## Load the Linear Algebra Package and the Plots Package.

```maple
with(linalg): with(plots):
```

## This builds the diagonal matrix D on the left hand side of the equation - voice sources only

```maple
MakeDMat := proc(alpha,C,N)
local row, col, dmat;
dmat := matrix((N + 1),(N + 1));
for row from 1 to (N + 1) do
for col from 1 to (N + 1) do
if ((col < row) or (col > row)) then
    dmat[row, col] := 0;
else
    dmat[row,col] := ((row - 1) - C)*alpha;
fi;
od;
dmat;
end:
```

## This builds the triadiagonal matrix M - on the right hand side of the equation

```maple
MakeMMat := proc(alpha,lambda,C,N)
local row, col, dmat;
dmat := matrix((N + 1),(N + 1));
for row from 1 to (N + 1) do
for col from 1 to (N + 1) do
if ((col < (row - 1)) or (col > (row + 1))) then
    dmat[row, col] := 0;
elif (col = (row - 1)) then
    dmat[row,col] := (row - 1)*alpha;
elif (col = (row + 1)) then
    dmat[row,col] := (N + 1 - row)*lambda;
else dmat[row,col] := - ((row - 1)*alpha + (N + 1 - row)*lambda);
fi;
od;
edmat;
end:
```

## This finds the index of the zero eigenvalue.  We need a user defined error
## for the floating point approximations.

```maple
ZeroEigenvalueIndex := proc(solnArray, sols, error)
local i;
for i from 1 to sols while
((solnArray[i][1] > error) or (solnArray[i][1] < -error)) do; od;
if (i > sols) then 0; else i; fi;
end:
```

## This is just a utility function - there may be a built in that does it...

```maple
MakeMatrixFromVector := proc(vect, dim)
local mat, i;
mat := matrix(1,dim);
for i from 1 to dim do
    mat[1,i] := vect[i];
od;
RETURN(evalm(mat));
end:
```
This finds the largest negative eigenvalues and returns the pair consisting of its INDEX and its value. Notice the user-defined error to allow for floating point approximations.

```maple
GetLargestNegativeEigenvalue := proc(solnArray, dim, error)
local i, candidate, index, approx;
candidate := - infinity;
index := 1;
for i from 1 to dim do
    approx := evalf(solnArray[i][1]);
    if ((approx < -error) and (approx > candidate)) then
        candidate := approx;
        index := i;
    fi;
od;
[index, solnArray[index][1]];
end:
```

This just extracts the eigenvector in the array of triples, given the index.

```maple
GetEigenvector := proc(solnArray, index);
solnArray[index][3][1];
end:
```

This extracts the list of indices corresponding to negative eigenvalues; Notice the error term to handle floating point approximations.

```maple
NegEigenvalueIndices := proc(solnArray, sols, error)
local i, indexList;
indexList := NULL;
for i from 1 to sols do
    if (evalf(solnArray[i][1]) < -error) then #dangerous...!!
        indexList := indexList, i;
    fi;
od;
[indexList];
end:
```

Recall that the solution will be made up with the constant term (picked up elsewhere) and the negative eigenvalue eigensolutions.

```maple
MakeSolution := proc(solnArray, dim, negList)
local soln, i;
soln := matrix(1, dim);
for i from 1 to dim do soln[1,i] := 0; od;
for i from 1 to dim do
    if member(i, negList) then
        soln := matadd(soln,
        a.i*MakeMatrixFromVector(evalm(solnArray[i][3][1]),dim)*
        exp(evalm(solnArray[i][1])*x));
    fi;
od;
evalm(soln);
end:
```

Utility function to check if matrix has zero-sum rows...

```maple
SumRow := proc(m, dim, row)
local i, partial;
partial := 0;
for i from 1 to dim do
    partial := partial + m[row,i];
od;
```
simplify(partial);
end:

## Utility function for conversion.

> BitsPerFrameToCellsPerSecond := proc(framesPerSecond, bits)
  bits*framesPerSecond/(48*8);
end: