Text reference page 195.

Determinants and Flops

Purpose
To investigate computational issues for determinants and productions $A * B$ and to compare the commands `det` and `cond` for determining the invertibilty of a matrix.

MATLAB Functions
`det, cond, cofactor`

This project assumes that the reader is already familiar with the material from the project on Floating Point Operations, referenced on page 146 of Section 2.5 in the text.

The definition of *determinant* is given on page 187 of the text:

$$\det A = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det A_{1j}$$

This is a recursive definition, in that it draws on the computation of the determinants for the smaller matrices A_{1j} obtained by removing the first row and the jth column. If A_{1j} is a 1×1 matrix, $\det A_{1j} = a_{11}$. This definition is particularly useful for understanding theoretical aspects of the determinant, but it is disastrous for computations.

The Laydata function `cofactor.m`, which is listed on the next page computes the determinant using this recursive definition. You can see how the program recursively calls itself with `cofactor(M)`, where `M` is the submatrix A_{1j} from the definition. The program includes a flops counter to illustrate how inefficient this particular means of computing the determinant is.

Try out the `cofactor` function on a matrix and compare the result with MATLAB's determinant function `det(A)`:

```
A = rand(5);
[dt,flops] = cofactor(A)
det(A)
```

The discussion on page 194 of the text shows that we can compute the determinant by row reduction. If A is invertible and it row reduces to U, then

$$\det A = (-1)^r (\text{product of the pivots in } U)$$

In MATLAB, the function `[L,U,P] = lu(A)` uses Gaussian elimination to compute a lower triangular matrix L, an upper triangular matrix U, and a permutation matrix (a product of row switching matrices) P, where $PA = LU$. Using the multiplicative property of the determinant we get

$$\det P \det A = \det L \det U$$

The determinant of P is 1 if the number of row switches to make P the identity matrix is even, and it is -1 if the number of row switches to make P the identity matrix is odd. Thus $\det P$ is easily computed. The matrix L is lower triangular with ones on the diagonal, so that $\det L$ is 1. It follows $\det A = \pm \det U$, where the sign is determined by $\det P$ and and $\det U$ is the product of the diagonal entries. Try this:

```
A = magic(5)
[L,U,P] = lu(A)
det(U)
det(P)
det(A)
prod(diag(U))
```

```
function [dt,flops] = cofactor(A)

% [dt,flops] = cofactor(A). Computes the determinant using the
% cofactor expansion.  This version always expands along the
% first row.  The variable flops will return the number of flops
% in the computation.  The Laydata version of this checks to make
% sure that A is square.

flops = 0;
n = size(A,1);
if n == 1,
    dt = A(1,1);
else
    sign = 1;
    dt = 0;
    for j = 1:n
        % Form the minor matrix by removing the
        % first row and the jth column of A.
        if j == 1
            M = A([2:n],[2:n]);
        elseif j == n
            M = A([2:n],[1:n - 1]);
        else
            M = A([2:n],[1:j - 1, j + 1:n]);
        end
        % Make the recursive call to the determinant
        % of the minor matrix.
        [dt1,newflops] = cofactor(M);
        % Build the expansion along the 1st row.
        dt = dt + sign*A(1,j)*dt1;
        % Alternate the sign.
        sign = -sign;
        % Update flops by counting the additions and
        % multiplications (but not the sign change).
        flops = flops + newflops + 2;
    end
end
```

I can't understand this step, and when I debug, it seems that it produces many steps that won't be needed in hand computing using the same method. Yet the answer is correct (after so many unexplainable steps !)

MATLAB Exercises

1. Use `cofactor` to compute the determinant of each matrix and report the number of flops.

a. `pascal(5)`
b. `magic(5)`
c. `rand(5)`
d. `eye(5)`

How many flops are performed to compute the determinant of a 5×5 matrix using `cofactor`? How many flops are performed to compute the determinant of a 6×6 matrix using `cofactor`?

2. The Laydata function `cofactorflops(n)` returns the number of flops needed when using `cofactor` to compute the determinant of an $n \times n$ matrix. The function `cofactorflops` is based on the recurrence relations

$\text{flops}(1) = 0$
$\text{flops}(n+1) = n(\text{flops}(n) + 2)$

Compare the values returned by `cofactorflops(n)` and `[dt,flops] = cofactor(rand(n))` for $n = 5,\ 6,\ 7,$ and 8. You should see a noticeable change in your waiting time as well as a dramatic increase in the flops count. What is the flops count when the determinant of a 10×10 matrix is computed using `cofactor`?

3. The function flops(n) computed by `cofactorflops` dominates the factorial function

$$n! = n(n-1)\cdots\ 3 \cdot 2 \cdot 1$$

That is, flops(n) $\geq n!$ for $n \geq 2$. The MATLAB function `factorial(n)` computes the factorial. Run this loop to compare the values of the two functions:

```
for n = 1:8 [cofactorflops(n), factorial(n)], end
```

To get an idea of the relationship between the two functions, try this:

```
for n = 1:8 cofactorflops(n)/factorial(n), end
```

What happens when you look at higher values of n? Describe a relationship between `cofactorflops(n)` and `factorial(n)` for large n. What happens to `cofactorflops(n)` and `factorial(n)` when $n = 12$ and $n = 13$?

4. For each of the following matrices compute the determinant using the MATLAB function `[L,U,P] = lu(A)`, and then check the result using `det(A)`.

a. `magic(5)`
b. `pascal(5)`
c. `rand(5)`
d. `magic(6)`

5. There is a significant computational advantage to using Gaussian elimination to compute the determinant rather than the cofactor expansion method. The function `lu(A)` requires approximately $2n^3/3$ flops for an $n \times n$ matrix A. Compare $n!$ and $2n^3/3$ for $n = 5,\ 6,\ 7,$ and 8. What happens to the ratio $n!/(2n^3/3)$ for large n?

6. You can get an idea of how flops are related to actual computing time by using the MATLAB functions `tic` and `toc`, which measure the elapsed time between the invoking of `tic` the invoking of `toc`. Thus

```
A = rand(1000); B = rand(1000);
tic, A*B; toc
```

gives the time it takes to multiply `A` and `B`. If you try this several times you will see that there is a lot of variation in the reported time. There are many variables that affect this time: processor speed, memory, number of open applications, and operating system, to name a few. This is one of the reasons for using a machine-independent measure like flops. Using `tic` and `toc`, we are going to make some plots of the elapsed time required to compute the determinant using MATLAB's `lu(A)` function.

```
x = 300:100:800
t = [ ];
for n = x,
    A = rand(n);
    tic;
    [L,U,P] = lu(A);
    prod(diag(U));
    t = [t,toc];
end
plot(x,t), hold
```

The vector `t` carries the time data. The initialization `t = [ ]` sets `t` to an empty vector. The `hold` command saves this graph so that we can plot another graph with it. If you get a bunch of zeros in `t`, reset `x` with `x = 3000:1000:8000` and recompute `t`. We have seen that the flops count for computing the determinant this way is approximately $2n^3/3$. So look at

```
y = 2/3 * x.^ 3
```

Most likely there is a noticeable difference between `y` and `t`, but they should be nearly proportional, that is `t = c*y` for some scalar `c`. We can solve for `c` with

```
c = t(1)/y(1)
y = y*c;
hold off, plot(x,y), hold, plot(x,t)
```

This solution should look pretty good. A better way to choose `c` would be to use a least-squares approximation. Print the graph with both plots, and label the plot of the curve $2n^3/3$.

7. Build the matrix `A` with

```
A = rand(5); B = 10 ^(-10)*A
```

Use MATLAB's `det` to compute the determinant. How close to 0 is `det(B)`? Based on this calculation, discuss the invertibility of `A`.

When you type `help det`, MATLAB delivers this message:

```
Use COND instead of DET to test for matrix singularity.
```

Read `help cond` to learn how to use `cond` to determine invertibility. Now try

```
cond(A)
```

Based on this calculation, discuss the invertibility of `A`.

8. Some matrices are mathematically invertible but behave computationally as if they are singular. The MATLAB function `cond(A)` is a way to measure "computational invertibility." Determine the mathematical invertibility of the following matrices, and compare with the value of `cond` for each one. Try using `inv` and testing `A*inv(A) - eye(n)` to see if MATLAB can compute an inverse. Which of these matrices is invertible, but has a large `cond` value, suggesting that it is difficult to compute the inverse?

a. `magic(7)`
b. `magic(8)`
c. `vander(1:6)`
d. `A = B'*B` where `B = vander(1:6)`
e. `ones(5)`
f. `ones(5)+10^(-10)*eye(5)`

9. Cramer's rule is a method for computing the solution to a system $A\mathbf{x} = \mathbf{b}$ when A is invertible (see page 201 of the text). The solution is given by $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$, where

$$x_i = \frac{\det A_i}{\det A}$$

and A_i is the matrix obtained by substituting $\mathbf{b}$ for the ith column of A. Try this in MATLAB to solve $A\mathbf{x} = \mathbf{b}$ where `A=magic(5)` and `b=ones(5,1)`. Find

```
A1 = A; A1(:,1) = b;
A2 = A; A1(:,2) = b;
A3 = A; A1(:,3) = b;
A4 = A; A1(:,4) = b;
A5 = A; A1(:,5) = b;
```

The solution is

```
x = [det(A1); det(A2); det(A3); det(A4); det(A5)]/det(A)
```

Compute the solution to $A\mathbf{x} = \mathbf{b}$ directly, and compare the result with the Cramer's rule solution.

Computing the determinant using Gaussian elimination uses approximately $2n^3/3$ flops for an $n \times n$ matrix. How many flops would it take to compute the solution to $A\mathbf{x} = \mathbf{b}$ for a 5×5 system using Cramer's rule and finding the determinant using Gaussian elimination? How many flops would it take to compute the solution to $A\mathbf{x} = \mathbf{b}$ for a 5×5 system using Cramer's Rule and computing the determinant with the function `cofactor`?