B and C, and y and z, reversed because $B(y)z \equiv C(z)y$. Here we write the homogeneous bilinear system as

$$C(z)y = 0.$$

Recall that $C(z) \in \mathbf{R}^{l \times m}$. It follows that nontrivial solutions to the homogeneous system exist iff there exists $z \neq 0$ such that

$$\operatorname{rank}(C(z)) < m \tag{22}$$

or, equivalently,

$$\det(C^T(z)C(z)) = 0. \tag{23}$$

For each such z that satisfies (23), any

$$y \in \operatorname{null}(C(z))$$

completes a solution (y, z).

In section 5 we show that a non-homogeneous system can be reduced to a homogeneous system. Thus equation (21) seems to imply that solving the original system (3) is as hard as solving a polynomial of degree $2n$ in the $m-1$ variables $y_2, \ldots, y_m$ (if $y_1 = 1$ for normalization). However, this polynomial has a strong structure because of equation (8), so a solution may be easier to find.

4 Some Simple Existence Results

A homogeous linear system has a nontrivial solution whenever the number of equations is less than the number of unknowns. The bilinear analog to this result is given below.

Theorem 1 *Any homogeneous bilinear system defined by a coefficient tensor $T \in \mathbf{R}^{l \times m \times n}$ with $l < m$ or $l < n$ admits a nontrivial solution.*

Proof 1 *If $l < m$, then (22) holds for every z because $C(z)$ has fewer rows than columns. Similarly, $l < n$ implies that (20) holds for every y.*

If $l < n$, the solution set is

$$S_y = \{\ (y, z)\ :\ y \in \mathbf{R}^m,\ z \in \operatorname{null}(B(y))\ \}\ ;$$

if $l < m$, the solution set is

$$S_z = \{\ (y,z)\ :\ y \in \mathrm{null}(C(z)),\ z \in \mathbf{R}^n\ \}\ .$$

Note that the number of equations is compared separately to the number of unknowns in the vectors y and z, not to the total number of unknowns $m+n$.

Another result for linear systems that has an analog for bilinear systems is as follows: Suppose that the *square* homogeneous linear system $Ax = 0$ does *not* have a nontrivial solution. Then $Ax = b$ admits a solution for every right-hand side vector b.

Theorem 2 *Suppose $T \in \mathbf{R}^{(l=m)\times m\times n}$ or $T \in \mathbf{R}^{(l=n)\times m\times n}$, and that the homogeneous bilinear system $yTz = 0$ has no nontrivial solution. Then $yTz = d$ admits a solution for every non-zero right-hand side vector d.*

Proof 2 *Suppose $T \in \mathbf{R}^{(l=m)\times m\times n}$. Then $C(z) \in R^{m\times m}$ is a square matrix for every $z \in \mathbf{R}^n$. We are given that $C(z)y = 0$ has no nontrivial solution. This implies that $\det C(z) \neq 0$ for every z. Consequently, $C(z)y = d$ admits a solution for every z. The case when $T \in \mathbf{R}^{(l=n)\times m\times n}$ follows from a similar argument.*

Finally, we show that a single, nontrivial bilinear equation (i.e. a bilinear system with $l = 1$ and $A_1 \neq 0$) always admits a solution.

Theorem 3 *If $A \neq 0$, then the single bilinear equation $y^T Az = d$ admits a solution.*

Proof 3 *Let $A = U\Sigma V^T$ be an SVD of A, $\hat{y} = U^T y$, and $\hat{z} = V^T z$. Then*

$$\begin{aligned} y^T Az = d \quad &\Longleftrightarrow \quad \hat{y}^T \Sigma \hat{z} = d \\ &\Longleftrightarrow \quad \sum_{j=1}^{\mathrm{rank}(A)} \sigma_j \hat{y}_j \hat{z}_j = d. \end{aligned}$$

Since $A \neq 0$, we have $\mathrm{rank}(A) \geq 1$. Set $\hat{y}_1 = 1$, $\hat{z}_1 = d/\sigma_1$, $\hat{y}_j = \hat{z}_j = 0$ for $j = 2,\ldots,\mathrm{rank}(A)$. For arbitrary values of the remaining elements in $\hat{y}$ and $\hat{z}$, the pair $(y,z) = (U\hat{y}, V\hat{z})$ is a solution to $y^T Az = d$.

In fact, one can pick z or y arbitrarily, except that $z \notin \mathrm{null}(A)$ and $y \notin \mathrm{null}(A^T)$, and solve $y^T Az = d$ for the other vector. For instance, if $w = Az \neq 0$, any y on the plane $y^T w = d$ will do (e.g., $y = dw/||w||^2$). Similar reasoning applies to $v = A^T y \neq 0$.

5 Reduction to a Homogeneous System

In this section, we present a strategy for reducing a non-homogeneous bilinear system to a homogeneous one. This technique can be applied to linear systems as well, although it is somewhat nonstandard. The right-hand side vector $d \neq 0$ can be transformed by a linear transformation, for example by a Householder reflection ([3]), to a vector of the form

$$\hat{d} = [\ \delta\ 0\ \cdots\ 0\ ]^T\ ,$$

where $\delta \neq 0$, and the same transformation can be applied to the columns of the tensor T. If $Rd = \hat{d}$, then multiplying both sides of the bilinear system $B(y)z = d$ by R on the left gives

$$\begin{aligned} RB(y)z &= \hat{d} \\ R(y_1 B_1 + \cdots + y_m B_m)z &= \hat{d} \\ (y_1(RB_1) + \cdots + y_m(RB_m))z &= \hat{d} \\ (y_1 \hat{B}_1 + \cdots + y_m \hat{B}_m)z &= \hat{d} \\ \hat{B}(y)z &= \hat{d}. \end{aligned} \tag{24}$$

Equation (24) represents the bilinear system with coefficient tensor $\hat{T}$, where the $\hat{t}_{ijk}$ is equal to the (i,k)th element of $\hat{B}_j$. If $\hat{A}_i$ is defined so that its (j,k)th element is $\hat{t}_{ijk}$, then the equations in the transformed bilinear system (24) are

$$y^T \hat{A}_i z = \begin{cases} \delta & \text{if } i = 0 \\ 0 & \text{otherwise} \end{cases} . \tag{25}$$

If R is invertible, then (y,z) is a solution to the original bilinear system (5) iff (y,z) is a solution to the transformed bilinear system (25). The equations of (25) for $i = 2, \ldots, l$ form a homogeneous system. If (y,z) is a unit-norm solution of the latter, the pair $(\beta y, z)$, where β is an unknown parameter, can be replaced into the first equation ($i = 1$) to determine β:

$$\beta y^T \hat{A}_1 z = \delta$$

If $y^T \hat{A}_1 z = 0$, this solution is inconsistent with the original system. Otherwise,

$$\beta = \frac{\delta}{y^T \hat{A}_1 z} .$$

If this procedure is repeated for all the solutions to the homogeneous system and the inconsistent solutions are discarded, all consistent solutions to the full system remain.

6 Solving Homogeneous Bilinear Systems

6.1 An Equivalent Problem

Consider now a homogeneous bilinear system, and let (y,z) be a nontrivial solution (i.e. $y, z \neq 0$). Then, there exist invertible matrices (e.g., Householder reflections) R_y, R_z such that

$$R_y y = \delta_y e_1^{(m)} \quad \text{and} \quad R_z z = \delta_z e_1^{(n)},$$

where $\delta_y, \delta_z \neq 0$, and $e_1^{(p)}$ is the p-dimensional elementary vector with a 1 in its first component and zeros everywhere else. But then equation (5) can be rewritten as

$$y^T R_y^T \left(R_y^T\right)^{-1} A_i \,(R_z)^{-1} R_z z = 0,$$

or, equivalently,

$$\left(e_1^{(m)}\right)^T \tilde{A}_i \, e_1^{(n)} = 0,$$

where

$$\tilde{A}_i = \left(R_y^T\right)^{-1} A_i \, (R_z)^{-1} \, .$$

But

$$\left(e_1^{(m)}\right)^T \tilde{A}_i \, e_1^{(n)} = \tilde{a}_{11}^{(i)} \, ,$$

the first entry of $\tilde{A}_i$, so that we have

$$\tilde{a}_{11}^{(i)} = 0 \qquad \text{for} \;\; i = 1, \ldots, l \, .$$

In conclusion, for every nontrivial solution to a homogeneous system there exist two invertible matrices that, when applied to the j-rows and the k-rows of the tensor of the homogeneous system, zero the tensor's first column.

The converse of the last statement is also true. Suppose R_y and R_z^T are invertible matrices that zero the coefficient tensor's first column when applied to the j-rows and the k-rows of the tensor. Then

$$\left(e_1^{(m)}\right)^T R_y A_i R_z^T \, e_1^{(n)} = 0 \qquad \text{for} \;\; i = 1, \ldots, l \, ,$$

and

$$(y, z) = \left(R_y^T e_1^{(m)}, R_z^T e_1^{(n)}\right)$$

is obviously a solution to the homogeneous bilinear system. Since R_y^T and R_z^T are invertible, $y = R_y^T e_1^{(m)}$ and $z = R_z^T e_1^{(n)}$ are nonzero. Therefore, finding a nontrivial solution to a homogeneous bilinear system is equivalent to finding invertible matrices which zero the coefficient tensor's first column when applied to its j-rows and k-rows.[2] Because the argument above holds also when R_y and R_z are orthogonal (e.g., Householder reflections), finding a unit norm solution to a homogeneous bilinear system is also equivalent to finding orthogonal matrices which zero the coefficient tensor's first column when applied to its j-rows and k-rows.

6.2 Zeroing the First Column of a Homogeneous System

We could not find a method for zeroing the first column of a tensor by orthogonal matrices applied to its rows. Here is the rationale behind a simple attempt, which does often work whenever $l >> m, n$.

Consider for instance the matrices B_j in equation (8). If B_j is rank deficient, we have a solution to the homogeneous system, given by $y = e_j^{(m)}$ and a solution to the linear system $B_j z = 0$. Assume now that B_j is full rank for all j. By the minimax characterization of singular values, the smallest vector that can be formed by a unit-norm linear combination of the columns of B_j is $\sigma_n^{(j)} u_n^{(j)}$, that is, the smallest singular value of B_j times the corresponding left singular vector. Let now

$$\sigma_n = \min_j \sigma_n^{(j)} \, ,$$

achieved for $j = j_{\min}$, and let u_n and v_n be the corresponding left and right singular vectors. The vector $\sigma_n u_n$ is the smallest among the unit-norm combinations of columns all from B_1, or all from

[2] An analogous result holds for linear systems. A homogeneous linear system admits a nonzero solution iff the first column of its coefficient matrix can be zeroed by an invertible linear transformation of its columns.

B_2, ..., or all from B_m. This vector can be brought to the first column of the tensor by two Householder reflections. In fact, let

$$h = -\text{house}(v_n) ,$$

where house(v_n) is the Householder vector for v_n (the first component of house(v_n) is normalized to be 1). Since the Householder matrix

$$P(h) = I - 2\frac{hh^T}{h^T h}$$

maps the elementary vector $e_1^{(n)}$ into v_n, the first column of $P(h)$ is equal to

$$P(h)e_1^{(n)} = v_n ,$$

and the first column of $B_{j_{\min}}P(h)$ is $\sigma_n u_n$. Because the first column of $B_{j_{\min}}$ is also a column of C_1 (see equations (7) and (9)), this reflection has the effect of moving the "small" vector $\sigma_n u_n$ to C_1. The same procedure, repeated with the roles of the B and C matrices reversed, will then bring this "small" vector, or an even smaller one, to the first column of the tensor ($j = k = 1$). The second reflection changes the column spaces of the matrices B_j, so there is a chance that another, smaller vector can be produced by repeating this two-step procedure. This, of course, need not be the case in general, and in fact for systems where l is not much larger than m and n failure to converge occurs often. If $l >> m, n$, on the other hand, this procedure has always succeeded in several random trials. Perhaps this observation can be of heuristic value in the search for a solution method.

Of course, zeroing the first column of the tensor is not the whole story, since this yields only one solution to the original system.

7 Homogeneous Bilinear Systems and Eigenvalue Problems

For the special case $n = 2$, the homogeneous system $C(z)y = 0$ becomes

$$(z_1 C_1 + z_2 C_2)y = 0, \tag{26}$$

where $C_1, C_2 \in \mathbf{R}^{l\times m}$. If we normalize z so that $z_1 = 1$, and write $z = [\, 1 \ -\lambda \,]^T$, $A = C_1$, and $B = C_2$, then (26) becomes

$$(A - \lambda B)y = 0. \tag{27}$$

If $l = m$, then (27) is the *generalized eigenvalue problem.* The normalization $z_1 = 1$ results in only a slight loss of generality because any solution with $z_1 \neq 0$ can be divided by z_1 to produce a solution with $z_1 = 1$. Setting $z_1 = 0$ produces an $l \times m \times 1$ homogeneous bilinear system

$$z_2 C_2 y = 0.$$

In section 2.2, we showed that homogeneous bilinear systems in which one of the unknown vectors is a scalar are essentially equivalent to linear systems. Thus the difficultly in solving (26) lies in solving the eigenvalue problem (27).[3] Here "solving the eigenvalue problem" means finding *both*

[3]Solving the homogeneous bilinear system (26) entails considering solutions to (27) corresponding to *real* eigenvalues.

the eigenvalues and corresponding eigenvectors. The eigenvalues define z, while the corresponding eigenvectors give y. When A and B are square, finding the eigenvalues is a well-studied problem. Given a finite set of eigenvalues, the corresponding set of eigenvectors follow easily. Difficulties arise in finding the set of eigenvectors when every $\lambda \in \mathbf{C}$ is an eigenvalue.

Removing the $l = m$ restriction results in non-square eigenvalue problems. Considering the case $n > 2$ shows that a homogeneous bilinear system generalizes the eigenvalue problem to *more than two matrices*

$$(C_1 + z_2 C_2 + \cdots + z_n C_n) y = 0.$$

The generalized eigenvalue problem (with two, possibly non-square matrices) is the topic of section 8.

8 The Generalized Eigenvalue Problem $Ax = \lambda Bx$

The *generalized eigenvalue problem* is to solve the equation

$$Ax = \lambda Bx \tag{28}$$

for $\lambda \in \mathbf{C}$ and $x \neq 0$, where $A \in \mathbf{R}^{n \times n}$ and $B \in \mathbf{R}^{n \times n}$ are square matrices. We can rewrite (28) as

$$(A - \lambda B)x = 0 \ . \tag{29}$$

There exists an $x \neq 0$ that satisfies (29) iff $\operatorname{rank}(A - \lambda B) < n$. For square matrices A and B, this condition is equivalent to the condition

$$\det(A - \lambda B) = 0. \tag{30}$$

The eigenvalues of the pair (A, B) are the values of $\lambda \in \mathbf{C}$ that satisfy (30). The set of eigenvalues of (A, B) is denoted by $\lambda(A, B)$.

Define the characteristic polynomial $p(\lambda) = \det(A - \lambda B)$. Note that p is a polynomial of degree at most n. If $p(\lambda) \equiv \delta \neq 0$, then $\lambda(A, B) = \emptyset$. If $p(\lambda) \equiv 0$, then $\lambda(A, B) = \mathbf{C}$. The remaining possibility is that $1 \leq \deg(p) \leq n$. In this case, the pair (A, B) has exactly $\deg(p)$ eigenvalues $\lambda \in \mathbf{C}$ (counting multiplicities).

If $B = I_n$, the $n \times n$ identity matrix, then (28) reduces to the *ordinary eigenvalue problem* $Ax = \lambda x$. In fact, the generalized eigenvalue problem reduces to the ordinary eigenvalue problem $B^{-1}Ax = \lambda x$ if B is invertible. The general case in which B is not necessarily invertible is reviewed in section 8.1.

Note that equation (28) still makes sense when $A \in \mathbf{R}^{m \times n}$ and $B \in \mathbf{R}^{m \times n}$ are not square. In this case, the generalized eigenvalue problem asks for nonzero vectors x which are mapped by A and B into parallel vectors Ax and Bx. We can modify our definition of $\lambda(A, B)$ to include the non-square case:

$$\lambda(A, B) \equiv \{ \lambda \in \mathbf{C} \ : \ \operatorname{rank}(A - \lambda B) < n \}.$$

We will show in section 8.2 that a non-square problem can be reduced to a square problem.

The set of eigenvectors $x(A, B)$ which satisfy (29) is

$$x(A, B) = \bigcup_{\lambda \in \lambda(A, B)} \operatorname{null}(A - \lambda B).$$

For a given λ, an SVD of $A - \lambda B$ provides a basis for null$(A - \lambda B)$. If $\lambda(A, B)$ is a finite set, we "compute" $x(A, B)$ by computing a basis for null$(A - \lambda_i B)$ for each $\lambda_i \in \lambda(A, B)$. Computing $x(A, B)$ when $\lambda(A, B) = \mathbf{C}$ is the subject of section 8.3. This is *not* an uncommon case. In fact, $\lambda(A, B) = \mathbf{C}$ whenever $m < n$, since then rank$(A - \lambda B) < n$.

8.1 The Square Generalized Eigenvalue Problem

Let us assume that $A, B \in \mathbf{R}^{n \times n}$. The matrix decomposition which solves the generalized eigenvalue problem is the *generalized Schur decomposition*. It states that there exist unitary matrices Q and Z such that the two matrices

$$
\begin{aligned}
Q^H A Z &= R \quad \text{and} && (31)\\
Q^H B Z &= U && (32)
\end{aligned}
$$

are upper triangular. From (31) and (32), it follows easily that

$$\det(A - \lambda B) = \det(QZ^H) \prod_{i=1}^{n} (r_{ii} - \lambda u_{ii}) \ .$$

If there exists k such that $r_{kk} = u_{kk} = 0$, then $\lambda(A, B) = \mathbf{C}$. Otherwise,

$$\lambda(A, B) = \{ \ r_{ii}/u_{ii} \ : \ u_{ii} \neq 0 \ \}$$

is the finite (possibly empty) set of eigenvalues of (A, B).

8.2 The Non-Square Generalized Eigenvalue Problem

Suppose that we want to solve (28) for matrices $A, B \in \mathbf{R}^{m \times n}$ which are not necessarily square. We now show that the non-square problem can be reduced to a square problem. The following reduction is credited in [6] to Professor Gene Golub of the Computer Science Department at Stanford University.

A key fact to the reduction is that rank$(A - \lambda B) < n$ iff rank$((A - \lambda B)^T (A - \lambda B)) < n$ iff there exists $y \neq 0$ such that

$$(A - \lambda B)^T (A - \lambda B) y = 0 \ . \qquad (33)$$

Simple algebraic manipulation of (33) produces the equivalent equation

$$(\lambda^2 C + \lambda D + E) y = 0 \qquad (34)$$

where

$$
\begin{aligned}
C &= B^T B, \\
D &= -(A^T B + B^T A), \qquad \text{and} \\
E &= A^T A \ .
\end{aligned}
$$

If we let $z = \lambda y$, then (34) can be rewritten as

$$
\begin{aligned}
z &= \lambda y \\
\lambda C z + \lambda D y + E y &= 0 \ .
\end{aligned}
$$

These two equations may be written in matrix form as

$$\begin{bmatrix} 0 & I_n \\ E & 0 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \lambda \begin{bmatrix} I_n & 0 \\ -D & -C \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} . \tag{35}$$

If we let

$$\begin{aligned} F &= \begin{bmatrix} 0 & I_n \\ E & 0 \end{bmatrix} \in \mathbf{R}^{2n\times 2n} \quad \text{and} \\ G &= \begin{bmatrix} I_n & 0 \\ -D & -C \end{bmatrix} \in \mathbf{R}^{2n\times 2n} \end{aligned}$$

then (35) is the square generalized eigenvalue problem

$$F \begin{bmatrix} y \\ z \end{bmatrix} = \lambda G \begin{bmatrix} y \\ z \end{bmatrix} . \tag{36}$$

The above discussion shows that λ, y satisfy (33) iff λ, y, z satisfy (36). If $y \neq 0$, then $\begin{bmatrix} y^T & z^T \end{bmatrix}^T \neq 0$, and it follows that $\lambda(A, B) \subset \lambda(F, G)$. The opposite inclusion $\lambda(F, G) \subset \lambda(A, B)$ will follow if $\begin{bmatrix} y^T & z^T \end{bmatrix}^T \neq 0$ implies that $y \neq 0$. But if $y = 0$, then $z = \lambda y = 0$ and $\begin{bmatrix} y^T & z^T \end{bmatrix}^T = 0$. Thus we have proven

$$\lambda(A, B) = \lambda(F, G) .$$

8.3 Computing $x(A, B)$ when $\lambda(A, B) = \mathbf{C}$

When $\lambda(A, B) = \mathbf{C}$, then for each value of λ there is a nonzero vector $x = x(\lambda)$ that satisfies (29). We start by looking for real solutions $x(\lambda)$ which are polynomials in λ. Suppose we seek a solution of degree d:

$$x(\lambda) = x_0 + \lambda x_1 + \lambda^2 x_2 + \cdots + \lambda^d x_d \qquad (x_d \neq 0). \tag{37}$$

Here each x_i is an n-dimensional real vector. Substituting (37) into (29) and equating the coefficients of $\lambda^0, \lambda^1, \ldots, \lambda^d$ to zero, we obtain

$$\begin{aligned} Ax_0 &= 0 \\ Ax_1 - Bx_0 &= 0 \\ Ax_2 - Bx_1 &= 0 \\ &\vdots \\ Ax_d - Bx_{d-1} &= 0 \\ -Bx_d &= 0. \end{aligned} \tag{38}$$

We can rewrite (38) as a homogeneous linear system

$$T_d(A, B)x = 0, \tag{39}$$

where

$$T_d(A,B) = \begin{bmatrix} A & & & & \\ -B & A & & & \\ & -B & A & & \\ & & \ddots & \ddots & \\ & & & -B & A \\ & & & & -B \end{bmatrix} \in \mathbf{R}^{(d+2)m\times(d+1)n}$$

and

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{d-1} \\ x_d \end{bmatrix} \in \mathbf{R}^{(d+1)n}.$$

Any solution x to (39) with $x_d \neq 0$ yields a degree d solution to (29) via (37).

Suppose now that we want to know the degree ϵ_1 of the smallest degree polynomial solution $x(\lambda)$ to (29). We claim that ϵ_1 is the least value of the index d for which the sign $<$ holds in the relation $\mathrm{rank}(T_d(A,B)) \leq (d+1)n$. Otherwise, there would be a non-zero solution to $T_\epsilon(A,B)x = 0$ with $\epsilon < \epsilon_1$. But such a solution vector x would yield an $x(\lambda)$ which has degree less than or equal to ϵ and, therefore, less than ϵ_1.

It turns out that any solution function $x(\lambda)$ to (29) can be written as a linear combination (where the coefficients are polynomials in λ) of linearly independent polynomial solutions (stated without proof in a footnote on p. 29 in [2]). A set of k polynomial vectors $x_1(\lambda),\ldots,x_k(\lambda)$ is linearly independent iff

$$q_1(\lambda)x_1(\lambda) + \cdots + q_k(\lambda)x_k(\lambda) \equiv 0 \qquad \Longleftrightarrow \qquad q_i(\lambda) \equiv 0 \;\; \forall i.$$

A basis of polynomial solutions to (29) is formed as follows. The process begins by choosing a non-zero solution $x_1(\lambda)$ of least degree. During the ith step, choose the smallest degree solution $x_i(\lambda)$ which is linearly independent of the previously chosen solutions $x_1(\lambda),\ldots,x_{i-1}(\lambda)$. If such a solution does not exist, then the process stops. The number of linearly independent solutions to (29) is at most n. If we find p basis solutions $x_1(\lambda),\ldots x_p(\lambda)$ of degrees $\epsilon_1,\ldots,\epsilon_p$, then

$$\epsilon_1 \leq \epsilon_2 \leq \cdots \leq \epsilon_p. \tag{40}$$

A polynomial basis of solutions to (29) is not uniquely defined (to within scale factors). However, any two polynomial bases have the same sequence of degrees (40). The degrees $\epsilon_1,\epsilon_2,\ldots,\epsilon_p$ are called the *column minimal indices* (cmi) of the the pencil $A - \lambda B$. The sequence of distinct cmi's is written as

$$\hat{\epsilon}_1 < \hat{\epsilon}_2 < \cdots < \hat{\epsilon}_u.$$

We denote the multiplicity of the cmi $\hat{\epsilon}_i$ by ρ_i. Note that the number of basis functions is

$$p = \sum_{i=1}^{u} \rho_i.$$

We shall describe an algorithm due to Karcanias ([4]) which computes a basis of solutions from the sequence $\{T_k(A,B)\}$. Although this algorithm is not numerically efficient, it is simple to describe and easy to implement in MATLAB because it uses only basic linear algebra computations on real matrices. See [1] for an efficient algorithm which relies on the generalized Schur form of a matrix pencil. The output of the Karcanias algorithm is a sequence of matrices $N_1, \ldots, N_u$ which defines the polynomial basis $x_1(\lambda), \ldots, x_p(\lambda)$ as follows. The matrix

$$N_i = \begin{bmatrix} x^i_{0,1} & x^i_{0,2} & \cdots & x^i_{0,\rho_i} \\ x^i_{1,1} & x^i_{1,2} & \cdots & x^i_{1,\rho_i} \\ \vdots & \vdots & \cdots & \vdots \\ x^i_{\hat{\epsilon}_i,1} & x^i_{\hat{\epsilon}_i,2} & \cdots & x^i_{\hat{\epsilon}_i},\rho_i \end{bmatrix} \in \mathbf{R}^{(\hat{\epsilon}_i+1)n\times\rho_i}$$

defines the ρ_i basis polynomials of degree $\hat{\epsilon}_i$. Here each x^i_{kj} is a vector in $\mathbf{R}^n$. The jth column of N_i defines the basis function

$$x_{ij}(\lambda) = \sum_{k=0}^{\hat{\epsilon}_i} x^i_{kj}\lambda^k.$$

Before giving the algorithm to compute a basis of solutions, we need to introduce some more notation. Suppose the matrix $M \in \mathbf{R}^{(d+1)n\times r}$ is partitioned as

$$M = \begin{bmatrix} M_0 \\ M_1 \\ \vdots \\ M_d \end{bmatrix},$$

where $M_i \in \mathbf{R}^{n\times r}$. Then we define the matrix $T^k_d(M)$ for $k \geq d$ as

$$T^k_d(M) = \begin{bmatrix} M_0 & & & \\ M_1 & M_0 & & \\ \cdot & M_1 & \ddots & \\ \cdot & \cdot & \ddots & M_0 \\ \cdot & \cdot & & M_1 \\ M_d & \cdot & & \cdot \\ & M_d & & \cdot \\ & & \ddots & \cdot \\ & & & M_d \end{bmatrix} \in \mathbf{R}^{(k+1)n\times(k-d+1)r},$$

Finally, let $r_k = \dim(\mathrm{null}(T_k(A,B)))$ in the algorithm description below.

Algorithm Given a matrix pencil $A - \lambda B \in \mathbf{R}^{m\times n}$ with $\lambda(A,B) = \mathbf{C}$, the following algorithm computes a polynomial basis of solutions $x_1(\lambda), \ldots, x_p(\lambda)$ for the eigenvalue problem $(A - \lambda B)x = 0$.

Step 1 Let $\hat{\epsilon}_1 = \min\{\ k\ :\ r_k > 0\ \}$. Then $\hat{\epsilon}_1$ is the smallest cmi and $\rho_1 = r_{\hat{\epsilon}_1}$ is its multiplicity. Take N_1 to be any basis matrix for $\mathrm{null}(T_{\hat{\epsilon}_1}(A,B))$.

Step 2 Let $\hat{\epsilon}_2 = \min\{ \ k \ : \ r_k > \rho_1(k - \hat{\epsilon}_1 + 1) \ \}$. Then $\hat{\epsilon}_2$ is the next smallest distinct cmi ($\hat{\epsilon}_2 > \hat{\epsilon}_1$) and $\rho_2 = r_{\hat{\epsilon}_2} - \rho_1(\hat{\epsilon}_2 - \hat{\epsilon}_1 + 1)$ is its multiplicity. The vector space $\mathrm{null}(T_{\hat{\epsilon}_2}(A, B))$ can be expressed as a direct sum

$$\mathrm{null}(T_{\hat{\epsilon}_2}(A, B)) = \mathrm{range}(T_{\hat{\epsilon}_1}^{\hat{\epsilon}_2}(N_1)) \oplus \mathcal{V}_2,$$

where $\dim(\mathcal{V}_2) = \rho_2$. Take N_2 to be any basis matrix for the vector space $\mathcal{V}_2$. If Z_2 and R_2 are orthonormal basis matrices for $\mathrm{null}(T_{\hat{\epsilon}_2}(A, B))$ and $\mathrm{range}(T_{\hat{\epsilon}_1}^{\hat{\epsilon}_2}(N_1))$, respectively, then N_2 may be chosen as a basis matrix for $\mathrm{range}(Z_2 - R_2(R_2^T Z_2))$.

$\vdots$

Step i Let $\hat{\epsilon}_i = \min\{ \ k \ : \ r_k > \sum_{j=1}^{i-1} \rho_j(k - \hat{\epsilon}_j + 1) \ \}$. Then $\hat{\epsilon}_i$ is the next smallest distinct cmi ($\hat{\epsilon}_i > \hat{\epsilon}_{i-1} > \cdots > \hat{\epsilon}_1$) and $\rho_i = r_{\hat{\epsilon}_i} - \sum_{j=1}^{i-1} \rho_j(\hat{\epsilon}_i - \hat{\epsilon}_j + 1)$ is its multiplicity. The vector space $\mathrm{null}(T_{\hat{\epsilon}_i}(A, B))$ can be expressed as a direct sum

$$\mathrm{null}(T_{\hat{\epsilon}_i}(A, B)) = \mathrm{range}(H_i) \oplus \mathcal{V}_i,$$

where

$$H_i = \left[\ T_{\hat{\epsilon}_1}^{\hat{\epsilon}_i}(N_1) \ \ T_{\hat{\epsilon}_2}^{\hat{\epsilon}_i}(N_2) \ \ \cdots \ \ T_{\hat{\epsilon}_{i-1}}^{\hat{\epsilon}_i}(N_{i-1}) \ \right] \in \mathbf{R}^{(\hat{\epsilon}_i+1)n \times \sum_{j=1}^{i-1} \rho_j(\hat{\epsilon}_i - \hat{\epsilon}_j + 1)},$$

and $\dim(\mathcal{V}_i) = \rho_i$. Take N_i to be any basis matrix for the vector space $\mathcal{V}_i$. If Z_i and R_i are orthonormal basis matrices for $\mathrm{null}(T_{\hat{\epsilon}_i}(A, B))$ and $\mathrm{range}(H_i)$, respectively, then N_i may be chosen as a basis matrix for $\mathrm{range}(Z_i - R_i(R_i^T Z_i))$.

$\vdots$

End The number of basis polynomials $p \leq \min\{m, n\}$, and therefore the algorithm halts during step i_* if the computation of $\hat{\epsilon}_{i_*}$ considers a candidate $k > \min\{m, n\}$. Obviously, at most $\min\{m, n\}$ steps are required.

A MATLAB implementation of this algorithm is given in appendix B.

9 Least Squares Solutions of Inconsistent Equations

In practice, the equations in system (3) usually come from noisy data, and are therefore inconsistent. The solution to the system can then be required to be optimal in the sense of least squares:

$$\min_{y,z} e^2(y, z) = \min_{y,z} \sum_{i=1}^{l} \sum_{j=1}^{m} \sum_{k=1}^{n} (t_{ijk} y_j z_k - d_i)^2 \ .$$

This leads to a system of equations of the third degree. In fact, by using notations (6) and (7) we have

$$\begin{aligned} e^2(y, z) &= \|B(y)z - d\|^2 = z^T B^T(y) B(y) z - 2z^T B^T(y) d + d^T d \\ e^2(y, z) &= \|C(z)y - d\|^2 = y^T C^T(z) C(z) y - 2y^T C^T(z) d + d^T d. \end{aligned}$$

The normal equations are

$$\begin{aligned}\frac{1}{2}\frac{\partial e^2}{\partial y} &= C^T(z)C(z)y - C^T(z)d = 0\\ \frac{1}{2}\frac{\partial e^2}{\partial z} &= B^T(y)B(y)z - B^T(y)d = 0\ .\end{aligned}$$

The resulting system

$$\begin{aligned}C^T(z)C(z)y &= C^T(z)d\\ B^T(y)B(y)z &= B^T(y)d\end{aligned}$$

has m equations in the first part and n in the second, is biquadratic (that is, separately quadratic in y and in z), and of total degree 3. Note that $(y, z) = (0, 0)$ is always a solution to the normal equations.

10 Related, Harder Problems

10.1 Extreme Singular Values of a Unit-Norm Linear Combination of Matrices

Solving a homogeneous bilinear system $B(y)z = 0$ means finding vectors $y \in \mathbf{R}^m$ for which $B(y)$ does *not* have full column rank. As mentioned in section 3, we may assume $||y|| = 1$ WLOG. Under this assumption, $B(y)$ is a unit norm linear combination of matrices of size $l \times n$:

$$B(y) = y_1 B_1 + \cdots + y_m B_m \qquad ||y|| = 1.$$

If $l < n$, then for every y $B(y)$ is not full column rank. If $l \geq n$, then $B(y)$ is not full column rank iff its smallest singular value $\sigma_n(B(y)) = 0$. Therefore, the homogeneous bilinear system $B(y)z = 0$ with $l \geq n$ has a nontrival solution iff

$$\min_{||y||=1} \sigma_{\min}(y_1 B_1 + \cdots + y_m B_m) = 0.$$

A more general optimization problem is

Problem 1 *Given l matrices $A_1, \ldots, A_l \in \mathbf{R}^{m\times n}$, find a unit norm vector $\alpha = [\ \alpha_1\ \cdots\ \alpha_l\ ]^T$ that minimizes (maximizes) the smallest (largest) singular value of the linear combination $\sum_{i=1}^{l} \alpha_i A_i$.*

As detailed above, an algorithm to solve the minimization version of problem 1 can be applied to determine the existence of a solution to a homogeneous bilinear system.

A solution to the maximization version of problem 1 may also prove useful in the theory of bilinear systems. The singular value decomposition (SVD) of a matrix A yields a least squares solution to the linear system $Ax = b$. Is there a "tensor SVD" of T that yields a least squares solution to $yTz = d$? One possible strategy is to examine matrix SVD properties and try to extend them naturally to order-3 tensors. For example, it is well known that

$$\max_{||x||=||y||=1} x^T A y \;=\; \max_{||x||=||y||=1} \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} x_i y_j \;=\; \sigma_1 \text{ at } x = u_1, y = v_1.$$

A natural extension to the above maximization problem is

Problem 2 *Find*

$$\sigma_1 = \max_{||x||=||y||=||z||=1} \sum_{i=1}^{l} \sum_{j=1}^{m} \sum_{k=1}^{n} t_{ijk} x_i y_j z_k$$

and unit norm vectors $x = u_1, y = v_1, z = w_1$ at which the maximum is achieved.

Do σ_1, u_1, v_1, and, w_1 play a role in a least squares solution of bilinear systems? We currently do not have an answer to this question. The most concrete link that we can establish between problem 2 and bilinear systems is via problem 1 for which we have already established a clear connection with bilinear systems.

We now show that problem 2 reduces to problem 1. Let

$$f(x,y,z) = \sum_{i=1}^{l} \sum_{j=1}^{m} \sum_{k=1}^{n} t_{ijk} x_i y_j z_k,$$

and let A_i be the $m \times n$ matrix

$$A_i = [\ t_{ijk}\ ]_{j=1,\ldots,m}^{k=1,\ldots,n}, \quad i = 1, \ldots, l.$$

Then we may write $f(x,y,z)$ as

$$f(x,y,z) = \sum_{i=1}^{l} b_i x_i = b^T x,$$

where

$$b_i = \sum_{j=1}^{m} \sum_{k=1}^{n} t_{ijk} y_j z_k = y^T A_i z.$$

Clearly,

$$\max_{||x||=||y||=||z||=1} f(x,y,z) = \max_{||x||=||y||=||z||=1} b^T x$$

and

$$\max_{||x||=1} b^T x = ||b|| \quad \text{at} \quad x = \frac{b}{||b||}.$$

Therefore we can solve problem 2 by maximizing $||b||$ for $||y|| = ||z|| = 1$.

Note that

$$||b||^2 = b^T b = \sum_{i=1}^{l} b_i^2 = \sum_{i=1}^{l} y^T A_i z z^T A_i^T y.$$

Therefore we may write

$$||b||^2 = y^T B y,$$

where

$$B = \sum_{i=1}^{l} A_i z z^T A_i^T = \sum_{i=1}^{l} c_i c_i^T, \quad c_i = A_i z.$$

Finally, viewing matrix multiplication as the process of summing outer products of columns and rows, we can express B as the product $B = CC^T$, where $C = [\ c_1 \ \cdots \ c_l\ ]$. Now we know that

$$\max_{||y||=1} ||b||^2 = \max_{||y||=1} y^T B y = \sigma_1(B) \quad \text{at} \quad y = v_1(B), \quad B = V\Sigma V^T.$$

Therefore we want to maximize $\sigma_1(B) = \sigma_1(B(z))$ for $||z|| = 1$.

Using the simple obersvation that $\sigma_1(B) = (\sigma_1(C))^2$, we see that maximizing $\sigma_1(B(z))$ for $||z|| = 1$ is equivalent to maximizing $\sigma_1(C(z))$, the largest singular value of C, for $||z|| = 1$. Note that $C = [\ A_1 z \ \cdots \ A_l z\ ]$ and consider Cw for $||w|| = 1$. We have

$$Cw = \sum_{i=1}^{l} w_i (A_i z) = \left(\sum_{i=1}^{l} w_i A_i \right) z = Dz,$$

where D is the unit norm linear combination of the matrices

$$D = \sum_{i=1}^{l} w_i A_i.$$

Summarizing,

$$\max_{||z||=1} \sigma_1(C(z)) = \max_{||z||=1} \max_{||w||=1} ||C(z)\, w|| = \max_{||w||=1} \max_{||z||=1} ||D(w)\, z|| = \max_{||w||=1} \sigma_1(D(w)).$$

Here we have written $C(z)$ and $D(w)$ to make the dependence of C and D on z and w explicit. We can therefore solve problem 2 by finding a unit norm vector $w = w_*$ which makes the maximum singular value of $D(w) = w_1 A_1 + \cdots + w_n A_n$ as large as possible. This is exactly the maximization version of problem 1.

Once we have w_*, unit norm vectors x_*, y_*, z_* which maximize $f(x, y, z)$ are

$$\begin{aligned} z_* &= v_1(D(w_*)) \\ y_* &= v_1(B(z_*)) \\ x_* &= \frac{b(y_*, z_*)}{||b(y_*, z_*)||}. \end{aligned}$$

The desired maximum value of $f(x, y, z)$ is

$$\max_{||x||=||y||=||z||=1} f(x, y, z) = ||b(y_*, z_*)||.$$

10.2 Simultaneous, Connected Eigenvalue Problems

A bilinear system may be viewed as a linear system with a nonlinear constraint. Let

$$A = \begin{bmatrix} t_{111} & t_{112} & \cdots & t_{11n} & t_{121} & \cdots & t_{12n} & \cdots & t_{1m1} & \cdots & t_{1mn} \\ t_{211} & t_{212} & \cdots & t_{21n} & t_{221} & \cdots & t_{22n} & \cdots & t_{2m1} & \cdots & t_{2mn} \\ & & & & & \vdots & & & & & \\ t_{l11} & t_{l12} & \cdots & t_{l1n} & t_{l21} & \cdots & t_{l2n} & \cdots & t_{lm1} & \cdots & t_{lmn} \end{bmatrix}$$

and

$$x = \begin{bmatrix} y_1 z_1 \\ y_1 z_2 \\ \vdots \\ y_1 z_n \\ y_2 z_1 \\ \vdots \\ y_2 z_n \\ \vdots \\ y_m z_1 \\ \vdots \\ y_m z_n \end{bmatrix}. \tag{41}$$

The ith row of the matrix A is the matrix A_i in row major order, and the vector x collects the mn unknown products $y_j z_k$. The bilinear system (3) may be written as an $l \times mn$ linear system

$$Ax = d. \tag{42}$$

A solution x to the linear system (42) yields a solution to the bilinear system (3) if x is constrained as in (41). If the linear system (42) does not have a solution, then the bilinear system (3) does not have a solution either.

Now consider the homogeneous case $d = 0$. The linearized version of a homogeneous bilinear system is

$$Ax = 0. \tag{43}$$

If $\{ x_1, \ldots, x_f \}$ is a basis for null(A), then

$$x = s_1 x_1 + \cdots + s_f x_f \in \mathbf{R}^{mn}$$

is a solution to (43) $\forall s = [\, s_1 \ \cdots s_f \,]^T$, and $x = 0$ iff $s = 0$. Switching from mn-dimensional vectors to $m \times n$ matrices (where contiguous vector elements form the rows of the corresponding matrix),

$$X = s_1 X_1 + \cdots + s_f X_f \in \mathbf{R}^{m \times n} \tag{44}$$

and $X = 0$ iff $s = 0$. The constraint (41) can be expressed as

$$X = y z^T.$$

In words, X must be a rank one matrix if it yields a nontrivial solution to the homogeneous bilinear system (4).

The condition that X is rank one means that column two of X is a multiple of column one, column three is a mulitple of column two, ..., column n is a multiple of column $n-1$ (and not all columns are zero). If we collect all jth columns of X_i's into an $m \times f$ matrix W_j, then the rank one condition can be expressed as a set of simultaneous connected eigenvalue problems

$$W_{j+1} s = \lambda_j W_j s \qquad \text{for } \ j = 1, \ldots, n-1.$$

Each vector $s \neq 0$ which is a generalized eigenvector for the $n-1$ pairs of matrices (W_2, W_1), (W_3, W_2), ..., (W_n, W_{n-1}) yields a rank one matrix X via (44). If an SVD of X is $X = U\Sigma V^T = \sigma_1 u_1 v_1^T$, then, for example, $(y, z) = (\sigma_1 u_1, v_1)$ is a solution to (4).

The general problem can be stated as follows (in more familiar notation).

Problem 3 *Given $l \geq 2$ matrices $A_1, \ldots, A_l \in \mathbf{R}^{m\times n}$, compute*

$$S = \bigcap_{i=1}^{l-1} \{\; x \neq 0 \;:\; \exists \lambda \in \mathbf{R} \text{ such that } A_{i+1}x = \lambda A_i x \;\} \; .$$

Note that an eigenvector $x \in S$ may correspond to different eigenvalues for different pairs of matrices:

$$S = \{\; x \neq 0 \;:\; \exists \lambda_1, \ldots \lambda_{l-1} \in \mathbf{R} \text{ such that } A_{i+1}x = \lambda_i A_i x \;\} \; .$$

When $l = 2$, problem 3 reduces to the generalized eigenvalue problem discussed in section 8.

11 Conclusion

This report introduced the problem of solving a system of bilinear equations $y^T A_i z = d_i$, for $i = 1, \ldots, l$. If $y \in \mathbf{R}^m$ and $z \in \mathbf{R}^n$, then there are $m + n$ unknowns. We showed that a non-homogeneous bilinear system is no more difficult to solve (in theory) than a homogeneous bilinear system ($d_i = 0 \;\forall i$).

Solving a bilinear system in which one of the unknown vectors is a scalar (i.e. $m = 1$ or $n = 1$) reduces to solving a linear system. The case of a homogeneous system with $m = 2$ or $n = 2$ reduces to a generalized eigenvalue problem for two $l \times n$ or $l \times m$ matrices, respectively. The cases $n, m \geq 3$ remain unsolved, and such homogeneous systems can be viewed as generalized eigenvalue problems for three or more matrices. When there are fewer equations than unknowns in one of y or z, there may well be a theory of multivariable polynomial solutions which is analogous to the one presented in section 8.3. In practice, however, the system is likely to be overdetermined and inconsistent. The best that we could offer in the case of a general bilinear system is a heuristic for finding one non-trivial solution to a homogeneous bilinear system. We also derived the normal equations for a least squares solution to an inconsistent system. However, solving the normal equations appears more difficult than solving a bilinear system.

The solutions to a bilinear system are the common zeros of a collection of multivariable polynomials (i.e. the solution set is an affine variety), namely the second total-degree polynomials $p_i(y_1, \ldots, y_m, z_1, \ldots, z_n) = y^T A_i z - d_i$. The problem of solving bilinear systems, however, is not as hard as solving a general polynomial system because the p_i have a very special structure — all nonconstant terms have the form $y_j z_k$. The key to understanding bilinear systems as well as we understand linear systems is to exploit this structure.

References

[1] T. Beelen and G. Veltkamp. Numerical computation of a coprime factorization of a transfer function matrix. *Systems & Control Letters*, 9(4):281–288, Oct. 1987.

[2] F. R. Gantmacher. *Matrix Theory*, volume II. Chelsea Publishing Company, 1959.

[3] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, second edition, 1989.

[4] N. Karcanias. Minimal bases of matrix pencils: Algebraic Toeplitz structure and geometric properties. *Linear Algebra and its Applications*, 205–206:831–868, July 1994.

[5] N. Karcanias and G. Kalogeropoulos. Right, left characteristic sequences and column, row minimal indices of a singular pencil. *International Journal of Control*, 47(4):937–946, Apr. 1988.

[6] M. Raghavan and B. Roth. Solving polynomial systems for the kinematic analysis and synthesis of mechanisms and robot manipulators. *Transactions of the ASME. Journal of Vibration and Acoustics*, 117(3B):71–79, June 1995.

[7] H. Turnbull and A. Aitken. *An Introduction to the Theory of Canonical Matrices*. Dover Publications, inc., 1961.

A Tensor MATLAB Code

This section lists a few simple MATLAB functions for manipulating tensors of order 3 (that is, three-dimensional). This includes the functions `tmincol` and `tzerocol` that implement the procedure described in section 6.2 that attempts to zero the first column of a tensor by Householder reflections. The code below is not fully-general tensor manipulation code, but just the bare minimum to implement the ideas discussed in these notes for people more familiar with matrix notation than with tensor algebra. Please look at

`http : //robotics.stanford.edu/groups/vision/bilinear/tensor`

for an online version of this code.

A.1 Creation and Access Routines for Tensors

```
******** tzeros.m ********

% make an l x m x n tensor of zeros

function t = tzeros(l,m,n)

if nargin == 1,
  if length(l) == 1,
    m = l;
    n = l;
  elseif length(l) == 3,
    m = l(2);
    n = l(3);
    l = l(1);
  else
    error('input must have either one or three entries')
  end
elseif nargin == 2 | nargin > 3,
```

```
  error('either one or three arguments required')
end

if l*m*n == 0,
  t = [];
else
  cols = m*n+1;
  t = zeros(l,cols);
  t(1,cols) = m;
end
```

```
******** trand.m ********

% make a random l x m x n tensor

function t=trand(l,m,n)

if nargin == 1,
  t = tzeros(l);
elseif nargin == 3,
  t = tzeros(l,m,n);
else
  error('either one or three arguments required')
end

cols = size(t,2)-1;
t(:,1:cols) = rand(l(1),cols);
```

```
******** tstorage.m ********

% return the submatrix indices corresponding to the subtensor of t
% specified by the index range arguments; a range argument can be ':'
% (alas, with the quotes) to specify the whole range; also returns
% the dimensions d of the subtensor

function [rows,cols,d] = tstorage(t,ir,jr,kr)

tcols = size(t,2)-1;
tdim = tdimensions(t);
l = tdim(1);
m = tdim(2);
n = tdim(3);

if ir == ':',
  ir = 1:l;
end
if jr == ':',
  jr = 1:m;
end
if kr == ':',
  kr = 1:n;
end

if any(ir < 1) | any(ir > l) | ...
   any(jr < 1) | any(jr > m) | ...
   any(kr < 1) | any(kr > n),
```

```
  error('subscript out of range')
end

rows = ir;
kcols = m*(kr-1);
cols = (jr(:)*ones(1,length(kcols))) + (ones(length(jr),1)*kcols(:)');
cols = cols(:);

d = [length(ir) length(jr) length(kr)];
```

******** tgetsubtensor.m ********

```
% returns the subtensor of t specified by the given index ranges;
% a range argument can be ':' (alas, with quotes) to denote the whole
% range; warning: matrix outputs are still represented as tensors;
% use tgetmatrix to get a regular matrix

function ts = tgetsubtensor(t,ir,jr,kr)

[r,c,dim] = tstorage(t,ir,jr,kr);
ts = tzeros(dim);
ts(:,1:(size(ts,2)-1)) = t(r,c);
```

******** tputsubtensor.m ********

```
% puts the given tensor ts into the subtensor of t specified by the
% given index ranges; a range argument can be ':' (alas, with quotes)
% to denote the whole range; warning: matrix outputs are still
% represented as tensors; use tgetmatrix to get a regular matrix

function t = tputsubtensor(ts,t,ir,jr,kr)

[r,c] = tstorage(t,ir,jr,kr);
t(r,c) = ts(:,1:(size(ts,2)-1));
```

******** tgetmatrix.m ********

```
% returns slice s orthogonal to dimension d from tensor t;
% the result is a matrix

function a = tgetmatrix(t,s,d)

if (d < 1 | d > 3),
  error('d must be between 1 and 3')
end

tdim = tdimensions(t);
tdim(d) = s;
from = ones(3,1);
to = tdim;
from(d) = s;
to(d) = s;

a = tten2mat(tgetsubtensor(t,from(1):to(1),from(2):to(2),from(3):to(3)));
```

```
******** tputmatrix.m ********

% returns the tensor obtained by inserting the given matrix m
% into slice s orthogonal to dimension d in tensor t

function t = tputmatrix(t,s,d,a)

if (d < 1 | d > 3),
  error('d must be between 1 and 3')
end

[ta to] = tmat2ten(a,d);
from = ones(3,1);
from(d) = s;
to(d) = s;
t = tputsubtensor(ta,t,from(1):to(1),from(2):to(2),from(3):to(3));



******** tdimensions.m ********

% dimensions of a tensor (returns either a row vector or three scalars)

function [d,m,n] = tdimensions(t)

if size(t) == [0 0],
  d = [0 0 0];
else
  d = zeros(1,3);
  d(1) = size(t,1);
  cols = size(t,2);
  d(2) = t(1,cols);
  d(3) = (cols-1)/d(2);
end

if nargout == 3,
  m = d(2);
  n = d(3);
  d = d(1);
elseif nargout ~= 0 & nargout ~= 1,
  error('either zero, one or three output variables are required')
end



******** tsvd.m ********

% computes the singluar values of all the matrices in tensor t

function [Si,Sj,Sk] = tsvalues(t)

[l m n] = tdimensions(t);

Si = zeros(l,min(m,n));
Sj = zeros(m,min(l,n));
Sk = zeros(n,min(l,m));

for i=1:l, Si(i,:) = svd(tgetmatrix(t,i,1))'; end
```

```
for j=1:m, Sj(j,:) = svd(tgetmatrix(t,j,2))'; end
for k=1:n, Sk(k,:) = svd(tgetmatrix(t,k,3))'; end
```

A.2 Tensor-Matrix Type Transformations

```
******** tten2mat.m ********

% transforms a tensor with at least one single-plane dimension into a
% matrix

function a = tten2mat(t)

if isempty(t),
  a = [];
else
  % find the single-plane dimension
  tdim = tdimensions(t);
  [m,d] = min(tdim);
  if m ~= 1,
    error('the order of t is 3: cannot convert to a matrix')
  end

  % remove the tensor information
  a = t(:,(1:size(t,2)-1));

  % 'i' matrices are stored as row vectors
  if d == 1,
    a = reshape(a,tdim(2),tdim(3));
  end
end
```

```
******** tmat2ten.m ********

% transforms a matrix into a tensor with the specified single-plane dimension;
% also returns the dimensions of the new tensor

function [t,tdim] = tmat2ten(a,d)

if isempty(a),
  t = [];
else
  if d == 1,
    tdim = [1 size(a)];
    % 'i' matrices are stored as row vectors
    a = a(:)';
  elseif d == 2,
    tdim = [size(a,1) 1 size(a,2)];
  elseif d == 3,
    tdim = [size(a) 1];
  else
    error('d must be between 1 and 3')
  end

  t = tzeros(tdim);
```

```
  t(:,1:(size(t,2)-1)) = a;
end
```

A.3 Tensor Multiplication by Scalar, Vector, Matrix

```
******** tscale.m ********

% multiply a tensor by a scalar

function t = tscale(t,s)

cols = size(t,2)-1;
t(:,1:cols) = t(:,1:cols)*s;



******** tcontract.m ********

% contract the d-th dimension of tensor t with the vector v to produce
% a matrix

function a = tcontract(t,v,d)

tdim = tdimensions(t);

adim = [];
for p=1:3,
  if p ~= d,
    adim = [adim tdim(p)];
  end
end

a = zeros(adim);

for s=1:tdim(d),
  a = a + v(s)*tgetmatrix(t,s,d);
end



******** tleftmult.m ********

% multiply a tensor t by a matrix m from the left along the slices
% orthogonal to dimension d

function tnew = tleftmult(a,t,d)

olddim = tdimensions(t);
newdim = olddim;
if d == 1,
  cdim = 2;
else
  cdim = 1;
end
newdim(cdim) = size(a,1);

tnew = tzeros(newdim);
```

```
for s=1:olddim(d),
  tnew = tputmatrix(tnew,s,d,a*tgetmatrix(t,s,d));
end



******** trightmult.m ********

% multiply a tensor t by a matrix m from the right along the slices
% orthogonal to dimension d

function tnew = trightmult(t,a,d)

olddim = tdimensions(t);
newdim = olddim;
if d == 3,
  cdim = 2;
else
  cdim = 3;
end
newdim(cdim) = size(a,2);

tnew = tzeros(newdim);

for s=1:olddim(d),
  tnew = tputmatrix(tnew,s,d,tgetmatrix(t,s,d)*a);
end
```

A.4 Householder Transformations for Matrices and Tensors

```
******** house.m ********

% given an n-vector x, this function computes an n-vector v with v(1)
% = 1 such that (I - 2*v*v'/(v'*v))*x is zero in all but the first
% component (from Golub and Van Loan, page 196)

function v = house(x)

n = length(x);
mu = norm(x);
v = x;
if mu ~= 0,
  beta = x(1) + sign(x(1))*mu;
  v(2:n) = v(2:n)/beta;
end
v(1) = 1;



******** lefthouse.m ********

% given an m-by-n matrix A and a nonzero m-vector v with v(1) = 1, the
% following algorithm returns PA where P = I - 2*v*v'/(v'*v)
% (this is row.house(A,v) in Golub and Van Loan, page 197)

function A = lefthouse(v,A)

beta = -2/(v'*v);
```

```
w = beta*A'*v;
A = A + v*w';
```

```
******** righthouse.m ********

% given an m-by-n matrix A and a nonzero n-vector v with v(1) = 1, the
% following algorithm returns AP where P = I - 2*v*v'/(v'*v)
% (this col.house(A,v) in Golub and Van Loan, page 197)

function A = righthouse(A,v)

beta = -2/(v'*v);
w = beta*A*v;
A = A + w*v';
```

```
******** tlefthouse.m ********

% Householder-rotate a tensor t from the left along the slices
% orthogonal to dimension d, using Householder vector h

function t = tlefthouse(h,t,d)

dim = tdimensions(t);
for s =1:dim(d),
  t = tputmatrix(t,s,d,lefthouse(h,tgetmatrix(t,s,d)));
end
```

```
******** trighthouse.m ********

% Householder-rotate a tensor t from the right along the slices
% orthogonal to dimension d, using Householder vector h

function t = trighthouse(t,d,h)

dim = tdimensions(t);
for s =1:dim(d),
  t = tputmatrix(t,s,d,righthouse(tgetmatrix(t,s,d),h));
end
```

A.5 Application Routines

```
******** example.m ********

% create a random tensor system for testing

% tensor dimensions
l=40;
m=10;
n=2;

% elements of the right-hand side
t = trand(l,m,n);
```

```
y=rand(m,1);
z=rand(n,1);
z=z/norm(z);

% right-hand side
b = tcontract(t,y,2)*z;

% Householder-rotate the system so that b becomes (b1, 0, ..., 0)'
h = house(b);
t = tlefthouse(h,t,3); % could also do this along the j dimension
b = lefthouse(h,b);

% reduced homogeneous system
tr = tgetsubtensor(t,2:l,':',':');
```

******** tmincol.m ********

```
% makes the first column of a slice in dimension d of tensor t as small
% as possible by a right Householder rotation of t; also returns the
% Householder vector h, the norm nmin of the first column, and the slice
% number smin

function [t,h,nmin,smin] = tmincol(t,d)

dim = tdimensions(t);

nmin=Inf;
for s=1:dim(d),
  a = tgetmatrix(t,s,d);
  [u sigma v] = svd(a,0);
  c = size(a,2);
  n = sigma(c,c);
  if n < nmin,
    nmin = n;
    smin = s;
    vmin = v(:,c);
  end
end

h = -house(vmin);
t = trighthouse(t,d,h);
```

******** tzerocol.m ********

```
% attempts to zero the first column of tensor t by Householder rotations

function [t,nmin] = tzerocol(t)

threshold = eps;

d = 3;
[t,h,n,s] = tmincol(t,d);
nmin = n;
nold = n+threshold+1;
clc;home;disp(0);disp(nmin)
```

```
iter = 1;
while nold-nmin > threshold,
  d = 5-d;
  [t,h,n,s] = tmincol(t,d);
  if n >= nmin, break, end
  nold = nmin;
  nmin = n;
  home;disp(iter);disp(nmin)
  iter = iter+1;
end
```

B Cspsolve MATLAB Code

This section lists MATLAB routines which implement the algorithm described in section 8.3 to compute a polynomial basis of solutions $x_1(\lambda), \ldots, x_p(\lambda)$ for the eigenvalue problem $(A - \lambda B)x = 0$ when $\lambda(A, B) = \mathbf{C}$ (i.e. when $A - \lambda B$ is a column singular pencil). Please look at

http://robotics.stanford.edu/groups/vision/bilinear/cspsolve

for an online version of this code.

B.1 Basis Computation

```
******** cspsolve.m ********

% Given a matrix pencil (A - lambda B) with lambda(A,B) = C (i.e.
% (A - lambda B) is a column singular pencil), cspsolve(A,B)computes
% a basis of polynomial solutions x_1(lambda), ... , x_p(lambda)
% for the eigenproblem (A - lambda B) x = 0. It uses the helper
% function computeHi.

% Output description:
%
% X is an (epsilon(p)+1) by p matrix, the jth column of which represents
% the jth polynomial solution x_j(lambda) in R^n[lambda]. All entries
% in column j after the first (epsilon(j)+1)n rows are zero. If the
% possibly nonzero rows of column j are partitioned into epsilon(j)+1
% column vectors x_0,j (row 1 to n), x_1,j (row n+1 to 2n), ... ,
% x_epsilon(j),j (row epsilon(j)n+1 to (epsilon(j)+1)n of length n,
% then the jth basis solution is
%
%                           epsilon(j)
%              x_j(lambda) =   sum      x_k,j lambda^k.
%                             k=0
%
% epsilon(j), j=1..p, is the degree of the jth basis solution. Here
% epsilon(1) <= epsilon(2) <= ... <= epsilon(p).
%
% p is the number of basis solutions.
%
% epsilonhat is a vector of the u unique degrees in epsilon. Here
% epsilonhat(1) < epsilonhat(2) < ... < epsilonhat(u).
%
% rho(i), i=1..u, is the multiplicity of the degree in epsilonhat(i).
```

```
% Note that p is the sum of the rho(i)'s.
%
% u is the number of unique degrees in epsilon.

function [X, epsilon, p, epsilonhat, rho, u] = cspsolve(A,B)

[m,n] = size(A);
minmn = min(m,n);
epsilon = zeros(1,minmn);
epsilonhat = zeros(1,minmn); rho = zeros(1,minmn);
Tk = []; X = [];

i = 1; k = -1; rk = -1; p = 0; q = 0;
while (1)
  while ((rk <= (k+1)*p-q) & (k <= min(m,n)))
    Tk = [Tk zeros((k+2)*m,n); zeros(m,(k+2)*n)];
    Tk((k+1)*m+1:(k+2)*m,(k+1)*n+1:(k+2)*n) = A;
    Tk((k+2)*m+1:(k+3)*m,(k+1)*n+1:(k+2)*n) = -B;
    k = k+1;
    rk = (k+1)*n-rank(Tk);
  end
  if (k > min(m,n)) break; end;
  epsilonhat(i) = k;
  rho(i) = rk-((k+1)*p-q);
  M = computeHi(i,n,p,q,X,rho,epsilonhat);
  [U,S,V] = svd(M);
  if (M ~= []) sigma1 = S(1,1); else sigma1 = 0; end
  rankM = sum(diag(S) >  max(size(M))*sigma1*eps);
  N = null(Tk);
  N1 = U(:,1:rankM);
  if (N1 ~= []) P = N-N1*N1'*N; else P = N; end
  [U,S,V] = svd(P);
  if (P ~= []) sigma1 = S(1,1); else sigma1 = 0; end
  rankP = sum(diag(S) >  max(size(P))*sigma1*eps);
  N2 = U(:,1:rankP);
  if (i > 1)
    X = [X; zeros((epsilonhat(i)-epsilonhat(i-1))*n,p)];
  end
  X = [X N2];
  epsilon(p+1:p+rho(i)) = k*ones(1,rho(i));
  p = p+rho(i);
  q = q+rho(i)*epsilonhat(i);
  i = i+1;
end

u = i-1;
epsilon = epsilon(1:p);
epsilonhat = epsilonhat(1:u); rho = rho(1:u);



******** computeHi.m ********

% computeHi constructs the matrix
%
% Hi = [T_(epsilonhat(1))^(epsilonhat(i))(N_1) ...
%       T_(epsilonhat(i-1))^(epsilonhat(i))(N_(i-1))]
%
% for the cspsolve routine. The individual component matrices
% are created via calls to the helper function computeTdk.
```

```
function Hi = computeHi(i,n,p,q,X,rho,epsilonh)

k = epsilonh(i);
Hi = zeros((k+1)*n,(k+1)*p-q);

lastcolX = 0;
lastcolHi = 0;
for l=1:i-1
  d = epsilonh(l);
  Nl = X(1:(d+1)*n,lastcolX+1:lastcolX+rho(l));
  Hi(:,lastcolHi+1:lastcolHi+(k-d+1)*rho(l)) = computeTdk(d,k,Nl);
  lastcolX = lastcolX+rho(l);
  lastcolHi = lastcolHi+(k-d+1)*rho(l);
end



******** computeTdk.m ********

% computeTdk(d,k,M) computes the matrix T_d^k(M). See the
% text of the paper for the definition of T_d^k(M).

function Tdk = computeTdk(d,k,M)

if (k < d) error('must have k >= d'); end

[rowsizeM,colsizeM] = size(M);
if (rem(rowsizeM,d+1) ~= 0)
  error('rowsize of M must be divisible by d+1');
end
n = rowsizeM/(d+1);
r = colsizeM;
Tdk = zeros((k+1)*n,(k-d+1)*r);

for j=1:k-d+1
  Tdk((j-1)*n+1:(j-1)*n+rowsizeM,(j-1)*r+1:j*r) = M;
end
```

B.2 Basis Verification

```
******** checkcspsolve.m ********

% checkcspsolve checks the solution returned by cspsolve for
% the eigenvalues specified in the vector lambda. Each solution
% function is checked at all values in lambda with a call to the
% helper function checkcspsoln. The output maxres is the largest
% value of ||(A - l B) x(l)||_2 over all l in lambda and x(l)
% in X. The output matrix res is defined by
% res(i,j) = ||(A - lambda(i) B) x_j(lambda(i))||_2.

function [maxres,res] = checkcspsolve(A,B,lambda,X,epsilon)

[m,n] = size(A);
p = size(X,2);
nlambda = length(lambda);
res = zeros(nlambda,p);
```

```
maxres = -inf;
for j=1:p
  x = X(1:(epsilon(j)+1)*n,j);
  [maxresj, res(:,j)] = checkcspsoln(A,B,lambda,x,epsilon(j));
  if (maxresj > maxres) maxres = maxresj; end
end
```

```
******** checkcspsoln.m ********

% checkcspsoln(A,B,lambda,x,d) checks the degree d solution
% x(l) for each value l in lambda. The output maxres is the
% largest value of ||(A - l B) x(l)||_2 over all l in lambda.
% The output column vector res is defined by
% res(i) = ||(A - lambda(i) B) x(lambda(i))||_2. Evaluating
% the vector polynomial x at l is done via a call to the
% helper function evalvecpoly.

function [maxres,res] = checkcspsoln(A,B,lambda,x,d)

nlambda = length(lambda);
res = zeros(nlambda,1);

maxres = -inf;
for i=1:nlambda
  res(i) = norm((A-lambda(i)*B)*evalvecpoly(x,d,lambda(i)));
  if (res(i) > maxres) maxres = res(i); end
end
```

```
******** evalvecpoly.m ********

% evalvecpoly(x,d,lambda) returns the vector p obtained by
% evaluating the vector polynomial x of degree d at the
% value lambda.

function p = evalvecpoly(x,d,lambda)

nx = length(x);
if (rem(nx,d+1) ~= 0)
  error('length of x must be divisible by d+1');
end
n = nx/(d+1);
p = zeros(n,1);

lambda_raisedto_j = 1;
for j=0:d
  p = p+lambda_raisedto_j*x(j*n+1:(j+1)*n);
  lambda_raisedto_j = lambda_raisedto_j*lambda;
end
```

B.3 Basis Display

```
******** showcspsolns.m ********

% showcspsolns(X,epsilon,lambda) makes a simultaneous plot of
```

```
% all the solutions functions specified in X. Here epsilon is
% the vector of solution degrees, and lambda is a vector of
% eigenvalues to use in the plot. The showlambda argument is
% optional and indicates whether or not to show the eigenvalues
% in the plot (default is NOT to show the eigenvalues). The real
% work is done in showcspsoln, which makes the plot for one
% solution. See the showcspsoln documentation for more plot
% details. Note that this routine first clears the current figure.

function showcspsolns(X,epsilon,lambda,showlambda)

[m,p] = size(X);
if (rem(m,epsilon(p)+1) ~= 0)
  error('number of rows in X must be divisible by epsilon(p)+1');
end
n = m/(epsilon(p)+1);
if (nargin < 4) showlambda = 0; end

clf;
for j=1:p
  showcspsoln(X(1:(epsilon(j)+1)*n,j),epsilon(j),lambda,showlambda);
  hold on;
end
```

******** showcspsoln.m ********

```
% showcspsoln(x,d,lambda) plots the degree d polynomial solution
% function x(l) = (x1(l),x2(l),...,xn(l)) forall l in lambda. This
% function requires n >= 2 and plots as many of the xi(lambda) as
% possible. The showlambda input argument is optional and defaults
% to 0 (false). If showlambda is true (i.e. any value other than 0),
% then a 3d plot of lambda vs. x1(lambda) vs. x2(lambda) is made. If
% showlambda is false, then a 2d plot x1 vs. x2 is made for n=2 and
% a 3d plot x1 vs. x2 vs. x3 is made for n >= 3.

function showcspsoln(x,d,lambda,showlambda)

nx = length(x);
if (rem(nx,d+1) ~= 0)
  error('length of x must be divisible by d+1');
end
n = nx/(d+1);
if (n < 2) error('n must be at least 2'); end
if (nargin < 4) showlambda = 0; end

nlambda = length(lambda);
V = zeros(n,nlambda);

for j=1:nlambda
  V(:,j) = evalvecpoly(x,d,lambda(j));
end

zlabelstr = '';
if (showlambda)
  graphtitle = 'lambda   vs.   x(lambda) = (x1(lambda),x2(lambda))';
  xlabelstr = 'lambda';
  ylabelstr = 'x1(lambda)';
  zlabelstr = 'x2(lambda)';
```

```
  plot3(lambda,V(1,:),V(2,:));
elseif (n == 2)
  graphtitle = 'eigenvectors x = (x1 x2)';
  xlabelstr = 'x1';
  ylabelstr = 'x2';
  plot(V(1,:),V(2,:));
else
  graphtitle = 'eigenvectors x = (x1 x2 x3)';
  xlabelstr = 'x1';
  ylabelstr = 'x2';
  zlabelstr = 'x3';
  plot3(V(1,:),V(2,:),V(3,:));
end

title(graphtitle);
xlabel(xlabelstr);
ylabel(ylabelstr);
zlabel(zlabelstr);
```