COLBY COLLEGE
Department of Mathematics
and Computer Science

MA332 **Allen B. Downey**
Fall 1998

Assignment 10: Fast Fourier Transform

Reading: Handouts from *Numerical Recipes* and *Physics of Music*.
Due: Thursday 3 December
Big picture: Using the DFT you wrote last week we're going to write a recursive implementation of the FFT.

Encapsulate the divide, conquer, and glue

1. Last time you wrote commands to split a signal into even and odd elements, calculate the DFT of the two halves, and then zip the demi-DFTs into the DFT of the whole. Take those commands and encapsulate them in a procedure named `fft`. The procedure should take the signal (and array of floating-point numbers) and the number of elements as arguments, just like `dft`. It should calculate and return an array of complex numbers.

2. Give some thought to proper encapsulation, which means that your procedure should not modify or read any global variables. It should only read its arguments, and all other variables should be declared local.

3. Use the `time` command to compare the run times of `dft` and `fft` for the same data (and make sure you get the same answer in both cases). (Read the documentation of `time` carefully!)

Make it recursive

The goal here is to make `fft` recursive by making it invoke itself instead of `dft`. In order to make that work, though, we have to avoid the possibility of infinite recursion by checking for a base case and terminating early (without making the recursive call) when we get there.

The base case for FFT is when the length of the array is 1, in which case the result is just the array itself.

1. Read the documentation for the `if` command in Maple. You have some choices regarding syntax, but I recommend something like the following:

```
fft:= proc (h, N)
  local x, y, z;
  if (N > 1) then

(put the complicated thing here)
```

```
    H;
  else
    h;
  fi;
end:
```

2. Recursive things can be hard to debug, so here are some suggestions:

   (a) Save your worksheet before you run the new routine. If you accidentally recurse infinitely, Maple might crash (although it shouldn't!).

   (b) Sneak up on it. If N=64, then try something like the following:

   ```
   fft:= proc (h, N)
     local x, y, z;
     if (N > 32) then

   (put the complicated thing here)

       H;
     else
       dft(h,N);
     fi;
   end:
   ```

   In other words, at the top level of the recursion (N=64), we do the divide, conquer and glue thing. But at the next level down we go ahead and use the `dft` procedure. Using this code, you can gradually increase the number of levels of recursion.

   (c) Alternatively, you can start with smaller values of N and gradually increase the number of levels from the other direction.

3. Time your program and compare it with the other timings. Try running FFTs and DFTs of different-sized signals (at least N=32, 64, and 128). Plot these points on a graph and comment on their trends.

4. While you're at it, make a note of how the shape of the FFT changes depending on N.

Optimize

This step is optional.

1. It is often the case with recursive things that we don't really want to recurse all the way down to the silly case where we take the DFT of a single element. Rather, once we get to something small, we go ahead and calculate it directly, using `dft`.

2. Modify your base case so that if $N > 2$ it recurses, but when $N = 2$ it calculates the DFT directly. Compare the run time to the original version. Experiment with different threshold values and find the optimum.