```
%Title: "A unimodular demand type which is not a basis change of substitutes"
%Last Edit:27/8/15
%For further details, please contact:
%       Timothy O'Connor, timothy.oconnor@economics.ox.ac.uk
%       Elizabeth Baldwin, e.c.baldwin@lse.ac.uk

%Introduction

%This program checks whether or not there exists a basis change for a
%SPECIFIC 4x9 matrix whose resulting basis change will have max 1 positive and
%max 1 negative entry in each column. As this is for a specific 4x9 matrix,
%this program is NOT immediately generalizable for any 4x9 matrix.

%We start by initiliazing all of our variables (1) before finding all
%possible combinations of the first four columns of our 4x9 matrix with
%<=2 nonzero entries in (2) and (3). We throw out from consideration any
%matrix who would not have the first four columns being invertible (4). (5)
%takes the invertible matrices and fills out the rest of the columns.
%(6),(7), and (8) will filter based on if a matrix generates >2 NZ entries
%in the later part of the matrix. (9) throws out any collection of columns
%that were originally invertible which are no longer invertible. (10) will
%then check to see if the last group of candidate matrices can have at most
%1 positive and at most 1 negative entry in each column.

%Notation:
%A = 4x4 matrix that features the first four columns of the matrix in question
%Dt = full matrix whose columns are the vectors of the demand type
%P = basis change on Dt, so that P.Dt is the matrix that would have at most one positive and at
most one negative entry in each column.

%The unimodular demand type, Dt, that we are investigating is given by:

a=[1,0,0,0];a=a';
b=[0,1,0,0];b=b';
c=[0,0,1,0];c=c';
d=[1,0,0,1];d=d';
A=[a,b,c,d];
Dt=[A,d-a+b,d-a+c,d+b,d+c,d-a+b+c];

%% 1. Initialization of the variables/matrices


%We start by creating the possible basis change matrix P by producing symbolic variables
%that are constrained to the reals.
syms p1_1 real p1_2 real p1_3 real p1_4 real p2_1 real p2_2 real p2_3 real p2_4 real p3_1 real p3_2
 real p3_3 real p3_4 real p4_1 real p4_2 real p4_3 real p4_4 real
p=[p1_1,p1_2,p1_3,p1_4;p2_1,p2_2,p2_3,p2_4;p3_1,p3_2,p3_3,p3_4;p4_1,p4_2,p4_3,p4_4];

%% 2. Initialization of set of matrices with number of nonzero entries <=2
% We gather all possible ways a 4x4 matrix can have at most 2 nonzero entries in each column
% Once we have found them, we will then assume this matrix has the form "PA" and set the remaining
entries of PA to zero.


perms=nchoosek(4,2);%Total number of ways to have 2 Nonzero (NZ) entries in a column
perms_1=nchoosek(4,1);%Total number of ways to have 1 NZ entry in a colum
perm_point=nchoosek((1:4),2);%List of combinations for the 2 NZ selections
perm1_point=nchoosek((1:4),1);%List of combinations for the 1 NZ selection
max_perms=perms^4+perms_1*perms^3*4+6*perms^2*perms_1^2+4*perms*perms_1^3+perms_1^4;%total amount
of combinations
max_matrix_collection=zeros(4,4,max_perms);%Preallocating space



%We collect all the possible matrices with 2 nonzero entries in each column
%such that we cycle through perm_point for each column
n=1;
for i=1:perms
    i_p=perm_point(i,:);
    for j=1:perms
        j_p=perm_point(j,:);
        for k=1:perms
            k_p=perm_point(k,:);
            for h=1:perms
                h_p=perm_point(h,:);
                blank_z=zeros(4,4);
                blank_z(i_p,1)=1;%We input 1 as these will be the locations that are NZ
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
```

```
                max_matrix_collection(:,:,n)=blank_z;%Collect the matrix
                n=n+1;
            end
        end
    end
end

%Now we collect all the combinations with 3 columns having 2 nonzero
%entries and 1 column having 1 nonzero entry

%4th column has 1 nonzero
for i=1:perms
    i_p=perm_point(i,:);
    for j=1:perms
        j_p=perm_point(j,:);
        for k=1:perms
            k_p=perm_point(k,:);
            for h=1:perms_1
                h_p=perm1_point(h);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%3rd column has 1 nonzero
for i=1:perms
    i_p=perm_point(i,:);
    for j=1:perms
        j_p=perm_point(j,:);
        for k=1:perms_1
            k_p=perm1_point(k);
            for h=1:perms
                h_p=perm_point(h,:);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%2nd column has 1 nonzero
for i=1:perms
    i_p=perm_point(i,:);
    for j=1:perms_1
        j_p=perm1_point(j);
        for k=1:perms
            k_p=perm_point(k,:);
            for h=1:perms
                h_p=perm_point(h,:);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%1st column has 1 nonzero
for i=1:perms_1
    i_p=perm1_point(i);
    for j=1:perms
        j_p=perm_point(j,:);
        for k=1:perms
            k_p=perm_point(k,:);
            for h=1:perms
                h_p=perm_point(h,:);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
```

```
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end

%Now we collect all the combinations with 2 columns having 2 nonzero
%entries and 2 columnns having 1 nonzero entry

%1st and 2nd have 1 nonzero
for i=1:perms_1
    i_p=perm1_point(i);%1st column has 1
    for j=1:perms_1
        j_p=perm1_point(j);%2nd column has 1
        for k=1:perms
            k_p=perm_point(k,:);
            for h=1:perms
                h_p=perm_point(h,:);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%1st and 3rd have 1 nonzero
for i=1:perms_1
    i_p=perm1_point(i);
    for j=1:perms
        j_p=perm_point(j,:);
        for k=1:perms_1
            k_p=perm1_point(k);
            for h=1:perms
                h_p=perm_point(h,:);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%1st and 4th have 1 nonzero
for i=1:perms_1
    i_p=perm1_point(i);
    for j=1:perms
        j_p=perm_point(j,:);
        for k=1:perms
            k_p=perm_point(k,:);
            for h=1:perms_1
                h_p=perm1_point(h);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%2nd and 3rd have 1 nonzero
for i=1:perms
    i_p=perm_point(i,:);
    for j=1:perms_1
        j_p=perm1_point(j);
        for k=1:perms_1
            k_p=perm1_point(k);
            for h=1:perms
```

```
                h_p=perm_point(h,:);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%2nd and 4th have 1 nonzero
for i=1:perms
    i_p=perm_point(i,:);
    for j=1:perms_1
        j_p=perm1_point(j);
        for k=1:perms
            k_p=perm_point(k,:);
            for h=1:perms_1
                h_p=perm1_point(h);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%3rd and 4th have 1 nonzero
for i=1:perms
    i_p=perm_point(i,:);
    for j=1:perms
        j_p=perm_point(j,:);
        for k=1:perms_1
            k_p=perm1_point(k);
            for h=1:perms_1
                h_p=perm1_point(h);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end

%And now we collect all the combinations with 3 columns having 1 nonzero
%and one column having 2 nonzero entries

%4th has 2 nonzero
for i=1:perms_1
    i_p=perm1_point(i);
    for j=1:perms_1
        j_p=perm1_point(j);
        for k=1:perms_1
            k_p=perm1_point(k);
            for h=1:perms
                h_p=perm_point(h,:);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%3rd has 2 nonzero
for i=1:perms_1
    i_p=perm1_point(i);
    for j=1:perms
        j_p=perm_point(j,:);
```

```
        for k=1:perms_1
            k_p=perm1_point(k);
            for h=1:perms_1
                h_p=perm1_point(h);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%2nd has 2 nonzero
for i=1:perms_1
    i_p=perm1_point(i);
    for j=1:perms_1
        j_p=perm1_point(j);
        for k=1:perms
            k_p=perm_point(k,:);
            for h=1:perms_1
                h_p=perm1_point(h);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end
%1st has 2 nonzero
for i=1:perms
    i_p=perm_point(i,:);
    for j=1:perms_1
        j_p=perm1_point(j);
        for k=1:perms_1
            k_p=perm1_point(k);
            for h=1:perms_1
                h_p=perm1_point(h);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end

%Now collect with all four having 1 NZ
for i=1:perms_1
    i_p=perm_point(i);
    for j=1:perms_1
        j_p=perm1_point(j);
        for k=1:perms_1
            k_p=perm1_point(k);
            for h=1:perms_1
                h_p=perm1_point(h);
                blank_z=zeros(4,4);%reset
                blank_z(i_p,1)=1;
                blank_z(j_p,2)=1;
                blank_z(k_p,3)=1;
                blank_z(h_p,4)=1;
                max_matrix_collection(:,:,n)=blank_z;
                n=n+1;
            end
        end
    end
end

%And now we have all of the matrices.
%Let us now put it into our actual format (with the pi_j's)

%% 3. Now we take the set of matrices and express it in "PA" matrix form.
```

```
%Preallocate space for the combinations of "PA" - Warning, this is where the
%computing time starts to increase as we loop over symbolic variables
A_combos= sym(zeros(4,4,max_perms));

%This gets us our "PA" matrix for all the combinations of columns with <=2
%nonzero entries
for n=1:max_perms
  for i=1:4
    for j=1:4
        if(max_matrix_collection(i,j,n)==1)%If this entry is NZ, then input the correct pi_j value
            A_combos(i,j,n)=dot(p(i,:),A(:,j));
        else
            A_combos(i,j,n)=0;
        end
    end
  end
end

%This makes sure that the appropriate substitutions are in place. If pi_1
%is equal to zero and if pi_1+pi_4 is NZ then we should just have pi_4 by itself
for n=1:max_perms
    for i=1:4
        for j=1:3
            if A_combos(i,j,n)==0
                A_combos(:,:,n)=subs(A_combos(:,:,n),p(i,j),0);
            end
        end
    end
end

%% 4. Now we filter all the "PA" matrices that would not be invertible

%Preallocating space for our invertible collection
count=0;
for n=1:max_perms
    if rank(A_combos(:,:,n))==4
       count=count+1;
    end
end
inv_A_combos= sym(zeros(4,4,count));

count=0;
for n=1:max_perms
    if rank(A_combos(:,:,n))==4%If this matrix is invertible, we add it to our list
        count=count+1;
        inv_A_combos(:,:,count)=A_combos(:,:,n);
    end
end

%% 5. Now we fill out our collection to include columns 5:9.  That is, we now work with the
collection of matrices of the form "P.Dt".
full_combos=sym(zeros(4,9,count));
for n=1:count
   %From our original matrix, if our first four columns are: a,b,c,d, then:
   c5=inv_A_combos(:,4,n)-inv_A_combos(:,1,n)+inv_A_combos(:,2,n);%column five is d-a+b
   c6=inv_A_combos(:,4,n)-inv_A_combos(:,1,n)+inv_A_combos(:,3,n);%column six is d-a+c
   c7=inv_A_combos(:,4,n)+inv_A_combos(:,2,n);%c7 is d+b
   c8=inv_A_combos(:,4,n)+inv_A_combos(:,3,n);%c8 is d+c
   c9=inv_A_combos(:,4,n)-inv_A_combos(:,1,n)+inv_A_combos(:,2,n)+inv_A_combos(:,3,n);%c9 is d-
a+b+c
   filling=[c5,c6,c7,c8,c9];
   full_combos(:,:,n)=[inv_A_combos(:,:,n),filling];
end

%% 6. Now we check the amount of NZ entries in each column.

alive=0;%new count of matrices that "survive" the filter

%We take our set and filter out the matrices with greater than
%2 NZ entries within a column. One important thing to note that if we have
%pi_j, for j=1,2,3, by itself in an entry, we know that it is NZ. However,
%this is not necessarily the case if we have pi_4 by itself. If all that we
%know is that pi_1+pi_4 is NZ, this does not tell us whether or not pi_4 is
%zero. Therefore, when we count the number of NZ entries in each column, we
%can add to our count if we have a pi_j, j~=4, by itself, and pi_4 will
%only increase our count if pi_1 is zero.

% It is much faster to first count how many matrices will remain after this
% step, preallocate the required space, and then collect those matrices.  Here
% we perform that first count.
```

```
for k=1:count
    flag=0;%Will flag on if we have more than two nonzero entries in a column. We will then not add
 that matrix to our next list.
    for j=5:9
        nz=0;%A counting term for the number of nonzero entries
        for i=1:4
            str=char(full_combos(i,j,k));%We string the entry
            len=length(str);%Find the length of the string so that we can see what the last
character is
            if len<6&&len>1%Provided that the entry is not just a "0"->length==1, or an additive
entry->length>5
                if str(len)~='4'%If we have a pi_j by itself and j~=4, we know it is a NZ entry
                    nz=nz+1;
                elseif full_combos(i,j,k)==full_combos(i,4,k)%If pi_4 is the only entry in the 4th
column, then pi_1=0 and so pi_4 is NZ
                    nz=nz+1;
                end
            elseif len>6%We know look for a double that we know would be NZ. The only double that
we would know is NZ would be the double found in the fourth column (pi_1+pi_4)
                if full_combos(i,j,k)==full_combos(i,4,k)
                    nz=nz+1;
                end
            end
        end
        if nz>2
            flag=1;
        end
        if full_combos(:,j,k)==[0;0;0;0]%After step five of filling out the rest of the columns,
there may be a column with all zeros
            flag=1;
        end
    end
    if flag==0%If we do not have an columns with >2 NZ entries then we increase our count.
      alive=alive+1;
    end
end

%Now that we have the count, preallocate, then write in data - this next part is the same loop as
above, we just write in the data now.
first_filter_num=alive;
first_filter=sym(zeros(4,9,first_filter_num));
alive=0;
for k=1:count
    flag=0;
    for j=5:9
        nz=0;
        for i=1:4
            str=char(full_combos(i,j,k));
            len=length(str);
            if len<6&&len>2
                if str(len)~='4'
                    nz=nz+1;
                elseif full_combos(i,j,k)==full_combos(i,4,k)
                    nz=nz+1;
                end
            elseif len>6
                if full_combos(i,j,k)==full_combos(i,4,k)
                    nz=nz+1;
                end
            end
        end
        if nz>2
            flag=1;
        end
        if full_combos(:,j,k)==[0;0;0;0]
            flag=1;
        end
    end
    if flag==0
      alive=alive+1;
      first_filter(:,:,alive)=full_combos(:,:,k);
    end
end

%% 7. Now subtitute for the matrices with >2 entries when 2 are for sure NZ

%If a column has for sure 2 nonzero entries and then other entries that are
%not formally set to zero, we can now formally set to zero the other
%entries. For example, the entries in a column are:
%[p1_2+p1_4,p2_1,p3_3,p4_2+p2_4]. As the second and third entry are for
%sure NZ we can formally set to zero the values of the first and fourth entry (throughout
```

```
%the entire matrix). Our column is then: [0,p2_1,p3_3,0].

for k=1:first_filter_num
    for j=5:9
        nz=0;
        flag=0;
        for i=1:4
            str=char(first_filter(i,j,k));%As before, we string the entries and inspect the last 
character. If we have pi_4, then it will only be for sure NZ if we know that pi_1 is zero
            len=length(str);%Store the location of the last character
            if len<6&&len>1%If the length is greater than 1 such that it is not "0" and less than 6
 such that it is a single element (pi_j or -pi_j)
                if str(len)~='4'
                    nz=nz+1;
                elseif first_filter(i,j,k)==first_filter(i,4,k)%Else if pi_4 is by itself in the 
fourth column:
                    nz=nz+1;
                end
            elseif len>6%We know look for a double that we know would be NZ. The only double that 
we would know is NZ would be the double found in the fourth column (pi_1+pi_4)
                if first_filter(i,j,k)==first_filter(i,4,k)
                    nz=nz+1;
                end
            end
        end
        if nz==2
            flag=1;
        end
        if flag==1
            %If we have two nonzero entries in this column, we will not go
            %back and formally set to zero any entries that we did not
            %formally know if they were NZ or not
            for i=1:4
                str=char(first_filter(i,j,k));
                len=length(str);
                if len>6 && first_filter(i,j,k)~=first_filter(i,4,k)%If this entry is a double and 
it is NOT the double whose formal value we could know (ie, not pi_1+pi_4)
                    first_filter(:,:,k)=subs(first_filter(:,:,k),first_filter(i,j,k),0);%Then we 
formally set this entry to zero and we make this substitution throughout the entire matrix
                elseif len<6 &&len>1 && str(len)=='4' && 
first_filter(i,j,k)~=first_filter(i,4,k)%If the entry is pi_4 where we did not know whether it was 
NZ prior. This would be the case if we only know that pi_1+pi_4 is NZ
                    first_filter(:,:,k)=subs(first_filter(:,:,k),first_filter(i,j,k),0);%Then we 
formally set set pi_4 to zero and we make this substitution throughout the entire matrix
                end
            end
        end
    end
end

%% 8. Now we filter again based on >2 NZ

%After our substitutions made in 7, we once again filter based on a count
%of the NZ entries in each column. This step is identical to step 6.
alive=0;
for k=1:first_filter_num
    flag=0;
    for j=5:9
        nz=0;
        for i=1:4
            str=char(first_filter(i,j,k));
            len=length(str);
            if len<6&&len>2
                if str(len)~='4'
                    nz=nz+1;
                elseif first_filter(i,j,k)==first_filter(i,4,k)
                    nz=nz+1;
                end
            elseif len>6
                if first_filter(i,j,k)==first_filter(i,4,k)
                    nz=nz+1;
                end
            end
        end
        if nz>2
            flag=1;
        end
    end
    if flag==0
      alive=alive+1;
    end
```

```
end
second_filter_num=alive;
second_filter=sym(zeros(4,9,second_filter_num));

alive=0;
for k=1:first_filter_num
    flag=0;
    for j=5:9
        nz=0;
        for i=1:4
            str=char(first_filter(i,j,k));
            len=length(str);
            if len<6&&len>2
                if str(len)~='4'
                    nz=nz+1;
                elseif first_filter(i,j,k)==first_filter(i,4,k)
                    nz=nz+1;
                end
            elseif len>6
                if first_filter(i,j,k)==first_filter(i,4,k)
                    nz=nz+1;
                end
            end
        end
        if nz>2
            flag=1;
        end
    end
    if flag==0
      alive=alive+1;
      second_filter(:,:,alive)=first_filter(:,:,k);
    end
end

%% 9. Now filter on Invertible grounds


%From this set, we know that every invertible subset of the original 9
%vectors (the columns of Dt) must also be invertible after Dt has been acted on by P. This is
because the product of two invertible matrices is
%invertible.
possib=nchoosek(9,4);%The amount of ways that one can make a 4x4 matrix from 9 columns
possib_list=nchoosek(1:9,4);%The combinations of columns to make a 4x4 matrix
count=0;

for n=1:possib
    check=Dt(:,possib_list(n,:));
    if rank(check)==4%If it is invertible we increase our count
        count=count+1;
    end
end

inv_num=count;
inv_list=zeros(inv_num,4);%Preallocate the space for the lists of combinations of columns that are
invertible.
count=0;
for n=1:possib
    check=Dt(:,possib_list(n,:));
    if rank(check)==4
        count=count+1;
        inv_list(count,:)=possib_list(n,:);%Stores the combination of columns that are invertible
    end
end

%Now that we know which combinations of columns are invertible in our
%original matrix, we check to see if the SAME combination of columns is
%invertible within our list of candidate matrices.
count=0;
for k=1:second_filter_num
    flag=0;
    for c=1:inv_num
        mat=second_filter(:,inv_list(c,:),k);%Selects the combination of columns that should be
invertible
        if rank(mat)~=4%If this is not invertible, we flag.
            flag=1;
        end
    end
    if flag==0%If each combination is invertible, we add it to our count
        count=count+1;
    end
end
```

```
third_filter_num=count;
third_filter=sym(zeros(4,9,third_filter_num));%Preallocate our space and run the loop once more to
store the information
count=0;
for k=1:second_filter_num
    flag=0;
    for c=1:inv_num
        mat=second_filter(:,inv_list(c,:),k);
        if rank(mat)~=4
            flag=1;
        end
    end
    if flag==0
        count=count+1;
        third_filter(:,:,count)=second_filter(:,:,k);
    end
end

%% 10. And now we filter among the combinations that cannot have at most 1 positive and at most 1
negative value in each column

%The general idea is that we will go through each column and make a
%"relationship" between pairs of entries in each column. For instance,
%if one column is [0,p2_2,0,p4_4]', we know that p2_2 and p4_4 must have
%opposite signs (if one is positive, the other is negative). If the column
%was rather [0,p2_2,0,-p4_4], we would then know that p2_2 and p4_4 have the
%same sign (both are positive or both are negative).

%We build these relationships for each column and see if there is a contradiction.
%For example, imagine that our 4x9 matrix includes the following column vectors:
%
%       [0,p2_2,0,p4_4]'        (1)
%       [0,0,p3_3,p4_4]'    (2)
%       [0,p2_2,p3_3,0]'        (3)
%
%From (1) we know that p2_2 and p4_4 are of opposite sign. From (2) we know that
%p3_3 and p4_4 are of opposite sign. We then have a contradiction in (3) as
%(3) says that p2_2 and p3_3 are of opposite sign, and yet (1) and (2) combined
%say that p2_2 and p3_3 must be of the same sign. This sort of contradiction provides
%the foundation for our final filter.

%We first make sure that there are 2 entries in each column that are not formally zero. We
%do not in fact need two non-formally zero entries in each column, but a
%"relationship" can only be formed when there are two non-formally zero entries. It could
%be the case that there are 3 symbolic zeros, but as it so happens, after
%all of the filtering it is the case that we have exactly two entries that
%are formally zero. We show this here:
check=0;
for n=1:third_filter_num
    for j=1:9
        nz_num=length(find(third_filter(:,j,n)));%counts the zeros in the column
        if nz_num~=2%If we do not have two entries formally set to zero:
            check=check+1;
        end
    end
end

%As one can see, check==0 such that we have exactly two non-formally zero entries
%in each column.

%A second check that we will have to make (to ensure that the next loop is
%specified correctly) is to make sure that we do not have pi_4's alone in a
%column where we do not know whether or not it is NZ. As we add
%relationships based on having the same or opposite sign, we want to make
%sure that we do not make a comparison with pi_4 when it may be zero.
%Therefore:
check2=0;
for k=1:third_filter_num
    flag=0;
    for j=5:9
        for i=1:4
            str=char(third_filter(i,j,n));%As before, we string the entries and inspect the last
character.
            len=length(str);%Store the location of the last character
            if len<6&&len>1%If the length is greater than 1 such that it is not "0" and less than 6
 such that it is a single element (pi_j or -pi_j)
                if str(len)=='4'
                    if third_filter(i,j,n)~=third_filter(i,4,n)%Such that whether pi_4=0 is then
unknown
                        flag=1;
```

```
                    end
                end
            end
        end
    end
    if flag==1
        check2=check2+1;
    end
end
%As we can see, check2==0. Therefore, we do not have any indeterminate
%pi_4's. All the pi_4's that are alone in a column will be definitively NZ.


for n=1:third_filter_num
    %We first build our collection of same/opposite relations between our
    %individual elements. "Opps" is a list of element pairs that we know
    %are opposite sign from each other. "Same" is a list of element pairs
    %that we know have the same sign. Note that we only add single element
    %relations. We shall skip doubles as we will have enough information
    %from the single element pairs.


    flag=0;%Will flag on if we have a contradiction
    opps_n=0;%The count of opposite relations
    same_n=0;%The count of same relations
    for j=1:9
        indx=find(third_filter(:,j,n)~=0);%We find all of our nonzero entries
        str1=char(third_filter(indx(1),j,n));%We string each entry
        str2=char(third_filter(indx(2),j,n));
        if length(str1)+length(str2)==9 %With a length of nine, we know that only one of the
entries has a negative sign in front
            same_n=same_n+1;%As one has a negative sign, we know that the elements must share the
SAME sign
            same(same_n,:)=[third_filter(indx(1),j,n),third_filter(indx(2),j,n)];%We add this
relation to our SAME list
        elseif length(str1)+length(str2)==8||length(str1)+length(str2)==10
            %If the length is 8, neither element has a negative sign. If it
            %is 10, they both have a negative sign. Either way, we know
            %that the elements then must be of OPPOSITE sign
            opps_n=opps_n+1;
            opps(opps_n,:)=[third_filter(indx(1),j,n),third_filter(indx(2),j,n)];
        end
    end
    %Now we clean so that we do not get negative signs in front of elements
    for i=1:same_n
        for j=1:2
            negcheck=char(same(i,j));
            if length(negcheck)==5
                same(i,j)=(-1)*same(i,j);
            end
        end
    end
    for i=1:opps_n
        for j=1:2
            negcheck=char(opps(i,j));
            if length(negcheck)==5
                opps(i,j)=(-1)*opps(i,j);
            end
        end
    end
    %Now we do a first check to see whether or not there will be any
    %contradictions directly
    flip=[opps(:,2),opps(:,1)];%also check the flip as p1_1, p1_2 would be the same as p2_1, p1_1
    check=intersect(same,opps,'rows');
    check2=intersect(same,flip,'rows');
    if ~isempty(check)||~isempty(check2)
        %If the intersection of the same and opposite relation is non-empty
        %such that a pair of elements is said to have both the same and
        %opposite sign, we know this matrix fails.
        flag=1;%And then we will skip the next loop and move on to the next matrix.
    end

    %We will now add to our list of same/opposites via transitivity
    loop_count=1;
    %The loop_count will count the number of times that we cycle through
    %our same and opposite lists, adding new relationship via transitivity.
    %We have an arbitrary limit on the loop count so that we do not loop
    %indefinitely if there was some error prior.
    while flag==0&&loop_count<5
        for i=1:length(opps)-1
            %As we will look at occurences of a selected element later in
```

```
            %the same and opposite lists, we do not look for more
            %occurences when we are at the end of the list.
            for j=1:2
                search=opps(i,j);
                %'Search' is the first element and we will look for more
                %occurences of this element down the list in opposite and
                %same. 'Pairing' is the element that we know has an
                %opposite relationship with 'search'.
                if j==1
                    pairing=opps(i,2);
                else
                    pairing=opps(i,1);
                end
                newopps=opps((i+1):length(opps),1:2);%Searches down the list from where we are
currently
                [r,c]=find(newopps==search);%returns index within the opposite to build up same, as
 the opposite of opposites is the same
                [r1,c1]=find(same==search);%returns index within same to build up opps, as the
opposite of same is opposite.
                if ~isempty(r)%if we find an entry down the opposite list such that 'search' is
located somewhere down the list
                    new_rels=sym(zeros(length(r),2));%Then the new relations that we make will be
added to same.
                    for b=1:length(r)%As we may have more than occurence of the 'search' element
                        if c(b)==2%If 'search' is in the second column, make the new relationship
with the element in the first column
                            new_rels(b,1:2)=[pairing,newopps(r(b),1)];%stores the new relationship
                        else%If 'search' is in the first column, make the new relationship with the
 element in the second column
                            new_rels(b,1:2)=[pairing,newopps(r(b),2)];
                        end
                    end
                    same=[same;new_rels];%As the opposite of opposite is same, we add to our same
list.
                end
                if ~isempty(r1)%if we find an entry down the same list...
                    new_rels=sym(zeros(length(r),2));%Then the new relations that we make will be
added to opposite.
                    for b=1:length(r1)
                        if c1(b)==2
                            new_rels(b,1:2)=[pairing,same(r1(b),1)];%stores the new relationship
                        else
                            new_rels(b,1:2)=[pairing,same(r1(b),2)];
                        end
                    end
                    opps=[opps;new_rels];%As the opposite sign of the same sign is opposite, we add
 to our opposite list.
                end
            end
        end
        flip=[opps(:,2),opps(:,1)];%We flip the list again before checking intersections
        check=intersect(same,opps,'rows');
        check2=intersect(same,flip,'rows');
        if ~isempty(check)||~isempty(check2)
        %If the intersection of the same and opposite relation is non-empty
        %such that a pair of elements is said to have both the same and
        %opposite sign, we know this matrix fails.
            flag=1;
        end
        loop_count=loop_count+1;
    end

    if flag==0%if we looped through 5 times and still were not able to find a contradiction..
        candidate_matrix=n;
        disp('Error: Matrix #n was not thrown out of consideration');
        %Therefore, if there is a printed message, we know that we were
        %unable to throw out the n'th matrix in third_filter.
    end
end
```