Mathematics of Finance, Courant Institute, Spring 2019
https://www.math.nyu.edu/faculty/goodman/teaching/MathFin2019/MathFinance.html

Linear Algebra, partial review

Linear relations

Linear algerbra is the algebra of linear relationships between variables. A linear relationship between x and y is $y = ax$, for some number a. The subject of linear algebra is linear relationships when there are many x and y variables. Suppose there are n variables $x_1, \ldots, x_n$ and n *coefficients* $a_1, \ldots, a_n$. Then y is a *linear* function of $x_1, \ldots, x_n$ if

$$y = \sum_{j=1}^{n} a_j x_j = a_1 x_1 + \cdots + a_n x_n \; . \tag{1}$$

This relationship may be described by saying that the variables x_j are *factors* that determine y and the coefficients a_j are *factor loadings*. The factor loading a_j determines how the individual factor x_j influences the outcome y.

Linear models like this are used all over. One example is the predictions Netflix makes about how much you will like a movie. The numbers x_j are numbers Netflix has on you, often called *features*. These might be ratings you have given other movies or your age, income (they know stuff), etc. The factor loadings are estimates of how much each factor influences your rating. Take a course on statistics or machine learning to find out how Netflix estimates the factor loadings.

There may be more than one y variable. For example, y_i could be the prediction of how much you will like movie i. Then each y variable would have its own factor loadings

$$y_i = a_{i1} x_1 + \cdots + a_{in} x_n = \sum_{j=1}^{n} a_{ij} x_j \;\; , \quad i = 1, \ldots, m \; . \tag{2}$$

The variables y_j are given in a linear way to, or related in a linear way from, the variables x_i.

It is possible to *compose* linear relations. (The word “compose” usually means “create”, as in composing music, or “arrange”, as in composing yourself after an argument. The mathematical “compose” is vaguely related to the second meaning.) Suppose there are l variables z_k, which are given in terms of y_j using factor loadings (coefficients) b_{jk}. This would be

$$z_k = \sum_{j=1}^{m} b_{kj} y_j \;\; , \qquad \text{for } k = 1, \ldots, l \; . \tag{3}$$

Each variable z_k has its factor loadings, $b_{k,1}, \ldots, b_{k,m}$, from y_j. The variables z_k are indirectly determined by the variables x_i. The factor loadings a_{ij} determine

y_j, then the factor loadings b_{jk} determine z_k. The *composite* relation, which is the *composition* of the two relations (a simpler description is below). There are factor loadings c_{ki} that determine the z_k directly from the x_i as

$$z_k = \sum_{i=1}^{n} c_{ki} x_i \ .$$

The formula is

$$c_{ki} = \sum_{j=1}^{m} b_{kj} a_{ji} \ . \tag{4}$$

This formula (derived below) makes sense because the j *index* in b_{kj} runs from 1 to m and the j index in a_{ji} also runs from 1 to m. You can see this from (2), where there is one sum for each variable y_j, and from (3), where the sum is over the m variables y_j.

As an example, suppose the $x \to y$ and $y \to z$ relations are

$$\begin{aligned} y_1 &= 2x_1 + 3x_2 + 4x_3 \\ y_2 &= 5x_1 + 6x_2 + 7x_3 \\ z_1 &= 8y_1 + 9y_2 \\ z_2 &= 2y_1 + 3y_2 \ . \end{aligned}$$

We can substitute the expressions for y_j into the expressions for z_k and get

$$\begin{aligned} z_1 &= 8(2x_1 + 3x_2 + 4x_3) + 9(5x_1 + 6x_2 + 7x_3) \\ &= (8 \cdot 2 + 9 \cdot 5)x_1 + (8 \cdot 3 + 9 \cdot 6)x_2 + (8 \cdot 4 + 9 \cdot 7)x_3 \\ z_1 &= \qquad 61x_1 + \qquad 78x_2 + \qquad 95x_3 \\ z_2 &= 2(2x_1 + 3x_2 + 4x_3) + 3(5x_1 + 6x_2 + 7x_3) \\ &= (2 \cdot 2 + 3 \cdot 5)x_1 + (2 \cdot 3 + 3 \cdot 6)x_2 + (2 \cdot 4 + 3 \cdot 7)x_3 \\ z_2 &= \qquad 21x_1 + \qquad 24x_2 + \qquad 29x_3 \end{aligned}$$

The general formula (4) gives the same thing:

$$\begin{aligned} c_{11} &= b_{11}a_{11} + b_{12}a_{21} \\ &= 8 \cdot 2 + 9 \cdot 5 \\ &= 61 \\ c_{12} &= b_{11}a_{12} + b_{12}a_{22} \\ &= 8 \cdot 3 + 9 \cdot 6 \\ &= 78 \\ c_{21} &= b_{21}a_{11} + b_{22}a_{21} \\ &= 2 \cdot 2 + 3 \cdot 5 \\ &= 21 \\ &\text{etc.} \end{aligned}$$

This means that

$$
\begin{aligned}
z_1 &= c_{11}x_1 + c_{12}x_2 + c_{13}x_3 \\
&= 61x_1 + 78x_2 + 95x_3 \\
z_2 &= c_{21}x_1 + c_{22}x_2 + c_{23}x_3 \\
&= 21x_1 + 24x_2 + 29x_3 \ .
\end{aligned}
$$

In my long experience as a teacher and a mathematician, I've seen that going through numbers like this makes the concepts clearer than a "clean" abstract proof.

Here is the clean abstract proof of the composition relations (4).

$$
\begin{aligned}
z_k &= \sum_{j=1}^{m} b_{kj} y_j \\
&= \sum_{j=1}^{m} b_{kj} \left(\sum_{i=1}^{n} a_{ji} x_i \right) \\
&= \sum_{j=1}^{m} \sum_{i=1}^{n} b_{kj} a_{ji} x_i \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} b_{kj} a_{ji} x_i \\
&= \sum_{i=1}^{n} \left(\sum_{j=1}^{m} b_{kj} a_{ji} \right) x_i
\end{aligned}
$$

This shows that

$$
z_k = \sum_{i=1}^{n} c_{ki} x_i \ ,
$$

where c_{ki} is given by the composition formula (4).

Matrices and vectors

Linear algebra, working with linear relations, is easier when it is made a little more abstract. This is true about the theory, and for working with them in `R`.

The numbers a_{ij} may be arranged into a *matrix* called A. This is an arrangement of the numbers:

$$
A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \ .
$$

The coefficients that determine y_1 are in the top row (the first row) of A. The coefficients that multiply x_1 are a_{11} (for y_1), a_{21} (for y_2), etc. These are in the first *column* of A. The matrix A has m rows, one for each variable y_i. It has n columns, one for each variable x_j. This is an $m \times n$ matrix. The matrix is *square* if $n = m$. This is common. Otherwise, A is *rectangular*. A matrix with just one column is called a *column vector*. A matrix with just one row is a *row vector*. The numbers a_{ij} are the *entries* of A. The (i, j) entry of A is a_{ij}.

The factors x_j may be arranged to form an $n \times 1$ column vector called x.

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} .$$

The numbers x_j are the *entries*, or *components*, or *coordinates* of x. If we think of x as a matrix with one column, we should write $x_{j,1}$ instead of x_j, but we usually don't. `R` (and Matlab, but not Python), also uses column or row vectors in this way. If `x` is a $n \times 1$ matrix in `R`, then we can "access" x_j using either `x[j]` or `x[j,1]`.

A vector (a column vector in this case) is said to be a *point* in $n-$dimensional space. For example, a point in three dimensional space is determined by its three coordinates. Instead of writing them (x, y, z), we write

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} .$$

It can be confusing to explain that "x" is x_1, "y" is x_2, and "z" is x_3. The $n-$dimensional space is written $\mathbb{R}^n$ (pronounced "r n"). The $\mathbb{R}$ is for real numbers. The n indicates that there are n components.

The operation of *matrix multiplication* corresponds to composition of linear relations. Suppose the composition is

$$x \overset{\{a_{ji}\}}{\longrightarrow} y \overset{\{b_{kj}\}}{\longrightarrow} z .$$

If the factor loadings c_{ki} are given by (4), then (we just saw this)

$$x \overset{\{c_{ki}\}}{\longrightarrow} z .$$

If A is the matrix with entries a_{ji} and B is the matrix with entries b_{kj} and C is the matrix with entries c_{ki} given by (4), then we write

$$C = BA . \tag{5}$$

In the composition formula (refcm), the b_{kj} entries are all on row k of B. This is because k is the same for each entry, while j runs from 1 to m. The A entries a_{ji} are all from column i. The formula says you multiply and add the entries from row k of B with the entries from column i of A. This gives you c_{ki}, which is the entry in row k and column i of C. In the illustration below, the bold element

$\mathbf{c_{ki}}$ is computed from the bold elements $\mathbf{b_{k1}}, \cdots, \mathbf{b_{km}}$ and the bold elements $\mathbf{a_{1i}}, \cdots, \mathbf{a_{m1}}$.

$$\begin{array}{cc} & \begin{array}{ccccc} & & \text{column } i & & \\ & & \downarrow & & \end{array} \\ \text{row } k \rightarrow & \begin{pmatrix} c_{11} & \cdots & c_{1i} & \cdots & c_{1n} \\ \vdots & & \vdots & & \vdots \\ c_{k1} & \cdots & \mathbf{c_{ki}} & \cdots & c_{kn} \\ & & \vdots & & \\ c_{l1} & \cdots & c_{li} & \cdots & c_{ln} \end{pmatrix} \end{array} = \begin{pmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & & \vdots \\ \mathbf{b_{k1}} & \cdots & \mathbf{b_{km}} \\ & & \\ b_{l1} & \cdots & b_{lm} \end{pmatrix} \begin{pmatrix} a_{11} & \cdots & \mathbf{a_{1i}} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{m1} & \cdots & \mathbf{a_{mi}} & \cdots & a_{mn} \end{pmatrix}$$

This works if the m for B (the number of columns) is equal to the m for A (the number of rows). The matrices B and A are *compatible for multiplication* if this is true. If you try to multiply matrices in `R` that are not compatible, you get an error message.

A linear relation (2) may be expressed as matrix multiplication. The matrix $m \times n$ matrix A has the entries a_{ij}. The other "matrix" is the column vector x with entries x_j. This is an $n \times 1$ matrix. The matrix product $y = Ax$ is an $m \times n$ matrix multiplied by an $n \times 1$ matrix, which gives an $m \times 1$ matrix. This is the $m-$component column vector y.

In the example above, $n = 2$ and $m = 2$. The matrix of factor loadings is

$$A = \begin{pmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{pmatrix}$$

The matrix-vector product that gives y is

$$\begin{aligned} y &= Ax \\ \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} &= \begin{pmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \\ \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} &= \begin{pmatrix} 2x_2 + 3x_3 + 4x_3 \\ 5x_1 + 6x_2 + 7x_3 \end{pmatrix} \ . \end{aligned}$$

The first entry of y, which is the first "row" of the column vector is found using the first row of A and the first column of x, which is all of x because it has one column. The second entry y_2 uses the second row of A.

Matrix "multiplication" (the composition formulas (4) is not *commutative.* If A and B are matrices, then $AB \neq BA$ most of the time. In fact, unless A and B are square matrices, and if AB is compatible (the number of columns of A is the number of rows of B) then BA is not compatible. In this case, the product AB is defined, but the product BA is not even defined. If A and B are square matrices of the same size, even then it is unlikely that $AB = BA$. For example

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \ , \quad B = \begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix}$$

has

$$AB = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix} = \begin{pmatrix} 1 \cdot 5 + 2 \cdot 0 & 1 \cdot 0 + 2 \cdot 6 \\ 3 \cdot 5 + 4 \cdot 0 & 3 \cdot 0 + 4 \cdot 6 \end{pmatrix} = \begin{pmatrix} 5 & 12 \\ 15 & 24 \end{pmatrix} .$$

but

$$BA = \begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 5 & 10 \\ 18 & 24 \end{pmatrix} \neq \begin{pmatrix} 5 & 12 \\ 15 & 24 \end{pmatrix} = AB .$$

But matrix multiplication is *associative*. That is, a product with more than two factors can be computed in any way that preserves the order of the matrices. For example, a triple product ABC may be computed as $(AB)C$ or as $A(BC)$. In the first case, you first calculate the matrix product AB and then multiply this by C. In the second case, you first compute the product BC and then multiply by A. For example, consider the triple product

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix} \begin{pmatrix} 0 & 2 \\ 3 & 0 \end{pmatrix} .$$

It may be computed as

$$\left[\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix} \right] \begin{pmatrix} 0 & 2 \\ 3 & 0 \end{pmatrix} = \begin{pmatrix} 5 & 12 \\ 15 & 24 \end{pmatrix} \begin{pmatrix} 0 & 2 \\ 3 & 0 \end{pmatrix} = \begin{pmatrix} 36 & 10 \\ 72 & 30 \end{pmatrix} .$$

It could be computed as

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \left[\begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix} \begin{pmatrix} 0 & 2 \\ 3 & 0 \end{pmatrix} \right] = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 0 & 10 \\ 18 & 0 \end{pmatrix} = \begin{pmatrix} 36 & 10 \\ 72 & 30 \end{pmatrix} .$$

The result is the same. Linear "algebra" (doing manipulations with matrices and vectors) often involves clever use of the fact that matrix multiplication is associative.

Matrix multiplication associativity applies to the special case when one of the matrices is a "vector" (a column vector $= n \times 1$ matrix, or row vector $= 1 \times n$ matrix). For example, if A and B are matrices compatible for multiplication and x is a column vector, then

$$(BA)x = B(Ax) .$$

A few pages ago we used notation $y = Ax$ and $z = By$, so $z = B(Ax)$. We found the composition formulas (4) so that the matrix $C = BA$ would satisfy $z = Cx$. This shows that the matrix multiplication formula was derived to make matrix multiplication associative. *Warning*: the matrix product BA means "first do A, then do B". In diagrams, this is

$$x \xrightarrow{A} Ax \xrightarrow{B} B(Ax) = (BA)x .$$

The *transpose* of a matrix is the matrix you get by reversing the order of the indices. If A an $m \times n$ matrix with entries a_{ji}, then A^t (the transpose of A), is the $n \times m$ matrix with entries a_{ij}. For example,

$$A = \begin{pmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{pmatrix} \iff A^t = \begin{pmatrix} 2 & 5 \\ 3 & 6 \\ 4 & 7 \end{pmatrix}$$

The transpose of A^t is A (see the example):

$$\left(A^t\right)^t = A \; .$$

Two common uses of matrix transpose and matrix multiplication involve column vectors x and y of the same dimension. The *inner product* of x and y is the 1×1 matrix

$$x^t y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n = \sum_{i=1}^{n} x_i y_i \; .$$

For $n = 2$ or $n = 3$ (two or three dimensions), this is sometimes called the *dot product* and written $x \cdot y$. The inner product (dot product) is one of the rare classes of matrix multiplication that is commutative: $x \cdot y = x^t y = y \cdot x = y^t x$. You can check this from the formula $\sum x_i y_i = \sum y_i x_i$. The inner product is (or, of you're a philosopher, may be interpreted as) just a number, which commutes with anything. The *outer product* of two column vectors of size n is the $n \times n$ matrix xy^t. For example, suppose we have $n = 3$ component vectors

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad , \quad y = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$$

Then the outer product is the 3×3 matrix

$$xy^t = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 4 & 5 & 6 \end{pmatrix} = \begin{pmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{pmatrix} \; .$$

The inner product is the number

$$x^t y = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 4 + 10 + 18 = 32 \; .$$

If A and B are compatible for multiplication, then B and A might not be compatible. More properly, the product AB is defined but the product BA is not defined. However, if the product AB is defined, then the product $B^t A^t$ is defined. The relation between these is

$$(AB^t) = B^t A^t \; .$$

You can prove this by writing formulas with sums and indices, or you can check it in simple examples.

Matrix inverse and solving systems of equations

It may happen that you know the outcome of a linear relation and you want to find the input. That is, you know the numbers y_i and you want to find the numbers x_j so that the formulas (2) are satisfied. You probably learned how to do this in high school (at least for $n = 2$). This is possible, and the answer is unique (only one set of x_i is consistent with the known y_j) if $n = m$ (square matrices) and if the matrix A is *invertible*. Usually, a square matrix is invertible unless there is a specific reason for it not to be. You would learn more about this in a linear algebra class. The result is expressions for the variables x_i in terms of the know quantities y_j.

These relations are also linear (it turns out), so they may be written as

$$x_i = \sum_{j=1}^{n} b_{ij} y_j \ .$$

In matrix form, this is $y = Bx$. The relationship between A and B can be found using a diagram

$$x \ \xrightarrow{A} \ Ax \ \xrightarrow{B} \ B(Ax) = (BA)x = x \ .$$

This says that BA is a matrix that "takes" x to itself. This matrix is called the *identity matrix* and written I. Its entries are written δ_{ij}, where

$$\delta_{ij} = 1 \ , \quad \text{if } i = j \ , \quad \delta_{ij} = 0 \ , \quad \text{if } i \neq j \ ,$$

The sums to apply this matrix are (only one term in the sum is different from zero)

$$(Ix)_j = \sum_{i=1}^{n} \delta_{ji} x_i = x_j \ .$$

The matrix relation $BA = I$ is written $B = A^{-1}$. We say that B is the *inverse matrix* for A.

It turns out that if $BA = I$ then $AB = I$ also. Matrix multiplication is not normally commutative, but it is for a matrix and its inverse. You can see this by showing that $BAy = y$ for all y. If x is the unique vector with $Ax = y$, and if $x = By$, then $A(By) = y$. The relation $BA = I$ shows that if B is the inverse of A, then A is the inverse of B.

Matrices and vectors in `R`

`R` makes it possible to work with matrices and vectors using matrix/vector notation. As with arrays, you can first create a matrix of a certain shape, then fill in the numbers. For example, here we make a 2×3 matrix with all ones:

```
> n = 2
> m = 3
> A = matrix(1,n,m)
> A
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
> |
```

Figure 1: The command `matrix(a,n,m)` makes an $n \times m$ matrix with all entries equal to a.

Here we make a more interesting square matrix. We start with all ones, then change the diagonal entries to $2, 3, 4$, then change a_{12} to 5. The `R` function `solve()` finds the inverse matrix. The multiplication symbol for matrices is `%*%`, not just `*`. We check that $BA = I$ by computing the matrix matrix product BA using `B %*% A`. The result may not look like the identity matrix until you look closer.

```
> n = 3
> A = matrix(1,n,n)
> for ( i in 1:n){
+     A[i,i] = 1+i
+ }
> A[1,2] = 5
> A
     [,1] [,2] [,3]
[1,]    2    5    1
[2,]    1    3    1
[3,]    1    1    4
> B = solve(A)
> B
     [,1] [,2] [,3]
[1,]  2.2 -3.8  0.4
[2,] -0.6  1.4 -0.2
[3,] -0.4  0.6  0.2
> A %*% B
              [,1]         [,2]         [,3]
[1,]  1.000000e+00 4.440892e-16 5.551115e-17
[2,] -1.110223e-16 1.000000e+00 5.551115e-17
[3,]  0.000000e+00 0.000000e+00 1.000000e+00
> |
```

Figure 2: Compute the inverse of an interesting matrix and check that it works.

The diagonal entries are 1, as they are supposed to be. But the $(1, 2)$ entry is `4.440892e-16`. The last part, `e-16`, means $\cdot 10^{-16}$. This number is $4.440892 \cdot 10^{-16}$, which is a very small number. Computer calculations are not exact, because the computer is not infinite. Computer calculations usually have *roundoff error*, which comes from rounding. For example $1/3 = .333333\cdots$ might be rounded to $x = .3333$. With this approximation $3x = .9999 \neq 1$. There is a little roundoff error. Typical roundoff error for `R` is on the order of 10^{-16}, so the computed number is consistent with zero.

Figure 3 illustrates column and row vectors. The matrix transpose, A^t, in `R` is `t(A)`. First create a 3×1 matrix x, which is a column vector with 3 components. Then set $x_i = i$. This is printed as a column vector – one column and three rows. The transpose, which is the row vector x^t, is `t(x)`. It is printed as a 1×3 matrix. The matrix xx^t is a $3 \times t$ matrix

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix} .$$

`R` gets the same answer. The matrix $x^t x$ is a 1×1 matrix, which is just a number. The value is

$$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 = 14 .$$

```
> n = 3
> x = matrix(0,n,1)
> x
     [,1]
[1,]    0
[2,]    0
[3,]    0
> for ( i in 1:n){
+     x[i] = i
+ }
> x
     [,1]
[1,]    1
[2,]    2
[3,]    3
> t(x)
     [,1] [,2] [,3]
[1,]    1    2    3
> x %*% t(x)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    3    6    9
> t(x) %*% x
     [,1]
[1,]   14
> 
```

Figure 3: Column and row vectors and transpose.

Figure 4 shows an `R` script that creates bigger matrices related to Assignment 6. The command `print()` prints a matrix in matrix format. You can achieve the same effect using formatted output and `sprintf`, but it's more work. The script plays with 7×7 matrices. You could do 100×100 matrices, but the printout would be hard to read. Lines 6 to 9 make $a_{i,i+1} = a_{i+1,i} = 1$. The *main diagonal* (or just *diagonal*) entries are a_{ii}. The *subdiagonal* entries are just below these, which is $a_{i+1,i}$ (just below a_{ii}). The *superdiagonal* entries are $a_{i,i+1}$ (just to the right, actually). A matrix with non-zeros only here is called *tridiagonal*. Lines 13 to 16 create a column vector x with components given by the geometric series $x_i = (\frac{1}{2})^i$. Finally, we take $A + xx^t$, find the inverse, and check that the inverse is correct. The output is in Figure 5

MatPlay.R

MatPlay.R 〉 No Selection

```r
n = 7                          # matrix size
A = matrix(0,n,n)
for (i in 1:n){                # 2 on the diagonal
    A[i,i] = 2
  }
for ( i in 1:(n-1)){           # 1 on the sub-diagonals
    A[i,i+1] = 1
    A[i+1,i] = 1
  }
print('The tri-diagonal matrix A')
print(A)

x = matrix(0, n, 1)
for ( i in 1:n){               # geometric series, x_i = (1/2)^i
    x[i] = .5**i
  }
A = A + x %*% t(x)             # A with a rank one modification
print('Adding xx^t gives')
print(A)
B = solve(A)
print('B = inverse(A) is')
print(B)
print('Check that AB = I, to within roundoff error')
print(A %*% B)
```

Figure 4: Column and row vectors and transpose.

```
> source("MatPlay.R")
[1] "The tri-diagonal matrix A"
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    2    1    0    0    0    0    0
[2,]    1    2    1    0    0    0    0
[3,]    0    1    2    1    0    0    0
[4,]    0    0    1    2    1    0    0
[5,]    0    0    0    1    2    1    0
[6,]    0    0    0    0    1    2    1
[7,]    0    0    0    0    0    1    2
[1] "Adding xx^t gives"
           [,1]        [,2]         [,3]         [,4]         [,5]         [,6]         [,7]
[1,] 2.25000000 1.125000000 0.0625000000 0.0312500000 0.0156250000 0.0078125000 0.0039062500
[2,] 1.12500000 2.062500000 1.0312500000 0.0156250000 0.0078125000 0.0039062500 0.0019531250
[3,] 0.06250000 1.031250000 2.0156250000 1.0078125000 0.0039062500 0.0019531250 0.0009765625
[4,] 0.03125000 0.015625000 1.0078125000 2.0039062500 1.0019531250 0.0009765625 0.0004882812
[5,] 0.01562500 0.007812500 0.0039062500 1.0019531250 2.0009765625 1.0004882812 0.0002441406
[6,] 0.00781250 0.003906250 0.0019531250 0.0009765625 1.0004882812 2.0002441406 1.0001220703
[7,] 0.00390625 0.001953125 0.0009765625 0.0004882812 0.0002441406 1.0001220703 2.0000610352
[1] "B = inverse(A) is"
           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]       [,7]
[1,]  0.7931885 -0.7202028  0.5803042 -0.4738621  0.3506918 -0.2358855  0.1168973
[2,] -0.7202028  1.4891474 -1.2337210  0.9904801 -0.7411465  0.4948593 -0.2470488
[3,]  0.5803042 -1.2337210  1.8505815 -1.4857202  1.1117198 -0.7422889  0.3705733
[4,] -0.4738621  0.9904801 -1.4857202  1.9916492 -1.4922338  0.9954906 -0.4974113
[5,]  0.3506918 -0.7411465  1.1117198 -1.4922338  1.8677774 -1.2458062  0.6225925
[6,] -0.2358855  0.4948593 -0.7422889  0.9954906 -1.2458062  1.4975649 -0.7486021
[7,]  0.1168973 -0.2470488  0.3705733 -0.4974113  0.6225925 -0.7486021  0.8741975
[1] "Check that AB = I, to within roundoff error"
              [,1]          [,2]          [,3]          [,4]          [,5]          [,6]          [,7]
[1,]  1.000000e+00  3.935654e-17 -5.247539e-17 -8.456777e-18 -8.586881e-17 -3.079134e-17 -1.994932e-17
[2,]  1.384526e-16  1.000000e+00  4.315125e-17 -1.013729e-16 -1.192622e-16 -1.886512e-17 -7.936360e-17
[3,] -3.659182e-17  2.440268e-16  1.000000e+00 -1.374226e-16 -5.854692e-18 -3.407647e-16 -1.125402e-16
[4,]  2.222886e-16 -4.808030e-16  3.880902e-16  1.000000e+00  3.567025e-17 -1.925001e-16  1.181780e-17
[5,]  5.475221e-17 -2.578368e-17  3.867891e-17 -1.679158e-17  1.000000e+00  1.396723e-16 -5.139118e-17
[6,] -1.387779e-17  1.942890e-16  1.110223e-16  3.330669e-16 -4.440892e-16  1.000000e+00  0.000000e+00
[7,]  8.326673e-17  0.000000e+00  1.110223e-16  0.000000e+00  2.220446e-16 -2.220446e-16  1.000000e+00
>
```

Figure 5: Column and row vectors and transpose.