Project 2: The Discrete Fourier transform and convolution

1. **Symmetric Difference:** The symmetric difference approximation for a derivative is given by

$$f(x_n) \approx \frac{f(x_{n+1}) - f(x_{n-1})}{2dx}.$$

   As in Problem 2, construct the toeplitz matrix for this approximate derivative, and compute it in three ways: a) By matrix multiplication, b) By the discrete convolution theorem, and c) by the discrete correlation theorem. All three of these computations should be identical to machine accuracy, or approximately 10^{-15}.

2. **Formal or Exact Derivatives:** Write a self-contained program which computes the formal or exact derivative of a function, as defined in (3.4.2). Be careful to make sure that your multipliers ik match up correctly with the proper frequency coefficients. Test this on a few functions such as $\sin(kt)$ to assure accuracy. Once again, this should be exact to machine precision.

3. Use matlab to add noise to $\sin(kt)$ for some $k > 1$. (the command is >> fnoisy = sin(k*t)+ c*randn(size(t)); for some level of c which you can adjust, perhaps starting with c = .05). 1) Try take the derivative of this noisy function, using both of the above derivative algorithms above. Which works best? 2) Try to use Averaging, Narrowband Filtering, or Nonlinear Filtering to reduce the noise before taking the derivative. Which works best?

4. Take the derivative of the function

$$f = [zeros(1, 256), ones(1, 256), zeros(1, 128), 2 * ones(1, 256), zeros(1, 128)],$$

   with both the approximate and exact derivatives. Are these what you expected?

5. **Noise Reduction** Create a test function which has zeros on 1024 points, except for a simple sinusoid with 3 cycles which is 32 data points long in the middle, (i.e. $\sin([0:63]/64 * 2 * pi * 3)$. 1) Add noise at different levels to this test function, and use the matched filter to detect the sinusoid. At what noise levels does this seem possible? 2) Increase the length of the sinusoid to 64 data points with 6 cycles. Now use the matched filter with various noise levels as before. Is this function easier to find? Can you guess why?

6. **Cosine Transform** Write a self-contained program which will take a signal and compute its Cosine transform, utilizing the methods described above. Write a program decide how many significant coefficients are necessary for a given input tolerance (.01, or .001 for instance). Use (3.4.3) to determine the error. Make sure that the fft is only returning real coefficients, with the exception of small machine level errors.

7. **Sine Transform** Write a self-contained program which will take a signal and compute its Sine transform, utilizing the methods above. Once again, write another program to decide how many coefficients are necessary for a given input tolerance using (3.4.3) as the measure of error.

8. Compare the Sine and Cosine transforms above on a number of examples, such as t, and $t.^2$. First see if they give the same results as in the example above. Then test them on various other examples. Examine the final transformations to see how many coefficients are significant at different error levels.

9. Develop a fast algorithm for evaluating the FFT on 6 points, or decomposing F_6. Utilize some of the techniques used in the examples of F_4 and F_6 in the book. The straight forward calculation takes 36 operations. How many can you reduce this to?

10. **[Bonus Project:]** Using the iterative methods of Chapter 3, write an FFT code for 64 points.