MATH 728D: Machine Learning Lab #2: LINEAR ALGEBRA

John Burkardt

December 5, 2018

Does MATLAB know more linear algebra than we do?

In this lab, we will look at linear algebra in MATLAB. We will see how to create and manipulate vectors and matrices. We will look at the problem of solving linear systems $Ax = b$.
Essentially, MATLAB treats mathematical scalars, vectors and matrices as instances of a MATLAB array, that is, an $m \times n$ numerical object, allowed to take on integer, real or complex values, whose individual values may be accessed by indexing.
An $m \times n$ MATLAB array A, is:

1. a **matrix** if $m > 1$ and $n > 1$;
2. a **row vector** if $m = 1$ and $n > 1$;
3. a **column vector** if $m > 1$ and $n = 1$;
4. a **scalar** if $m = 1$ and $n = 1$;
5. an **empty array** if $m = 0$ or $n = 0$.

It is a common MATLAB convention that the names of matrices are written with capital letters.

A matrix entry is usually accessed as $A(i, j)$: row and column vectors can both be accessed by a single index, as in $v(i)$, while a scalar needs no index at all.

An array can be initialized by listing its elements, with square brackets, one row at a time. Items in the same row are separated by spaces or commas, and a row is terminated by a semicolon:

```
> A = [ 11, 12, 13;
        21, 22, 23;
        31, 33, 33 ];

> row = [ 11, 12, 13 ];

> col = [ 11; 21; 22 ];

> scalar = 11;

> empty = [];
```

Any array can be multiplied or divided by a scalar:

```
> A = 2 * A;
> v = v / 7;
```

Arrays can be added or subtracted only if they have the same shape:

```
> A = B = 3 * C;    % A, B, and C must have same shape
> u = v - 3 * w;    % u, v, and w must have same shape
```

Arrays can be multiplied or divided or exponentiated *elementwise* only if they have the same shape, and if the operator is preceded by a dot:

```
> A = B .* C
> A = B ./ C
> A = B .^ C
> w = u .* v
> w = u ./ v
> w = u .^ v
```

Multiplying (without the dot) an $m1 \times n1$ array A1 times an $m2 \times n2$ array A2 is legal only if $n1 = m2$, and results in an $m1 \times n2$ result A3:

```
> A = B * C;
> b = A * x;
> s = x' * x;
```

Transposing an $m \times n$ array creates an $n \times m$ array; in particular, it converts a row vector to a column vector:

```
> Atrans = A'
> u = v';
```

Often, it is necessary to use the transpose operator in order to make a desired operation legal. For instance, to compute the dot product of two column vectors, we write

```
>  dot = u' * v;           % dot product of two column vectors;
>  Asq = A * A';           % "squares" a matrix
```

Some functions take an matrix, vector, or scalar and return a result of the same shape:

```
> abs(A), ceil(A), cos(A), exp(A), floor(A), round(A), sign(A), ...
> abs(v), ceil(v), cos(v), exp(v), floor(v), round(v), sign(v), ...
> abs(s), ceil(s), cos(s), exp(s), floor(s), round(s), sign(s), ...
```

Some take a matrix and return a row vector of results "per column":

```
> max(A), min(A), mean(A), std(A), sum(A), var(A), ...
```

...or take a vector and return a scalar:

```
> max(v), min(v), mean(v), std(v), sum(v), var(v), ...
```

Some are defined for a matrix or vector and return a single number.

```
> norm(A) ...
> norm(v) ...
```

Some are only defined for a matrix and return a single number.

```
> cond(A), det(A), rank(A), ...
```

Given an $m \times n$ matrix A and an $m \times 1$ column vector b, we can request a solution of the linear system $A * x = b$using the backslash operator:

```
  x = A \ b
```

If A is nonsingular, and $m = n$, then this is the standard solution. Otherwise, x will be a least-squares solution (minimal residual $||A * x - b||$ or minimal $||x||$).
Matrix factorizations:

1. The LU factorization is permutation, unit lower triangular, and upper triangular matrices P, L, U such that $A = P'LU$.

   ```
   > [ L, U, P ] = lu ( A );
   ```

2. The QR factorization is an orthogonal matrix Q and and an upper triangular matrix R such that $A = WR$.

   ```
   > [ Q, R ] = qr ( A );
   ```

3. The SVD factorization is orthogonal U, diagonal D, orthogonal V, such that $A = UDV'$.

   ```
   > [ U, D, V ] = svd ( A );
   ```

1 Solving a Square, Nonsingular System

In the nicest example of $Ax = b$, A is square ($m = n$) and nonsingular. Gauss factorization can find factor matrices so that $A = P'LU$, where:

- P is a permutation matrix;
- L is a unit lower triangular matrix;
- U is an upper triangular matrix.

To solve $Ax = b$, we can instead solve three simple systems:

1. Solve $P'z = b$ for z;
2. Solve $Ly = z$ for y;
3. Solve $Ux = y$ for x.

Suppose that the matrix A and vector x are defined as:

$$A = \begin{pmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{pmatrix}, \qquad x = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}$$

We can compute the product $b = A * x$. Suppose we don't know the value of x. If we are given the values of A and b, we can recover the value of x by solving the linear system $A * x = b$. We can do this using the PLU factorization provided by MATLAB's *lu()* command.

Exercise 1:

1. Define the array **A** and column vector x;
2. Compute **b=A*x;**
3. Use *lu()* to get the PLU factors of **A** ;
4. Solve P'*z=b;
5. Solve L*y=z;
6. Solve U*x=y;
7. Show how close your computed solution **x** is to the exact solution;

2 Rectangular or Singular Systems: Normal Equations

The PLU approach can't be used to solve a system that is singular or rectangular. In cases where $m > n$, a traditional approach is to seek a vector x which does the best job of approximately solving the equations, by minimizing the residual norm $||Ax - b||$. The *normal equations* can be formed by multiplying both sides of the linear system by A', creating an $n \times n$ square system.

$$A'Ax = A'b$$

Now if we think of $A'A$ as our system matrix, and $A'b$ as our right hand side, we can try the PLU factorization approach. We're not solving the original system, and so there will be some error in our results, but it turns out that it's actually the best answer we can get, under the circumstances.
Suppose the matrix A and right hand side b are defined by:

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \qquad b = \begin{pmatrix} 0 \\ 1 \\ 4 \\ 9 \end{pmatrix}$$

We would like to find a vector x so that $Ax = b$ as nearly as possible. Let's see how close we can get, using the normal equations.

Exercise 2:

1. define the matrix $AtA = A'A$;
2. define the right hand side $Atb = A'b$;
3. get the PLU factorization of AtA;
4. Solve $P'z = Atb$ for z;
5. Solve $Ly = z$ for y;
6. Solve $Ux = y$ for x;
7. Print the residual norm,$||A * x - b||$;

3 Rectangular or Singular Systems: The QR Factorization

The normal equations can fail if the square matrix $A' * A$ is singular. This approach also incurs decreased accuracy. A more accurate and robust approach to the problem uses the QR factorization of A.

$$\begin{aligned} A =& Q * R \\ A * x =& b \\ Q * R * x =& b \\ R * x =& Q' * b \end{aligned}$$

And now we solve the upper triangular system involving R.
Suppose the matrix A and right hand side b are defined by:

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{pmatrix} \qquad b = \begin{pmatrix} 0 \\ 1 \\ 4 \\ 9 \end{pmatrix}$$

Exercise 3:

1. get the QR factorization of A;
2. Set $Qtb = Q' * b$;
3. Solve $R * x = Qtb$ for x;
4. Print the residual norm,$||A * x - b||$;

4 Any Linear System: Singular Value Decomposition

The MATLAB **svd()** command takes any $m \times n$ matrix A and returns three factors called the singular value decomposition:

```
> [ U, D, V ] = svd ( A );
```

U and V are $m \times m$ and $n \times n$ orthogonal matrices, respectively, and D is an $m \times n$ diagonal matrix. The diagonal entries of D are known as the *singular values*. To recover A, we multiply the three factors in order, transposing the final factor V:

```
> A = U * D * V';
```

There is lots of information in the singular value decomposition, but let's concentrate on solving a linear system. We would like to write:

$$A^{-1} = VD^{-1}U'$$

but unless D is square and nonsingular, D^{-1} is not defined.
For our general problem, we can use this idea to create something like the inverse matrix. In place of D^{-1}, we set up the $n \times m$ matrix D^{+} in which we invert the nonzero elements of D', leaving any zero entries alone. This allows us to build the $n \times m$ *pseudoinverse* matrix of A:

$$A^{+} = VD^{+}U'$$

The pseudoinverse allows us to write down a formula that gives us an "answer" $x = A^{+}b$ to the problem $Ax = b$, no matter whether A is square or rectangular, singular or nonsingular. The vector x:

- exactly and uniquely solves the system, if A is determined (square and nonsingular);
- minimizes $||Ax - b||$ when A is overdetermined;
- exactly solves the system, and minimizes $||x||$, if A is underdetermined;

Suppose the matrix A and right hand sides $b1$ and $b2$ are defined by

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \qquad b1 = \begin{pmatrix} 10 \\ 30 \\ 50 \end{pmatrix} \qquad b2 = \begin{pmatrix} 10 \\ 30 \\ 51 \end{pmatrix}$$

Using the SVD factorization, seek solutions $x1$ and $x2$ so that $Ax1 = b1$ and $Ax2 = b2$ approximately.

Exercise 4:

1. define A, $b1$ and $b2$;
2. get the U, D, V factors of A;
3. create *Dplus* by transposing D and inverting the nonzero diagonal elements;
4. compute *Aplus* by combining V, *Dplus* and U;
5. compute $x1$ by multiplying *Aplus* times $b1$;
6. report the residual $||Ax1 - b1||$;
7. compute $x2$ by multiplying *Aplus* times $b2$;
8. report the residual $||Ax2 - b2||$;

5 MATLAB Makes it Easy (Too Easy!)

MATLAB provides a single operator to solve all linear systems, whether square or rectangular, singular or not. We can request a solution of $Ax = b$ using the backslash operator:

```
> x = A \ b
```

Remember that a unique and exact solution is mathematically only available in the case where A is square and nonsingular. Otherwise, some kind of least squares approximation will come into play, involving the pseudoinverse.

Also, note that the MATLAB command

```
> Aplus = pinv(A)
```

will automatically compute the pseudoinverse of a matrix for us.
The matrix A and right hand sides $b1$ and $b2$ are defined by

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \qquad b1 = \begin{pmatrix} 10 \\ 30 \\ 50 \end{pmatrix} \qquad b2 = \begin{pmatrix} 10 \\ 30 \\ 51 \end{pmatrix}$$

Exercise 5:

1. define A, $b1$ and $b2$;
2. compute $x1$ using the backslash operator;
3. report the residual $||Ax1 - b1||$;
4. compute $x2$ using the backslash operator;
5. report the residual $||Ax2 - b2||$;