

DESIGN WORK 3

Assigned 2 March 2006
Due 9 March 2006

Dynamics calculations using the time and frequency domains.

1. *Don't turn anything in on this part.* The FFT (Fast Fourier Transform) is a standard method for decomposition of a signal into its harmonic components. You need to know a few specific items about it:

- (a) The length of the input signal x - in the time domain - is N . The FFT itself doesn't care about what is the time step between x_n and x_{n+1} , although the spacing must be uniform; below, we'll call it Δt . Hence x_1 occurs at time 0, x_2 is at time Δt , and so on, up to x_N taken at time $(N-1)\Delta t$.
- (b) The formal definition of the transform and its inverse in Matlab is

$$X_k = \sum_{n=1}^N x_n e^{-i2\pi(k-1)(n-1)/N}, 1 \leq k \leq N;$$
$$x_n = \frac{1}{N} \sum_{k=1}^N X_k e^{i2\pi(k-1)(n-1)/N}, 1 \leq n \leq N,$$

where $i = \sqrt{-1}$. So the FFT takes a uniformly-spaced time signal x and makes a uniformly-spaced frequency signal X , of the same length. It is like a Fourier integral because of the exponential: $e^{iz} = \cos z + i \sin z$. Furthermore, you notice that the FFT (the transform that makes X) is a summation between the time-domain signal x and an exponential with imaginary argument, which has unity magnitude. Thus, the size of the elements in X are roughly the size of x , multiplied by N . The inverse FFT has the factor $1/N$ out front so as to cancel this effect out; `ifft(fft(x)) = x`; . Because of this arrangement, however, you can't take the FFT of two signals, multiply them (to effect a convolution), take the inverse FFT, and expect to get the right scaling - each FFT induces a scaling of N , but the IFFT only corrects for one of them. Remembering this point will save you a lot of trouble in using the FFT in Matlab!

You can see this scaling for example with

```
x = sin(0:0.1:100) ; X=fft(x);plot(abs(X));
```

 Here, you plot the absolute value (`abs()`) which is synonymous with magnitude, because X is complex.

There is another scaling factor to notice, but this is easier; because the Fourier integral involves both negative and positive frequencies, it puts half the area on each side.

- (c) What are the uniformly-spaced frequencies that go with X ? For the case of odd N (typical if the final time T is an even multiple of Δt), we have:

$$\omega_k = \left[0 \quad 1 \quad \cdots \quad \frac{N-1}{2} \quad \frac{1-N}{2} \quad \cdots \quad -1 \right] \frac{2}{N-1} \frac{\pi}{\Delta t}.$$

If you count these, you will see that there are N of them, exactly one frequency to go with each element of X . More interesting are the values, which start from zero and go to the Nyquist rate $\pi/\Delta t$, and then jump back to the negative of the Nyquist rate, and go to *almost*

zero. Together, these cover completely the range of negative Nyquist rate to positive Nyquist rate. The FFT can't see beyond the Nyquist rate, so we see that, in general, increasing N but keeping ΔT constant will give more resolution to a Fourier decomposition.

You can try making up the frequency vector that goes with the example above: Do the indices of the peak value in \mathbf{X} correspond with plus and minus one radian per second?

2. Now we use the FFT as a tool for analyzing system response to both specific and, later, to stochastic forcing signals.

First, find the impulse response of the system $x'' + 0.1x' + x = u$, using your simulation based on `ode45`. You'll have to construct the impulse explicitly in the derivative call, with something like this: `if t < epsilon, u0 = 1/epsilon ; else, u0 = 0 ; end;`. You can try values of $\epsilon < 0.1$ or so for this system. The response we are looking for is x , not x' , so you'll have to pick one column only out of the vector that `ode45` returns; call it \mathbf{h} . Use an initial time of zero in the simulation, and final time $T = 100$ seconds.

3. Regularize the impulse response you calculated by first making a uniformly spaced time vector like this: `treg = 0:dt:T`; a value for `dt` of 0.1 or so is OK. This `treg` is contrasted with the time vector returned by `ode45`, which is *not* uniformly spaced, because Matlab is automatically adjusting the time step as it does the integration for you. Use the spline function to make \mathbf{h} line up with the uniform time vector `treg`: `[hreg] = spline(t,h,treg) ;`.
4. Now, make up a new input u , that we will use in the rest of this problem: $u = 1$ if $t < T/2$, $u = 0$ if $t \geq T/2$. You need to put this in as an input in the derivative function for the simulation, and you need to also create it as a vector `ureg` that lines up with `treg`.

5. Now we are ready to make the computation of the system response to the input u in three different ways! The **first** is the simplest - just run the simulation with the input u . Make a plot of x vs. t .
6. The **second** method is to convolve the input u with the impulse response h . We have to use a consistent time base, and that is why we made `ureg` and `hreg`. Matlab has a command called `conv` which you can use; the convolution integral is the same as this vector convolution if you multiply the latter by `dt`; i.e., we have to calculate the area under the curve.

Plot the first N elements of the convolution result versus `treg`. It should look almost the same as what you found from the direct simulation.

7. The **third** method is to multiply the FFT of the impulse response with the FFT of the input u . In this case, you will do a vector pointwise multiplication (`.*` is the syntax) of `U = fft(ureg)` and the FFT of `h`. You could execute `fft(hreg)` of course, which builds off your original simulation. But the crucial observation here is that we already know the transform of the impulse response - it is the same as the transfer function!

$$H = \frac{1}{-\omega^2 + 0.1\sqrt{-1}\omega + 1}.$$

This is the calculation you will make, and you have to use the oddly-ordered frequency vector that is defined in (a) in order to make it work properly.

In summary for this step then, point-wise multiply the vector U times the vector H , then take the inverse FFT to get a third calculation for the response x when the system is driven by u . The IFFT result plotted versus `treg` should come out to be very close to the two results computed before. (Note you will get a little complaint from the plotting command, which doesn't know what to do with the very small imaginary parts of the IFFT result - ignore it.)

8. Why didn't we have to worry about the N -scaling of the FFT when we did the IFFT?