Overview

A time series is a sequence of measurements made over time, typically at equally spaced time intervals. For example:

- Temperature at Fairbanks International Airport measured every hour.
- Atmospheric pressure at the surface of a microphone measured 44100 times a second.
- Population size of a herd of caribou measured monthly.

After making N such measurements, we obtain an N-dimensional vector. Its first component is the first measurement, and its last component is the last measurement. In this lab we will examine a basis for $\mathbb{R}^N$ that is useful for analyzing time series data.

Before starting this lab, you will need the data file `lab3.mat` which can be downloaded from the course web pages. The first cell in the lab notebook loads this file and creates three variables from it, `data16`, `data256` and `data_sound`. You should execute this cell first before preceding with the lab.

Exercise 1: The variable `data16` contains 16 measurements. To visualize it, we don't think of it as a geometric vector in $\mathbb{R}^{16}$. Instead, we plot the measurements as a function of time (or of the sample index). Do this in the notebook.

Exercise 2: We can express `data16` in terms of the standard basis for $\mathbb{R}^{16}$:

$$\texttt{data16} = x_1\mathbf{e}_1 + \cdots x_n\mathbf{e}_n.$$

What is the specific value of x_5 in this case?

Exercise 3: Many time-series measurements can be thought of as a sum of oscillations at different frequencies. For example, a caribou population can be expected to oscillate over a period of one year due to annual season changes, but might also undergo oscillations over a fifty-year time period due to longer-term effects.

In this lab we will look at a basis for $\mathbb{R}^N$ that allows us to pick data apart into pieces oscillating at different frequencies. To begin, we need a nice way to represent the times corresponding to the measured data samples. We will scale time so that the measurements occur over the time interval $[0,1]$. If there are N samples, we will break this interval up into N subintervals. And we will assume that each sample occurs at the **middle** of its corresponding time interval.

Under these hypotheses, write down the sample times for N measurements in the following cases: $N = 1$, $N = 2$, $N = 3$, and $N = 4$. In the $N = 4$ case, also draw a schematic diagram of the interval $[0,1]$ broken into time intervals, with each of the sample times marked with an asterisk. Attach these hand computations along with your diagram at the end of your lab report PDF submission.

Exercise 4: Write down a formula for sample time t_k assuming that there are N samples, starting at $k = 1$ and ending at $k = N$. You can enter the formula into a Markdown cell in the notebook, or as part of your hand computations.

Exercise 5: Write a function `sample_times(N)` that returns a vector of length N of sample times corresponding to the formula you developed. I've started the function for you, and you just need to fill in a single line of it. Test that your function works by computing `sample_times(4)` and verifying that it gives the correct times.

Exercise 6: Make a variable `t16 = sample_times(16)`. Then make a plot `Plots.plot(t16, data16)`. What is different about this plot and the one you made in Exercise 1?

Exercise 7: Plot each of the following vectors:

- `g1 = cos.( 0 * pi * t16 )`
- `g2 = cos.( 1 * pi * t16 )`
- `g3 = cos.( 2 * pi * t16 )`
- `g4 = cos.( 3 * pi * t16 )`
- `g5 = cos.( 4 * pi * t16 )`

Plot all the series in a single plot, and ensure that there is a legend with appropriate labels of `g1` through `g5`.

Exercise 8: In general, let gk `= cos.((`$k-1$`) * pi * t16)`. What is the period of gk, assuming time is measured in seconds? What is the frequency?

Exercise 9: Plot `g17`. Then **explain** what you see. A full answer does not just describe the graph, but also gives an explanation for why it is what it is. Be sure you look at the scale on the y-axis. It might be helpful to compare the graph of `g17` to some of the earlier gk's. Keep in mind that the diameter of the nucleus of a gold atom is about 10^{-14} meters.

Exercise 10: We will now show that the 16 vectors `g1` through `g16` form a basis for $\mathbb{R}^{16}$. The first step is to make a matrix `G` such that column k of `G` is gk. Here's a slick way to do this using the outer product. Let

$$\texttt{T} = \texttt{t16} * (\texttt{0:15})' * \texttt{pi}.$$

This is the product of the column vector `t16` with the row `(0:15)'` to make a 16×16 matrix, and then scaled by π.

How is column k of `T` related to `t16`? Write down a one-line Julia command that builds the matrix `G` out of the matrix `T`. Verify for yourself that you have the right matrix by comparing the plots of `G(:,1)`, `G(:,2)`, `G(:,3)`, and `G(:,4)` with those of `g1`, `g2`, `g3`, `g4`.

Exercise 11: Since `G` is a 16×16 matrix, its columns form a basis for $\mathbb{R}^{16}$ if its columns are linearly independent. We can check for linear independence by performing QR factorization and verifying that the diagonal elements of R are all nonzero. Because we are working with floating point arithmetic, we'll need to check that the entries are all "far from" zero instead.

Follow the instructions in the notebook to compute the QR factorization and then extract the diagonal elements of R.

Exercise 12: Since the gk's form a basis for $\mathbb{R}^{16}$, every vector **b** in $\mathbb{R}^{16}$ can be written uniquely in the form

$$\mathbf{b} = x_1\ \mathtt{g1} + \cdots + x_{16}\ \mathtt{g16}$$

for certain scalars $x_1, \ldots, x_{16}$. To get a feeling for what a representation of this type means, plot for yourself `3*g1 + 1/2*g4 - 1/8*g10`. You can do this using

```
b = 3 * G[:,1] + (1/2) * G[:,4] - (1/8) * G[:,10]
Plots.plot(t16,b)
```

What part of the resulting graph does the term `3*g0` contribute? What about `1/2*g3`? What about `-(1/8)*g10`? You might find it easiest to answer these questions by making (for yourself) graphs that exclude one or more of the three terms.

Exercise 13: Suppose we want to find scalars x_i such that

$$\mathtt{data16} = x_1\ \mathtt{g0} + \cdots + x_{16}\ \mathtt{g15}.$$

Write this problem down as a matrix problem involving `G`.

Exercise 14: To solve this problem we would normally use a method like QR factorization. But in this case, `G` has a very nice inverse. Compute `G'*G` and then describe all of the entries. You should see that most entries are essentially zero. The notebook shows you a technique for replacing numbers in a matrix that are close to zero with numbers that are exactly zero. You might want to do that first before doing your analysis.

Be careful: one entry is different from all the others. This exercise shows that $\mathtt{G}^{-1}$ is nearly the same as $\mathtt{G}^T$.

Exercise 15: What is the value of the dot product gi · gj? Your answer should depend on i and j. Give an explicit formula. Hint: this has something to do with the previous Exercise.

Exercise 16: Find a diagonal matrix `S` such that `F = G * S` satisfies $\mathtt{F}^{-1} = \mathtt{F}^T$. Using this matrix, create the matrix `F` and verify that `F'*F` is essentially the identity. How is each column of `F` `F` related to the corresponding column of `G`? Why are we multiplying `G` on the right and not on the left?

Exercise 17: The columns of `F` are known as the Fourier basis for $\mathbb{R}^{16}$. They are better-scaled vectors from the original basis `g1` through `g16` you were working with before. I'll use the notation `f1` through `f16` to denote this new basis.

Given a vector `d` in $\mathbb{R}^{16}$ we can write

$$\mathtt{d} = x_1\ \mathtt{f1} + \cdots + x_{16}\ \mathtt{f16}$$

for unique numbers x_1 through x_{16}. To compute these numbers, we would need to solve

$$\mathtt{F} * \mathtt{x} = \mathtt{d}.$$

But since $\mathtt{F}^{-1} = \mathtt{F}^T$, the solution is simply

$$\mathtt{x} = \mathtt{F}' * \mathtt{d}.$$

The vector `x` is called the Fourier transform of `d`.

Plot the Fourier transform of `data16`.

By the time you get to the end of the lab, you'll have a good intuition about what this plot means.

Exercise 18: If I give you the Fourier transform `x`, how can you reconstruct `d` using `F`? Do so.

Exercise 19: Let `x = F' * data16` be the Fourier transform of `data16`. We want to examine what happens as we build up `data16` from `x` by adding more and more terms. That is, we want to consider the vectors

$$\begin{aligned} \texttt{y1} &= \texttt{x}[1] * \texttt{F}[:, 1] \\ \texttt{y2} &= \texttt{x}[1] * \texttt{F}[:, 1] + \texttt{x}[2] * \texttt{F}[:, 2] \\ \texttt{y3} &= \texttt{x}[1] * \texttt{F}[:, 1] + \texttt{x}[2] * \texttt{F}[:, 2] + \texttt{x}[3] * \texttt{F}[:, 3] \end{aligned}$$

and so forth.

Create the variables `y1`, `y3`, and `y5` according to the formulas above. The visualize them (for yourself) as follows:

$$\texttt{plot}(\texttt{t16}, [\texttt{data16}, \texttt{y1}, \texttt{y3}, \texttt{y5}])$$

This plots the original data vector and then the three approximations obtained by adding up the first view terms. Explain what you see in your plots as you add more terms of the Fourier basis vectors to your graphs.

Exercise 20: Enter the following Julia commands

```
z = copy(x)
z[9:16]. = 0
```

How is `z` different from `x`? Then enter the following Julia commands

```
w = copy(x)
w[1:8]. = 0
```

How is `w` different from `x`? What is `w + z`? What is `F*w+F*z`?

Exercise 21: Now execute `Plots.plot(t16, [data16,F*z,F*w],label=["data16" "F*z" "F*w"])`. What effect does leaving out early terms from the Fourier basis have? What effect does leaving out later terms have?

Exercise 22: Enter the following Julia commands to create a new vector `x` and the corresponding time series `d`. Note that we are starting from a Fourier transform and going back

to create a time series.

```
x = zeros(16)
x[2] = 1
x[15] = -.5
d = F * x
```

Plot the Fourier transform `x` and separately plot the data vector `d`. Notice that data vector is a sum of a low frequency component and a high frequency component. How can you tell this by looking at the graph of the **Fourier transform** `x`?

Exercise 23: The data vector `data256` contains 256 measurement samples. Plot it and observe that it is a low frequency signal that has been contaminated with noise.

Exercise 24: Construct the 256×256 matrix `F256` corresponding to the 16×16 matrix `F` we have been working with. You should give commands to make

- `T256` corresponding to `T`
- `G256` corresponding to `G`
- `S256` corresponding to `S`
- `F256` corresponding to `F`

If this has all worked correctly, then `F256'*F256` will be the identity matrix, up to round off error.

Exercise 25: As noted earlier, The measurement samples in `data256` have been contaminated by noise. We can use the Fourier transform to implement a **low-pass** filter, which allows low frequency components to pass through but removes high frequency components. This will eliminate the high frequency noise in the signal. Use `F256` and its inverse to remove much of the noise from the data. Plot of the original data and your version with the noise removed.

Extra Credit Past Here

Exercise 26: The vector `data_sound` contains 8820 samples corresponding to 1/5 of a second of me trying to sing a note.

Plot `data_sound`.

Also, for your amusement, follow the instructions in the notebook to play this very short snippet of sound. It will sound a bit weird because it doesn't last long, but you should be able to detect that there is a note in there.

Exercise 27: We would like to analyze it using the Fourier basis, but the methods we have been using are not efficient enough. For example, the matrix `F8820` would needlessly take

up over half a gigabyte of memory. Computing

$$\texttt{F8820}' * \texttt{data\_sound}$$

would take roughly $8820^2 \approx 10^8$ multiplications. This is far better than the roughly $8820^3 \approx 10^{11}$ multiplications required by first doing a QR factorization, but is still a bit large.

I need to confess at this stage I've been telling you a white lie. The transform we are using here is called the **discrete cosine transform**. The Fourier transform is a similar, but more complex object. In practice, discrete cosine transforms are computed using a fantastic algorithm called the Fast Fourier Transform, which requires only $O(N\log(N))$ multiplications. Note that $8820\log(8820) \approx 3.5 \cdot 10^4$, which is dirt cheap!

You can compute discrete cosine transformations in Julia by installing the FFTW package and using the `FFTW.dct` command. You can convert transform frequency data back into time-series data using the command `idct` (inverse discrete cosine transform).

Do the following (assuming you still have the matrix `F` you computed earlier still in memory):

```
x16_dct = dct(data16)
x16 = F' * data16
Plots.plot(t16, x16 - x16_dct)
```

and verify that the vectors `x16_dct` and `x16` are the same. The algorithm in `dct` is exactly what we have been doing (inefficiently).

Exercise 28: Compute the discrete cosine transform `x_sound` of `data_sound`. Plot `x_sound`, and then describe the features of the plot.

Exercise 29: What is the sample rate of the audio data in data_sound? Your answer should be in Hz.

Exercise 30: What is the period in seconds and the frequency in Hz of the function `f4` in this case? It might be helpful to go look at how you found your answer to Exercise 8. But keep in mind that for that problem we assumed time was scaled so that all the data was sampled between $t = 0$ and $t = 1$. You'll need to adjust this scaling to get a meaningful answer because your signal is only 1/5 of a second long.

Exercise 31: What are all of the dominant frequencies in the audio sample? That is, what are the frequencies associated with the spikes? What note was I trying to sing? Provide justification for your answers using the tools developed in this lab. Keep in mind your answer to Exercise 30.

Exercise 32: Thinking of the audio sample, notice that the discrete cosine transform of the full signal is nearly zero for most of the frequencies. If we needed to transmit (or store) this signal, we don't really need all 8820 components of `data_sound`. We could transmit (or store) just 1000 numbers instead. What 1000 numbers should we store? How would we reconstruct the signal based on those 1000 numbers? Do this! Then make a plot of the difference between reconstructed and original signal.

This principle underlies many techniques of "lossy compression". High frequencies are omitted with the expectation that they carry little data. The JPEG image standard uses a two dimensional discrete cosine transform applied to little 8x8 tiles in the picture. The MP3 audio format uses a variation of the discrete cosine transform as well.