14. september, 2015

MAT-INF4130: Compulsory exercise 2

Deadline: 1/10-2015, kl. 14:30

1. In exercise 4.35, we saw an algorithm (the function `rothesstri`) for solving an upper Hessenberg linear system using Givens rotations. But we can clearly solve any linear system using Givens rotations also. Rewrite the algorithm so that it applies for any linear system (call the new function `x=rotsolve(A,b)`).
Answer: The following algorithm can be used:

```
function x=rotsolve(A,b)
  n=length(A);
  A=[A b];
  for k=1:n-1
      for i=k+1:n
         r=norm([A(k,k),A(i,k)]);
         c=A(k,k)/r; s=A(i,k)/r;
         A([k i],k+1:n+1)=[c s;-s c]*A([k i],k+1:n+1);
         A(k,k)=r; A(i,k)=0;
      end
  end
  x=rbacksolve(A(:,1:n),A(:,n+1),n);
```

This code starts with rotations which zero out the elements below the diagonal in column 1, then in column 2, and so on on. For column k, Givens rotations on the form $P_{k,k+1}$, $P_{k,k+2}$, $P_{k,k+1}$,...,$P_{k,n}$ are applied (and in that order) to columns $k,k+1,...,n+1$, where the Givens rotation $P_{i,j}$ are defined as the rotation matrix where only elements ii, ij, ji, and jj differ from those of the identity matrix.

This is not the only order in which you can perform Givens rotations in order to solve the system. Many of you have also started with the rotations corresponding to the highest column indices, then the two next highest column indices and so on. This also gives a correct answer. Some of you have performed rotations corresponding to all indices in a row first. This is also possible to do in a correct way, but care has to be taken in order not to introduce non-zeros at

places which already have been zeroed out. Some of you did not do this correct-
ly.

2. In exercise 4.35 we also computed the number of arithmetic operations needed by `rothesstri`. In the book (on the bottom of page 116) it was also stated without proof that `rotsolve` needs to perform $2n^3$ arithmetic operations. Prove this statement.
Hint: see Equation (4.14) for how you can set up an integral for computing the number of operations. You also did this in the first compulsory exercise.
Answer: A Givens rotation requires 6 arithmetic operations (4 multiplications and 2 additions). In the inner for loop we perform $n-k+1$ Givens rotations, so a total of $6(n-k+1)$ operations. Taking the for loop into account we perform

$$\sum_{k=1}^{n-1}(n-k)6(n-k+1) \approx \int_{k=1}^{n-1}(6(n-k)^2+6(n-k))dk = \int_{k=1}^{n-1}(6u^2+6u)dk \approx 2n^3.$$

3. Let A be the $n\times n$ Hilbert matrix (Matlab has the built-in command `hilb(n)` for producing this matrix. In Python you can produce the matrix with elements $1/(i+j+1)$ manually, see Exercise 0.35). Let $\boldsymbol{x_e} = (1,1,\dots,1)$, and let $\boldsymbol{b} = A\boldsymbol{x_e}$. Write code which uses the algorithm `rotsolve` to solve the linear system

$$A\boldsymbol{x} = \boldsymbol{b}.$$

Finally, for each n from 1 to 20, compute the error $\|\boldsymbol{x}-\boldsymbol{x_e}\|_2$ and plot these deviations. It turns out that the Hilbert matrix is extremely ill-conditioned, and this should be reflected in your plots: the error grows rapidly as n increases.
Answer: The following code can be used to plot the plot the deviation between the exact solution $\boldsymbol{x_e}$ to $A\boldsymbol{x}=\boldsymbol{b}$, and the solution found with `rotsolve`.

```
function oblig2(nmax)
  dev = zeros(nmax,1);
  for n=1:nmax
    xexact = ones(n, 1);
    b = hilb(n)*xexact;
    x = rotsolve(hilb(n),b);
    dev(n) = norm(x-xexact);
  end
  plot(1:nmax,dev);
```

Test this code from matrix sizes $n = 10$ and upwards to see how the error grows.