18.330 Problem set 9 (spring 2020)

Submission deadline: 11:59pm on Tuesday, April 28

Exercise 1: Fourier series

In lectures we saw that we can write a periodic function f in the form

$$f(x) = \sum_{n=-N}^{N} \hat{f}_n e^{inx}$$

where $\hat{f}_n := \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-inx} \mathrm{d}x$ if the period is 2π.

The FFTW library uses a different convention, taking the period to be 1. Suppose we sample a function at $N+1$ uniformly-spaced points between $[0,1]$, obtaining values $f_0, f_1, \ldots, f_N$, where $f_n := f(nh)$ with $h := \frac{1}{N}$. By using the Fourier series we are implicitly assuming that f is periodic, i.e. $f_0 = f_N$. Then

$$f(x) = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \hat{f}_n e^{2\pi inx}$$

where the Fourier coefficients are

$$\hat{f}_n := \int_0^1 f(x)\, e^{-2\pi inx} \mathrm{d}x$$

This is the convention we will use in this problem.

1. Discretize the integral for the Fourier series coefficients using the trapezoidal rule using $N+1$ points. Using the assumption that $f_0 = f_N$ reduce this to a sum over f_i for $i = 0, \ldots, N-1$.

   Implement this as a function `fourier_coefficients_trapezoidal(f::Vector)` where `f` is the vector of samples of f_n for $n = 0, \ldots, N-1$. Your function should return the coefficients for $\hat{f}_k$ for $k = -\frac{N}{2}, \ldots, \frac{N}{2} - 1$ as a vector.

2. Consider the following three functions:

   1. The sawtooth function

      $$f(x) = \mathrm{mod}(x, 1)$$

      (The function $\mathrm{mod}(x,1)$ returns just the fractional part of a number.)

2. The "W" wave function

$$g(x) = \begin{cases} x & 0 \leq x < 0.5 \\ 1 - x & 0.5 \leq x < 1 \\ g(\text{mod}(x, 1)) & \text{otherwise} \end{cases}$$

3. The function

$$h(x) = \exp(\cos(2\pi x))$$

Plot them.

Given the properties of these functions, how would you expect their Fourier coefficients to decay?

3. Take $N = 100$. Calculate $\hat{f}, \hat{g}$ and $\hat{h}$ using your `fourier_ceofficients_trapezoidal` function. Plot the magnitude of the coefficients as a function of n only for $n \geq 0$. Do you see the expected behavior?

4. How accurately does our function `fourier_coefficients_trapezoidal(f::Vector)` calculate the Fourier coefficients? Use the following analytic solutions to calculate and plot the respective errors:

$$\hat{f}_0 = \frac{1}{2} \quad \text{and} \quad \hat{f}_n = \frac{i}{2\pi n} \text{ for } n \neq 0$$

$$\hat{g}_0 = \frac{1}{4}; \quad \hat{g}_n = -\frac{1}{\pi^2 n^2} \text{ for } n \text{ odd}; \quad \text{and} \quad \hat{g}_n = 0 \text{ for } n \text{ even}$$

$$\hat{h}_n = I_1(n)$$

where $I_1(n)$ is the modified Bessel function of the first kind, calculated in Julia as `besseli(n, 1)` using the `SpecialFunctions.jl` package.

Calculate the error (using `norm`) as the number of points used changes between $N = 10$ and $N = 1000$.

5. Write a function `reconstruct_fourier_series(f̂s::Vector, xs::Vector)` which reconstructs `f(x)` from the Fourier series coefficients.

Make a plot of `||f(xt) - reconstruct_fourier_series(f̂s, xt)||/N` as the number of coefficients used N is varied from $4 \to 200$ for each of the functions.

Sample the reconstructed Fourier series at the points `xt = 0:0.001:1`. What do you see? Can you explain why?

6. Make an interactive visualiztion that plots the following on the *same* axes:

   (i) the points used to calculate the Fourier coefficients in `fourier_coefficients_rectangle`;

   (ii) the reconstructed function from the Fourier coefficients, found using `reconstruct_fourier_series`; and

   (iii) the true function as the number of points is varied from $N = 10 : 2 : 250$.

   Does this help explain the results in [1.6]? In particular, what do you see for the sawtooth function? This is known as the **Gibbs phenomenon**, which occurs when a function is discontinuous.

7. What is the operation count for your naive `fourier_coefficients_trapezoidal` function? In general this will not behave well as N grows very large.

   The FFT, however, can calculate this in $O(\log(N))$ steps. The FFT implemented in `FFTW.jl` calculates

   $$\hat{f}_n = \sum_{k=0}^{N-1} f_k e^{-2\pi i n k/N}$$

   for $n = 0 : N-1$, which should be related to your discretization in [1.1].

   Note that the ns are different here. But since $e^{-2\pi i(n+lN)k/N} = e^{-2\pi i l k} e^{-2\pi i n k/N} = e^{-2\pi i n k/N}$ we have the relationship $\hat{f}_{N/2+i} = \hat{f}_{-N/2+i}$.

   The FFT algorithm therefore outputs

   $$\hat{\mathbf{f}} = [\hat{f}_0, \hat{f}_1, \dots, \hat{f}_{N/2-1}, \hat{f}_{-N/2}, \hat{f}_{-N/2+1}, \dots, \hat{f}_{-1}]$$

   The FFTWpackage defines a function `fftshift` that shifts this vector to the form

   $$\hat{\mathbf{f}} = [\hat{f}_{-N/2}, \hat{f}_{-N/2+1}, \dots, \hat{f}_{-1}, \hat{f}_0, \hat{f}_1, \dots, \hat{f}_{N/2-1}]$$

8. Implement a function `fast_fourier_coefficients` that outputs the same results as `fourier_coefficients_trapezoidal` but using the FFT from `FFTW.jl`.

   Check that the output is the similar to before.

   Time your two functions for $N = 2^10$. Is one faster than the other?

   How large can you take `N` such that `fast_fourier_coefficients` runs for under 1 second? What about for `fourier_coefficients_trapezoidal`?

Exercise 2: Solving an ODE with a spectral method

In this problem we will solve the boundary-value problem

$$f'' = g$$

with boundary conditions $f(0) = f(1) = 0$ by using a **spectral method**, i.e. by expanding in suitable basis functions.

Due to the chosen boundary conditions we will consider the sine series

$$f(x) = \sum_{n=1}^{\infty} b_n \sin(n\pi x),$$

which is the same as the Fourier series for an odd function.

In practice we have to truncate the summation as

$$f(x) = \sum_{n=1}^{N} b_n \sin(n\pi x)$$

The coefficients are given by

$$b_n = 2 \int_0^1 f(x) \sin(n\pi x) \mathrm{d}x$$

Similarly to the DFT we discretize this using the rectangle rule to get the Discrete Sine Transform (DST):

$$b_n = \frac{2}{N} \sum_{k=1}^{N-1} f(x) \sin\left(\frac{nk\pi}{N}\right)$$

This sum is implemented in julia using the `FFTW.jl` library (which you will need to install) as follows:

```
dst(x) = FFTW.r2r(x, FFTW.RODFT00) / length(x)
```

Its inverse is given by

```
idst(x) = FFTW.r2r(x, FFTW.RODFT00)
```

[`r2r` stands for "real to real", meaning that the transform maps a real vector to a real vector. `RODFT00` is a symbol that selects one particular type of transform.]

1. Assume that there is a solution of the form $\tilde{f}(x) = \sum_{n=1}^{N} b_n \sin(n\pi x)$. Substitute this into the ODE $f''(x) = g(x)$ to show that $\hat{\tilde{f}}_n = -\frac{\hat{g}_n}{\pi^2 n^2}$.

2. We can therefore solve the ODE for f by first calculating $\hat{g}$ using the DST, then calculate $\hat{f}_n$, and finally invert using the iDST.

   Write a function `spectral_solver(b)` that does this to solve the ODE, where `b` is the discretized version of `g(x)`. Solve the ODE with $g(x) = \sin(2\pi x)$. Plot the result.

   The right-hand side is given by

   ```
   h = 1 / N
   b = sin.(2π * (h:h:1-h))
   ```

3. Calculate the error as a function of N and plot it. What rate of convergence do you see?

4. Generate the error plots again, now for the right-hand side $g(x) = \exp(\sin(2\pi x)) - 1$. Use the solution from your spectral solver with $N = 2^{13}$ as the true solution. Calculate the error for $N = 2^n$ for $n = 1 \to 12$. Some care is needed to make sure you use the correct points from the "true" solution for comparison.

Exercise 3: Finding roots in a different way

In class we defined the Chebyshev expansion of a function as

$$f_N(x) = \sum_{n=0}^{N} c_n T_n(x)$$

which is an Nth-degree polynomial. The Chebyshev polynomials are defined as $T_n(x) = \cos(n \arccos(x))$.

In general for a smooth function the Chebyshev series converges rapidly. We therefore expect that the roots of $f_N(x)$ should be close to the roots of $f(x)$, provided that f_N is indeed a good approximation to f. We have already seen that we can find all the roots of a polynomial with various methods.

The Chebyshev polynomials satisfy the following recurrence relation:

$$T_{n+1}(x) = 2x\,T_n(x) - T_{n-1}(x)$$

We will use this to find the **companion matrix** for $f_N(x)$, from which we can find the roots of $f_N(x)$.

1. Consider the polynomial $xT_n(x)$. This is a degree $n+1$ polynomial and hence can be reexpanded in Chebyshev polynomials.

Consider the vector of Chebyshev polynomials

$$\mathbf{T}(x) = \begin{bmatrix} T_0(x) \\ T_1(x) \\ \vdots \\ T_{N-1}(x) \end{bmatrix}$$

Now we can write $x\mathbf{T}(x) = M\mathbf{T}(x) + k_N \hat{e}_N T_N(x)$, where M is an $(N-1)\times(N-1)$ matrix. We need the k_N term to account for the fact that xT_{N-1} is a degree-N polynomial. Here, $\hat{e}_N$ is the standard basis vector with zeros everywhere except in the Nth component.

Use the recurrence relation to find the form of M and k_N.

2. Verify what you found in [3.1] numerically when $N = 10$. Build the vector $\mathbf{T}(x)$ for x a random number in $[-1, 1]$. Compute M and check that it gives $x\mathbf{T}(x)$.

3. This looks almost like an eigenvalue problem, except for the T_N term. To remove this we can add and subtract $Cf_N(x)\hat{e}_N$ from the right-hand side. Writing $f_N(x) = c_N T_N(x) + \sum_{n=0}^{N-1} c_n T_n(x)$, what value of C should you choose so that

$$x\mathbf{T}(x) = M\mathbf{T}(x) + Cf_N(x)\hat{e}_N - C\hat{e}_N \sum_{n=0}^{N-1} c_n T_n(x) \qquad (1)$$

$$= \tilde{M}\mathbf{T}(x) + Cf_N(x)\hat{e}_N \qquad (2)$$

What is the new matrix $\tilde{M}$?

4. This becomes an eigenvalue problem when x is a root of $f_N(x)$. Therefore, the eigenvalues of $\tilde{M}$ are the roots of f_N.

Write a function `buildM(c::Vector)` that constructs the matrix $\tilde{M}$ from the coefficients in the Chebyshev expansion. Use this to write a `chebyshev_roots(c)` function that finds the roots of the polynomial defined using the Chebyshev coefficients `c`. Finally write a function `fN(x, c)` that calculates the series expansion to find $f_N(x)$ defined by the vector `c`.

5. We can calculate Chebyshev coefficients using the `dct` functions in FFTW. We will use the Chebyshev points $x_n := \cos(n\pi/N)$ for $n = 0 : N$. You can then calculate the Chebyshev coefficients using the following code:

```
chebyshev_points(N) = cos.(π*(0:1:N)/N)

function chebyshev_coefficients(x)
    N = length(x)
    c = FFTW.r2r(x, FFTW.REDFT00)/(N-1)
    c[1] /= 2
    c[N] /=2

    return c
end
```

6. Consider the polynomial $f(x) = x\,(x-1/2)^2\,(x^2-1/9)\,(x+1/4)$. Using 10 Chebyshev points calculate the Chebyshev coefficients and then calculate the roots using `chebyshev_roots`. What do you see. What about multiplicities?
7. Plot $f(x)$ and scatter plot the roots you find on top.
8. Now consider solving the problem $\exp(\cos(4\pi x)) = 1$. Using $N = 100$ points, alculate the Chebyshev coefficients for $g(x) = \exp(\cos(4\pi x)) - 1$. Do they decay quickly? Use these to calculate the roots of $g(x)$. Plot $g(x)$ and `scatter` the roots you find on top. Do you find all the roots?

   (Hint: you will find 100 eigenvalues. Only plot those that are real and lie between -1 and 1.)
9. Make an interactive visualization as N is varied between $N = 4:150$. Plot $g(x)$, the Cheyshev approximation to $g(x)$ using N coefficients and the roots you find on the same axes. Comment on what you see.

   At what value of N do you find all the roots? Remember to plot only those roots that are real and between -1 and 1.

Exercise 4: Gram–Schmidt for polynomials

In lectures we discussed treating the set of polynomials $\{1, x, x^2, x^3, ...\}$ as the basis of a vector space with the inner product

$$(f, g)_w = \int_a^b f(x)g(x)w(x)\mathrm{d}x$$

We can therefore carry out Gram–Schmidt orthogonalization on these polynomials to generate a family of **orthogonal polynomials**.

We will implemnent this using the `Polynomials.jl` package. Integrals can be performed using the `polyint(f, a, b)` function to integrate the polynomial f

from a to b. Here, f, g and w should all be `Polynomials`.

1. Write a function `gram_schmidt(vs::Vector, ip)` which accepts a vector of "vectors" in the vector space and the inner product on the vector spacce. For standard vectors this would be `dot(v1, v2)`. The function should implement the Gram–Schmidt algorithm and return a vector of the resulting orthonormal basis elements.

2. Test your function for standard vectors `vs = [rand(10) for i = 1:10]` using `ip = dot`. To check everything went according to plan, form the matrix $Dp_{ij} = q_i \cdot q_j$. If everything worked this should be the identity matrix.

3. For polynomials we define the inner product

$$(f, g)_w := \int_a^b f(x)g(x)w(x)\mathrm{d}x$$

   For Legendre polynomials we have $a = -1$, $b = 1$, and $w(x) = 1$. Using the `Polynomials.jl` package, implement a function `legendre_inner_product(f, g)`. Use this to orthogonalize the vector of monomials up to order 7.

4. Use the functions you found in [4.3] and your inner-product function, find the Legendre polynomial expansion coefficients for the function $f(x) = x\,(x - 1/2)^2\,(x^2 - 1/9)\,(x + 1/4)$.

   Plot the reconstructed polynomial using the first n coefficients for $N = 1:7$ and the true function. What do you see? How good are the Legendre polynomials at approximating the function?