Implementing Linear Algebraic Functions in Matlab

Richard A. Kell

Missouri University of Science and Technology

Table of Contents

Function Descriptions:

dtt.m is a function that returns the determinant of a matrix. This function requires a square matrix as input.

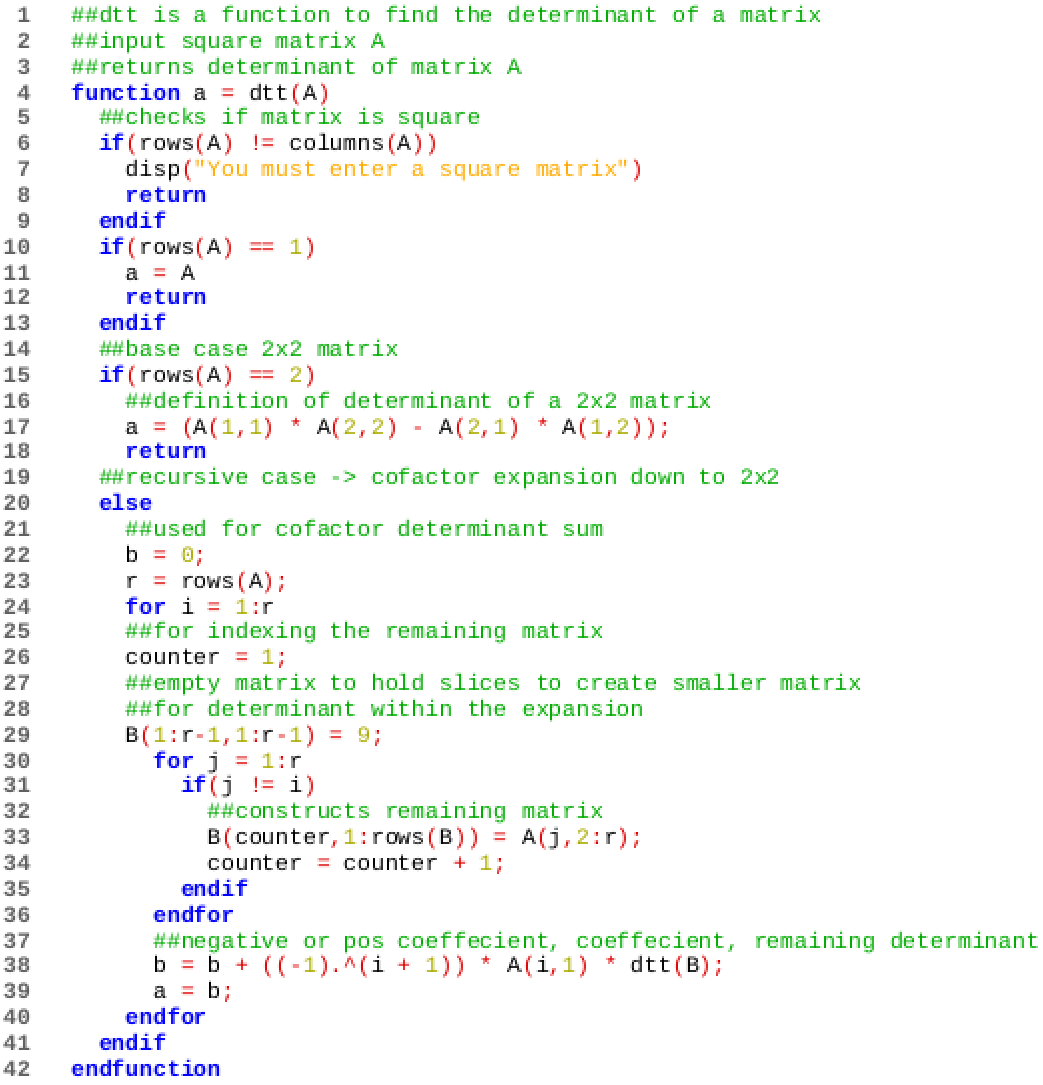
inver.m returns the inverse of a matrix. The input to this function must be square and have a nonzero determinant.

matrixEquation.m outputs the solution vector to the equation Ax = b. Where A is an input matrix and b is an input vector.

orthProj.m returns the orthogonal projection of a vector X onto the vectors input as a matrix A. If the vectors of matrix A are not linearly independent, as required for orthogonal projection, the function will construct a new matrix of linearly independent vectors.

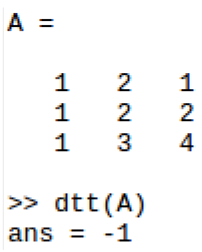
dtt.m

Code:

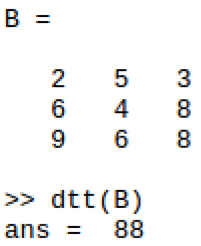


```
##dtt is a function to find the determinant of a matrix
##input square matrix A
##returns determinant of matrix A
function a = dtt(A)
  ##checks if matrix is square
  if(rows(A) != columns(A))
    disp("You must enter a square matrix")
    return
  endif
  if(rows(A) == 1)
    a = A
    return
  endif
  ##base case 2x2 matrix
  if(rows(A) == 2)
    ##definition of determinant of a 2x2 matrix
    a = (A(1,1) * A(2,2) - A(2,1) * A(1,2));
    return
  ##recursive case -> cofactor expansion down to 2x2
  else
    ##used for cofactor determinant sum
    b = 0;
    r = rows(A);
    for i = 1:r
    ##for indexing the remaining matrix
    counter = 1;
    ##empty matrix to hold slices to create smaller matrix
    ##for determinant within the expansion
    B(1:r-1,1:r-1) = 9;
      for j = 1:r
        if(j != i)
          ##constructs remaining matrix
          B(counter,1:rows(B)) = A(j,2:r);
          counter = counter + 1;
        endif
      endfor
      ##negative or pos coeffecient, coeffecient, remaining determinant
      b = b + ((-1).^(i + 1)) * A(i,1) * dtt(B);
      a = b;
    endfor
  endif
endfunction
```

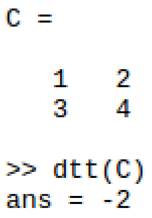
Examples:



```
A =

   1   2   1
   1   2   2
   1   3   4

>> dtt(A)
ans = -1
```



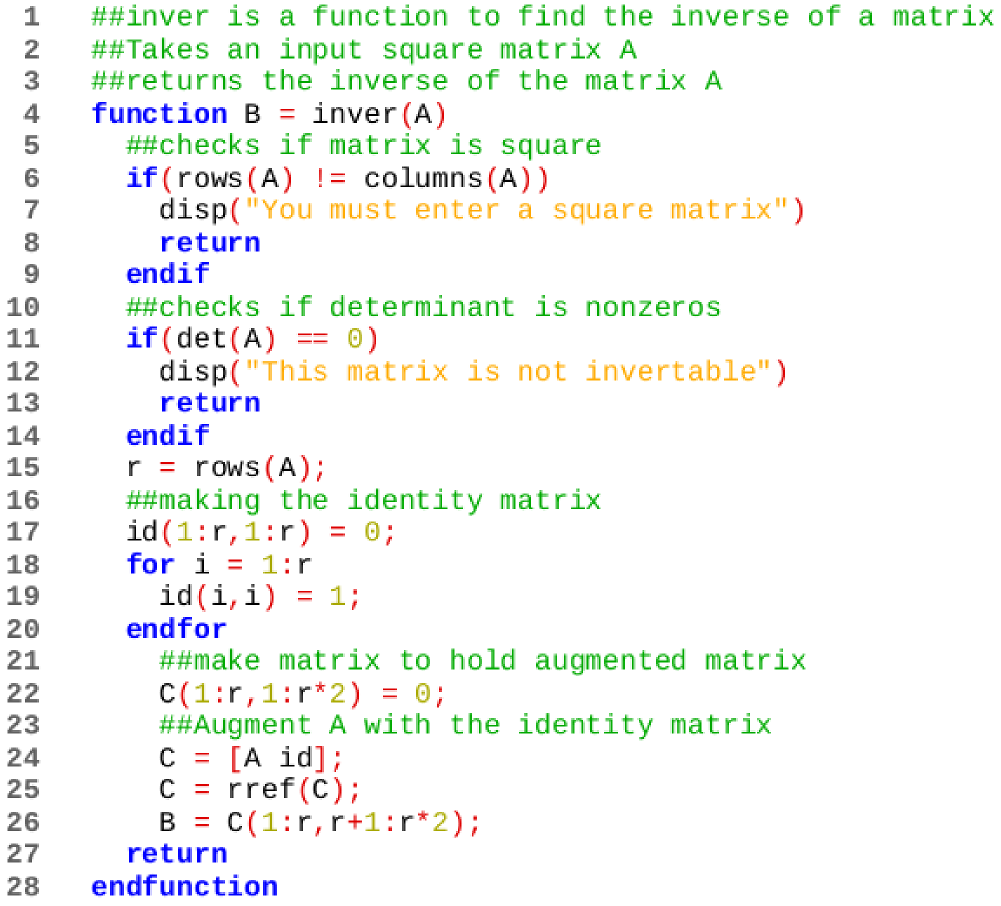
```
B =

   2   5   3
   6   4   8
   9   6   8

>> dtt(B)
ans =  88
```



```
C =

   1   2
   3   4

>> dtt(C)
ans = -2
```

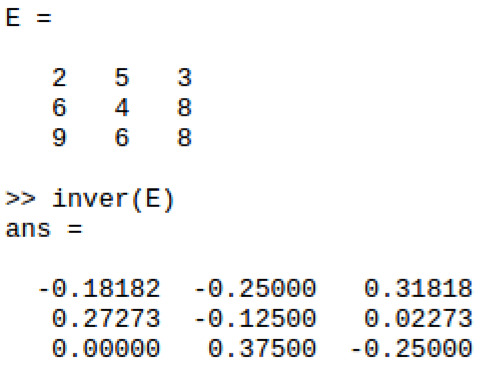
inver.m

Code:

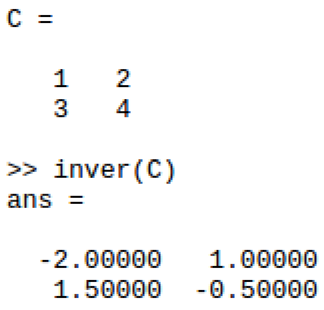


```
##inver is a function to find the inverse of a matrix
##Takes an input square matrix A
##returns the inverse of the matrix A
function B = inver(A)
  ##checks if matrix is square
  if(rows(A) != columns(A))
    disp("You must enter a square matrix")
    return
  endif
  ##checks if determinant is nonzeros
  if(det(A) == 0)
    disp("This matrix is not invertable")
    return
  endif
  r = rows(A);
  ##making the identity matrix
  id(1:r,1:r) = 0;
  for i = 1:r
    id(i,i) = 1;
  endfor
    ##make matrix to hold augmented matrix
    C(1:r,1:r*2) = 0;
    ##Augment A with the identity matrix
    C = [A id];
    C = rref(C);
    B = C(1:r,r+1:r*2);
  return
endfunction
```

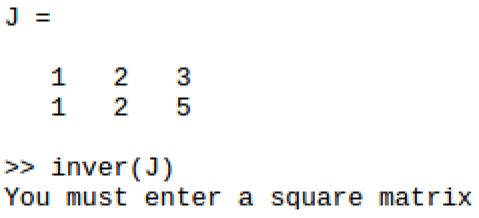
Examples:



```
E =

   2   5   3
   6   4   8
   9   6   8

>> inver(E)
ans =

  -0.18182  -0.25000   0.31818
   0.27273  -0.12500   0.02273
   0.00000   0.37500  -0.25000
```



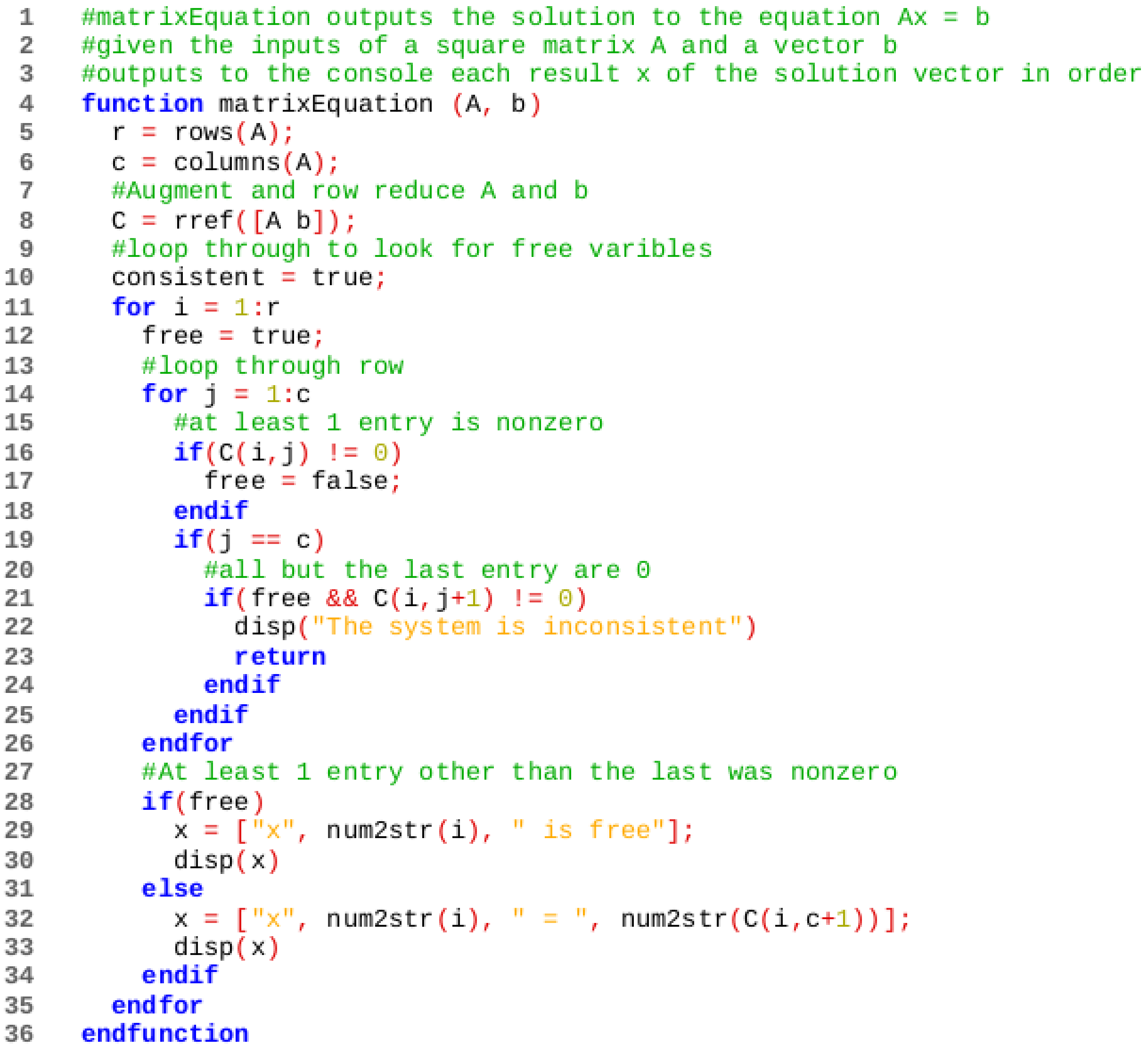
```
C =

   1   2
   3   4

>> inver(C)
ans =

  -2.00000   1.00000
   1.50000  -0.50000
```



```
J =

   1   2   3
   1   2   5

>> inver(J)
You must enter a square matrix
```

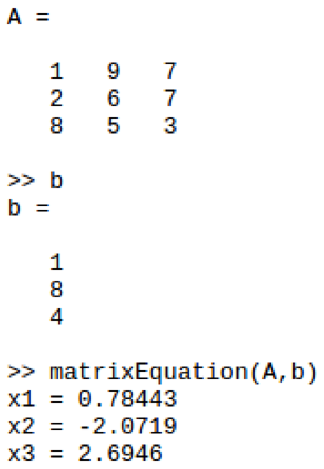
matrixEquation.m

Code:

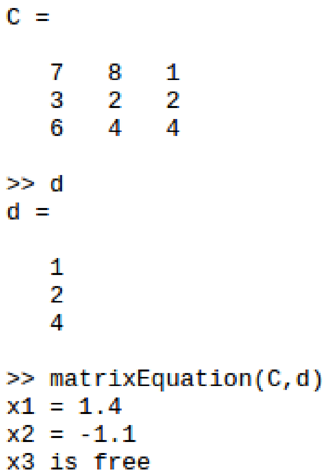


```
#matrixEquation outputs the solution to the equation Ax = b
#given the inputs of a square matrix A and a vector b
#outputs to the console each result x of the solution vector in order
function matrixEquation (A, b)
  r = rows(A);
  c = columns(A);
  #Augment and row reduce A and b
  C = rref([A b]);
  #loop through to look for free varibles
  consistent = true;
  for i = 1:r
    free = true;
    #loop through row
    for j = 1:c
      #at least 1 entry is nonzero
      if(C(i,j) != 0)
        free = false;
      endif
      if(j == c)
        #all but the last entry are 0
        if(free && C(i,j+1) != 0)
          disp("The system is inconsistent")
          return
        endif
      endif
    endfor
    #At least 1 entry other than the last was nonzero
    if(free)
      x = ["x", num2str(i), " is free"];
      disp(x)
    else
      x = ["x", num2str(i), " = ", num2str(C(i,c+1))];
      disp(x)
    endif
  endfor
endfunction
```

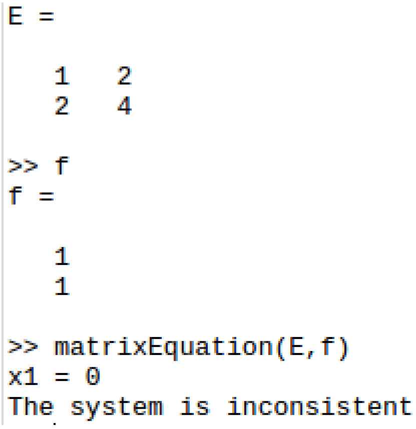
Examples:



```
A =

   1   9   7
   2   6   7
   8   5   3

>> b
b =

   1
   8
   4

>> matrixEquation(A,b)
x1 = 0.78443
x2 = -2.0719
x3 = 2.6946
```

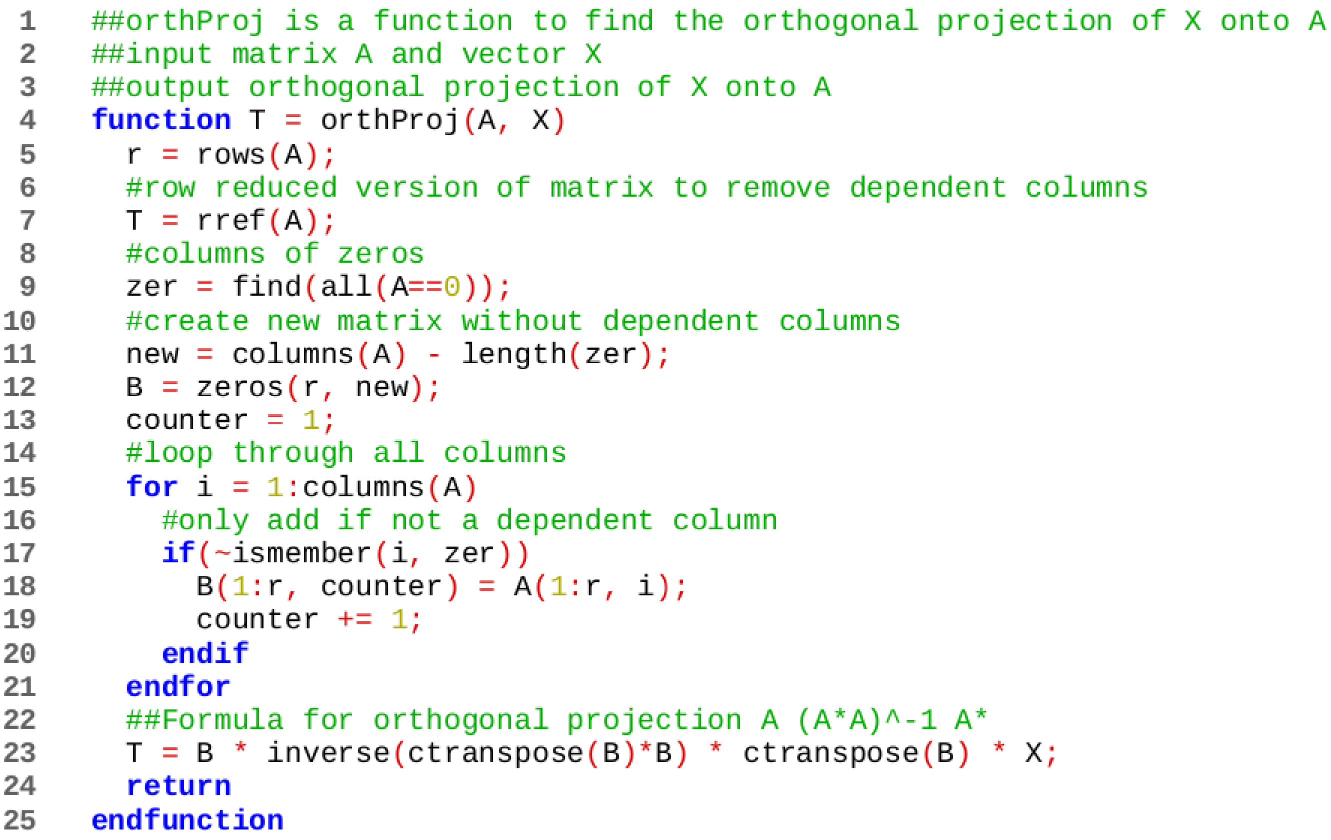


```
C =

   7   8   1
   3   2   2
   6   4   4

>> d
d =

   1
   2
   4

>> matrixEquation(C,d)
x1 = 1.4
x2 = -1.1
x3 is free
```



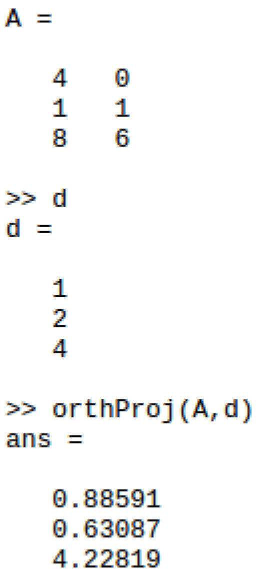
```
E =

   1   2
   2   4

>> f
f =

   1
   1

>> matrixEquation(E,f)
x1 = 0
The system is inconsistent
```

orthProj.m

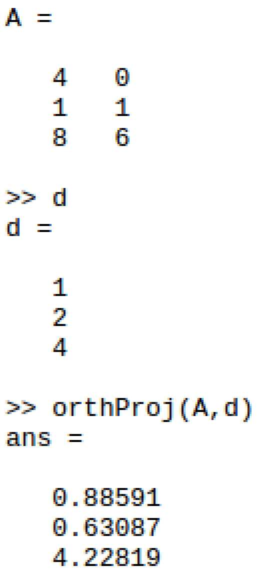


```
##orthProj is a function to find the orthogonal projection of X onto A
##input matrix A and vector X
##output orthogonal projection of X onto A
function T = orthProj(A, X)
  r = rows(A);
  #row reduced version of matrix to remove dependent columns
  T = rref(A);
  #columns of zeros
  zer = find(all(A==0));
  #create new matrix without dependent columns
  new = columns(A) - length(zer);
  B = zeros(r, new);
  counter = 1;
  #loop through all columns
  for i = 1:columns(A)
    #only add if not a dependent column
    if(~ismember(i, zer))
      B(1:r, counter) = A(1:r, i);
      counter += 1;
    endif
  endfor
  ##Formula for orthogonal projection A (A*A)^-1 A*
  T = B * inverse(ctranspose(B)*B) * ctranspose(B) * X;
  return
endfunction
```

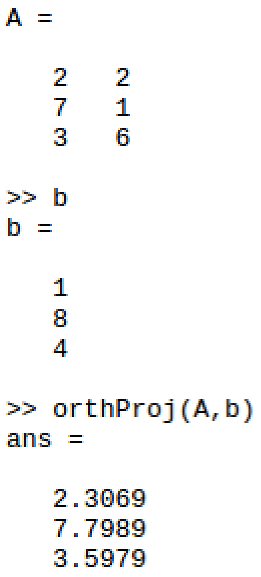
Examples:



```
A =

   4   0
   1   1
   8   6

>> d
d =

   1
   2
   4

>> orthProj(A,d)
ans =

   0.88591
   0.63087
   4.22819
```



```
A =

   4   0
   1   1
   8   6

>> d
d =

   1
   2
   4

>> orthProj(A,d)
ans =

   0.88591
   0.63087
   4.22819
```



```
A =

   2   2
   7   1
   3   6

>> b
b =

   1
   8
   4

>> orthProj(A,b)
ans =

   2.3069
   7.7989
   3.5979
```