A Recursive Algorithm to find the Determinant

CIS008-2 Logic and Foundations of Mathematics

David Goodwin

david.goodwin@perisic.com

University of Bedfordshire

11:00, Tuesday 6th March 2012

Outline

Outline

Forming a recursive algorithm for a Determinant

- Finding the determinant involves making smaller and smaller minors, until we find 2×2.
- 2×2 can be solved directly:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

- we start by defining the determinant as a function that will solve for a 2×2 case:

```
function D = Det_algo (A)
%make sure not to use a function name that may
%already be a matlab function, like det
%predefine D for efficiency
D = zeros(size(A,1)-1,size(A,2)-1);
if size(A,1)~=size(A,2)
  message='matrix not square; no proper determinant';
else
  if max(size(D))==2
    D = (A(1,1)*A(2,2))-(A(1,2)*A(2,1));
  end
end
```

Forming a recursive algorithm for a Determinant

- The function on the previous page should find the determinant for a 2×2 matrix
- Also test if the matrix is square, to avoid improper input errors.
- The next stage would be to recursively use the Det_algo function to find the determinant for a 3×3 matrix.
- The algorithm would not enter the inner if statement since it is not a 2×2 matrix, so we need an else condition to account for when the matrix is not 2×2.
- Although we will account here for all cases where the matrix is not 2×2, we will only, initially, test matrices that are 3×3.

```
function D = Det_algo (A)
%make sure not to use a function name that may
%already be a matlab function, like det
if size(A,1)~=size(A,2)
  message='matrix not square; no proper determinant';
else
  if max(size(A))==2
    D = (A(1,1)*A(2,2))-(A(1,2)*A(2,1));
  else
    for i=1:size(A,1)
      D_temp=A;
      D_temp(1,:)=[ ];
      D_temp(:,i)=[ ];
      if i==1
        D=(A(1,i)*((-1)^(i+1))*Det_algo(D_temp));
      else
        D=D+(A(1,i)*((-1)^(i+1))*Det_algo(D_temp));
      end
    end
  end
end
```

Forming a recursive algorithm for a Determinant

- We then test the function for a 3×3 matrix.
- It is useful here, keeping in mind we are still debugging, to explicity output the result of sections of our calculation. Debugging tools are good, but they generally do not show you the numbers calculated at each iteration of the loops. Also, it is a good idea to section and large calculations into smaller chunks.
- To output the result of a calculation, simple remove the output suppression.

```
function D = Det_algo (A)
%make sure not to use a function name that may
%already be a matlab function, like det

if size(A,1)~=size(A,2)
  message='matrix not square; no proper determinant';
else
  if max(size(A))==2
    D = (A(1,1)*A(2,2))-(A(1,2)*A(2,1));
  else
    for i=1:size(A,1)
      D_temp=A;
      D_temp(1,:)=[ ];
      D_temp(:,i)=[ ];
      o=A(1,i)
      o=((-1)^(i+1))
      o=Det_algo(D_temp)
      o=(A(1,i)*((-1)^(i+1))*Det_algo(D_temp))
      if i==1
        D=(A(1,i)*((-1)^(i+1))*Det_algo(D_temp));
      else
        D=D+(A(1,i)*((-1)^(i+1))*Det_algo(D_temp));
      end
    end
  end
end
```

Forming a recursive algorithm for a Determinant

- If we test for a 3×3 case and all works well, we test for a 4×4 case.
- If all works, we may assume that the function will work for any size matrix, since there are no new types of operation for these increasing sizes, just added recursive elements. The 4×4 case was a good test for the recursive elements of the algorithm, so no more is needed..
- The next task would be to create a new function that uses the Det_algo function to find a matrix of cofactors. This would be useful in finding an inverse of a matrix.
- Another task may be to modify the Det_algo function so that it may be used to find eigenvalues of a matrix, where we are not just looking for the determinant as a number, but one that includes symbolic elements. Her a bit of leteral thinking is needed to find the coefficients of the characteristic polynomial produced when finding the eigenvalues. There are two main approaches to algorithm writing, top-down and bottom-up; the top-down approach may be useful here.

Outline

```
function C = Cofact_algo (A)

  for i=1:size(A,1)
   for j=1:size(A,2)
   C_temp=A;
   C_temp(:,j)= [ ];
   C_temp(i,:)= [ ];
   C(i,j)=((-1)^(i+j))*(Det_algo(C_temp));
   end
  end
```