Homework 2

** ***Note: To avoid confusing myself, I'm going to relabel the equation*** $\phi(t, \theta, \phi) \rightarrow f(t, \theta, \phi)$

> **a)** Numerically implement the discretiation of f in terms of spherical harmonics.

In order to discretize f in terms of spherical harmonics, we want to calculate the following:

$$f(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} c^{lm} Y_{lm}(\theta, \phi)$$

where the coefficients c^{lm} are found by calculating

$$c^{lm} = \int_0^{2\pi} \mathrm{d}\phi \int_0^{\pi} \mathrm{d}\theta f(\theta, \phi) Y^*_{lm}(\theta, \phi) \sin\theta$$

In [ ]:
```
using SphericalHarmonics
using HCubature
using DifferentialEquations
using Plots
```

We can obtain a vector of all c^{lm} coefficients using the `HCubature` and `SphericalHarmonics` packages:

In [ ]:
```
# Convert SHArray object to vector
Ylm_array(sharray) = [ y for y in sharray ]

# Integrate over θ and ϕ to get c_lm coefficients
c_vector(f, lmax) = hcubature(x -> integrand(f, lmax, x), [0, 0], [π, 2*π])[1]

# Calculate the integrand to be passed into hcubature()
function integrand(f, lmax, x)
    # x = (θ, ϕ)

    # Use SphericalHarmonics pacakge to calculate Y_lm(θ, ϕ)^*
    sharray = computeYlm(x[1], x[2], lmax=lmax)
    ylm_conj = conj(Ylm_array(sharray))

    # f(θ, ϕ) * conj(Y_lm(θ, ϕ)) * sin(θ)
    return f(x[1], x[2]) * ylm_conj * sin(x[1])
end

# Use c_lm coefficients to calculate f(θ, ϕ)
function expand_f(c_vector, θ, ϕ, lmax)
    # Ylm(θ, ϕ)
    sharray = computeYlm(θ, ϕ, lmax=lmax)
    ylm = Ylm_array(sharray)

    # f(θ, ϕ) = Σ_lm( c_lm * Y_lm(θ, ϕ) )
    return round(sum(c_vector .* ylm), digits=4)
end
```

```
expand_f (generic function with 1 method)
```

> **b)** Use an initial condition that is peaked around the North Pole, i.e., that looks similar to a Gaussian with a width equal to 0.2. (The exact initial condition does not matter).

To apply the initial condition peaked at the North Pole, we state that at time $t = 0$ we have a Gaussian that peaks for all values of ϕ when $\theta = 0$ (additionally, we can set $\psi_0 = 0$):

$$f(t=0,\theta,\phi) \equiv f_0(\theta) = e^{-\frac{\theta^2}{2*0.2^2}} = e^{-\frac{\theta^2}{0.08}}$$

$$\psi_0 = 0$$

To find the corresponding coefficient vectors at $t=0$:

```
In [ ]: # f_0(θ) = exp(-θ^2/0.08)
        c_f0_init(lmax) = c_vector((θ, ϕ) -> exp(-θ^2/0.08), lmax)

        # ψ_0 = 0
        c_ψ0_init(lmax) = c_vector((θ, ϕ) -> 0, lmax)
```

```
c_ψ0_init (generic function with 1 method)
```

> **c)** Evolve the system in time to see from $t=0$ to $t=10$ using your favorite ODE integrator. The resulting evolution should look similar to water waves moving on the surface of a pond, except that the pond is the surface of a sphere.

We will now make use of the following to set up our system:

$$\partial_t f = \psi \quad \rightarrow \quad \partial_t c_f^{lm} = c_\psi^{lm}$$

$$\partial_t \psi = \Delta f \quad \rightarrow \quad \partial_t c_\psi^{lm} = \Delta c_f^{lm} = -l(l+1) c_f^{lm}$$

With this in mind, our set of ODEs for all c_f^{lm}, c_ψ^{lm} will use the following function to record each set of values over time. `Udot()` takes in the flattened array of (c_f^{lm}, c_ψ^{lm}) and returns the corresponding $(\dot{c}_f^{lm}, \dot{c}_\psi^{lm})$.

```
In [ ]: function Udot(U, lmax)
            # U = vcat(c_f, c_ψ)

            # Separate our individual coefficient vectors
            n = round(Int, length(U)/2)
            c_f = U[1:n]
            c_ψ = U[n+1:end]

            # We already have ∂_t(c_f) = c_ψ
            c_fdot = c_ψ

            # Build a new vector containing coefficients ∂_t(c_ψ) = Δc_f = -l(l+1)c_f
            c_ψdot = Vector{ComplexF64}([])
            i = 1
            for l in 0:lmax
                for m in -l:l
                    push!(c_ψdot, -l*(l+1)*c_f[i])
                    i+=1
                end
            end

            # Recombine the two vectors
            return vcat(c_fdot, c_ψdot)
        end
```

```
Udot (generic function with 1 method)
```

Then, using the `DifferentialEquations` package to find our solution for a given l_{max}:

```
In [ ]: function solve_ode(lmax, tmax)
            # Generate initial coefficients
            c_f0 = c_f0_init(lmax)
            c_ψ0 = c_ψ0_init(lmax)

            c_f0 = [ round(x, digits=5) for x in c_f0 ]
            c_ψ0 = [ round(x, digits=5) for x in c_ψ0 ]
```

```
    U0 = vcat(c_f0, c_ψ0)

    # Pass our initial values and Udot() into the package's ODEProblem() function and solve
    prob = ODEProblem((U, p, t) -> Udot(U, lmax), U0, (0.0, tmax))
    sol = solve(prob)

    # Round output data to make our lives easier
    ts = [ round(t, digits=5) for t in sol.t ]
    us = [ [ round(c, digits=5) for c in u ] for u in sol.u ]

    return ts, us
end
```

```
solve_ode (generic function with 1 method)
```

Putting everything together, let's look at an example case with $l_{max} = 2$ from $t = 0$ to $t = 5$:

In [ ]:
```
lmax = 2
tmax = 5

ts, us = solve_ode(lmax, tmax)

# Display a fragment of sample data
for (i, t) in enumerate(ts)
    println(t, "   ", us[i][1:2:end])
end
```

```
0.0   ComplexF64[0.06996 + 0.0im, 0.11645 + 0.0im, 0.0 + 0.0im, 0.13886 + 0.0im, 0.0 - 0.0im, 0.0 +
0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
0.00198   ComplexF64[0.06996 + 0.0im, 0.11645 + 0.0im, 0.0 + 0.0im, 0.13886 + 0.0im, 0.0 + 0.0im, 0
.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
0.02181   ComplexF64[0.06996 + 0.0im, 0.11639 + 0.0im, 0.0 + 0.0im, 0.13866 + 0.0im, 0.0 + 0.0im, 0
.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
0.13589   ComplexF64[0.06996 + 0.0im, 0.11431 + 0.0im, 0.0 + 0.0im, 0.13124 + 0.0im, 0.0 + 0.0im, 0
.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
0.36591   ComplexF64[0.06996 + 0.0im, 0.1012 + 0.0im, 0.0 + 0.0im, 0.08672 + 0.0im, 0.0 + 0.0im, 0.
0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
0.67426   ComplexF64[0.06996 + 0.0im, 0.0674 + 0.0im, 0.0 + 0.0im, -0.01121 + 0.0im, 0.0 + 0.0im, 0
.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
0.99982   ComplexF64[0.06996 + 0.0im, 0.01819 + 0.0im, 0.0 + 0.0im, -0.10687 + 0.0im, 0.0 + 0.0im,
0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
1.3867   ComplexF64[0.06996 + 0.0im, -0.04431 + 0.0im, 0.0 + 0.0im, -0.13437 + 0.0im, 0.0 + 0.0im,
0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
1.78586   ComplexF64[0.06996 + 0.0im, -0.09505 + 0.0im, 0.0 + 0.0im, -0.04604 + 0.0im, 0.0 + 0.0im,
0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
2.24186   ComplexF64[0.06996 + 0.0im, -0.1164 + 0.0im, 0.0 + 0.0im, 0.09756 + 0.0im, 0.0 + 0.0im, 0
.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
2.69563   ComplexF64[0.06996 + 0.0im, -0.09123 + 0.0im, 0.0 + 0.0im, 0.13182 + 0.0im, 0.0 + 0.0im,
0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
3.18412   ComplexF64[0.06996 + 0.0im, -0.0242 + 0.0im, 0.0 + 0.0im, 0.00757 + 0.0im, 0.0 + 0.0im, 0
.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
3.69501   ComplexF64[0.06996 + 0.0im, 0.05717 + 0.0im, 0.0 + 0.0im, -0.12927 + 0.0im, 0.0 + 0.0im,
0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
4.21802   ComplexF64[0.06996 + 0.0im, 0.11061 + 0.0im, 0.0 + 0.0im, -0.08553 + 0.0im, 0.0 + 0.0im,
0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
4.74544   ComplexF64[0.06996 + 0.0im, 0.10595 + 0.0im, 0.0 + 0.0im, 0.08162 + 0.0im, 0.0 + 0.0im, 0
.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
5.0   ComplexF64[0.06996 + 0.0im, 0.08214 + 0.0im, 0.0 + 0.0im, 0.13186 + 0.0im, 0.0 + 0.0im, 0.0 +
0.0im, 0.0 + 0.0im, 0.0 + 0.0im, 0.0 + 0.0im]
```

> **d)** Create a series of figures or a movie that shows how the solution f evolves in time. Perform the simulation three times with different choices of l_{max}, and at least one of these with a small l_{max} (e.g., $l_{max} = 4$) to study the influence of the cut-off l_{max}.

Using the output of our ODE solver, we want to generate a list of all (f, θ, ϕ) values for each time t:

In [ ]:
```
function gather_data(ts, us, lmax)
```

```
    # Isolate c_lm coefficients in each U matrix
    n = round(Int, length(us[1])/2)
    all_c_lms = [ u_t[1:n] for u_t in us ]

    # Initialize plot ranges
    n_vars = 20
    θs = [ round(θ, digits=5) for θ in range(0, π, n_vars) ]
    ϕs = [ round(ϕ, digits=5) for ϕ in range(0, 2*π, 2*n_vars) ]

    # Create lists of θ and ϕ values to plot (i.e. with repetitions included)
        # ex: [θ_1, θ_1, θ_1, θ_2, θ_2, θ_2] ; [ϕ_1, ϕ_2, ϕ_3, ϕ_1, ϕ_2, ϕ_3]
    θs_toplot = vcat([ repeat([θ], 2*n_vars) for θ in θs ]...)
    ϕs_toplot = repeat(ϕs, n_vars)
    θ_ϕ_pairs = collect(zip(θs_toplot, ϕs_toplot))

    # Generate matrix of f(t, θ, ϕ) values with structure M[(θ, ϕ), time]
    f_vals_t(c_lms) = [ expand_f(c_lms, θ, ϕ, lmax) for (θ, ϕ) in θ_ϕ_pairs ]
    all_f_vals = hcat([ f_vals_t(c_lms_t) for c_lms_t in all_c_lms ]...)

    return θ_ϕ_pairs, all_f_vals
end
```

```
gather_data (generic function with 1 method)
```

Finally, converting to cartesian coordinates, we can plot our sphere with its radius fluctuating as $f(t, \theta, \phi) + 1$, where our unit sphere at $f = 0$ is ensured with an offset of $+1$.

In [ ]:
```
function data_all_times(ts, us, lmax)
    θ_ϕ_pairs, all_f_vals = gather_data(ts, us, lmax)

    # Base coordinates for unit sphere
    x0 = [ round(cos(ϕ)*sin(θ), digits=3) for (θ, ϕ) in θ_ϕ_pairs ]
    y0 = [ round(sin(ϕ)*sin(θ), digits=3) for (θ, ϕ) in θ_ϕ_pairs ]
    z0 = [ round(cos(p[1]), digits=3) for p in θ_ϕ_pairs ]

    return all_f_vals, x0, y0, z0
end

function data_at_time(f_vals_t, x0, y0, z0)
    # r = 1 + f(t, θ, ϕ)
    x = real((f_vals_t .+ 1) .* x0)
    y = real((f_vals_t .+ 1) .* y0)
    z = real((f_vals_t .+ 1) .* z0)

    return x, y, z
end
```

```
data_at_time (generic function with 1 method)
```

Now let's create a plot for each time step and turn them into an animation!

In [ ]:
```
lmax = 4
tmax = 10

ts, us = solve_ode(lmax, tmax)
all_f_vals, x0, y0, z0 = data_all_times(ts, us, lmax);
```

In [ ]:
```
anim = Animation()
fps = 15

for (i, t) in enumerate(ts)

    f_vals_t = real(all_f_vals[:,i])
    x, y, z = data_at_time(f_vals_t, x0, y0, z0)

    plot(x, y, z, title="lmax = $(lmax)\nfps = $(fps)\nt = $(t)", legend=false,
        xlim=(-1.2, 1.2), ylim=(-1.2, 1.2), zlim=(-1.2, 1.2))
    frame(anim)
end
```

```
gif(anim, fps=fps)
```

(see external files for gifs with $l_{max} = 4,\ 10,$ and 20)