In [1]:
```
] add SphericalFunctions
```
```
   Resolving package versions...
  No Changes to `C:\Users\thoma\.julia\environments\v1.10\Project.toml`
  No Changes to `C:\Users\thoma\.julia\environments\v1.10\Manifest.toml`
```

In [2]:
```
] add HCubature
```
```
   Resolving package versions...
  No Changes to `C:\Users\thoma\.julia\environments\v1.10\Project.toml`
  No Changes to `C:\Users\thoma\.julia\environments\v1.10\Manifest.toml`
```

In [3]:
```
] add Printf
```
```
   Resolving package versions...
  No Changes to `C:\Users\thoma\.julia\environments\v1.10\Project.toml`
  No Changes to `C:\Users\thoma\.julia\environments\v1.10\Manifest.toml`
```

In [4]:
```
] add Plots
```
```
   Resolving package versions...
  No Changes to `C:\Users\thoma\.julia\environments\v1.10\Project.toml`
  No Changes to `C:\Users\thoma\.julia\environments\v1.10\Manifest.toml`
```

In [5]:
```
using SphericalFunctions
using HCubature
using Printf
using Plots
```

Having prepared the packages that we will be using, we are ready to start the first part of the problem.

a) Numerically implement the discretization of ϕ in terms of spherical harmonics.

We will have a function that takes as arguments a function of the polar angles θ, φ and l_{max} that returns the vector of coefficients c_{lm} with l_{max}^2 corresponding to the expansion in spherical harmonics. Let's start by defining a useful function that will perform the polar integrals

In [6]:
```
integrate(f,xi,xf)=hcubature(x -> sin(x[1])*f(x),xi,xf;atol=sqrt(eps()))[1]
```

Out[6]:
```
integrate (generic function with 1 method)
```

Now we can define the expansion in spherical coordinates

In [7]:
```
expand_in_spherical(f,lmax)=[integrate(x -> f(x)*conj(sYlm_values(x[1],x[2],lmax,0)[i]
        sYlm_values(x[1],x[2],lmax,0)[i]*conj(sYlm_values(x[1],x[2],lmax,0)[i]),[0.0,0
```

Out[7]:
```
expand_in_spherical (generic function with 1 method)
```

Let's test the function with a simple case (I won't be printing numbers that are below 1e-10 and change them to 0 to make the result prettier):

In [8]:
```
f(x)=sin(x[1])
clm=expand_in_spherical(f,6)

threshold = 1e-10

i=1
for l in 0:6
    for m in -l:l
    real_part = real(clm[i])
    imag_part = imag(clm[i])
    if abs(real_part) > threshold || abs(imag_part) > threshold
        real_part_str = abs(real_part) > threshold ? @sprintf("%.2g", real_part) : "0'
        imag_part_str = abs(imag_part) > threshold ? @sprintf("%.2g", imag_part) : "0'
        if imag_part < 0
            println("$(l),$(m): $real_part_str - $(abs(imag_part_str))*im")
        else
            println("$(l),$(m): $real_part_str + $imag_part_str * im")
        end
    end
    i+=1
    end
end
```

```
0,0: 2.8 + 0 * im
2,0: -0.78 + 0 * im
4,0: -0.13 + 0 * im
6,0: -0.049 + 0 * im
```

This expansion is correct. Thus, we have numerically implemented the discretization of a function in terms of the spherical harmonics.

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).

We will first write a function that implements a 3D Gaussian restricted on a spherical surface of unit radius parameterised by the polar angles θ and φ centered around the North Pole with width 0.2:

In [9]:
```
function gauss(r)
    (theta,phi)=r
    x0=0
    y0=0
    z0=1
    return exp(-((sin(theta)cos(phi)-x0)^2+(sin(theta)sin(phi)-y0)^2+(cos(theta)-z0)^2
end
```

Out[9]:
```
gauss (generic function with 1 method)
```

The initialization function will decompose this gaussian in spherical harmonics. We will give input l_{max} and the output will be a matrix, which will have in its first column the coefficient vector for the Gaussian (initial position) and in the second column it will have the coefficients for the initial velocity, which we will take to be zero:

In [10]:
```
function init(lmax)
    clm=expand_in_spherical(gauss,lmax)
    clmpsi=[0.0+0.0*im]
    for i in 2:(lmax+1)^2
        push!(clmpsi,0.0+0.0*im)
    end
    return [clm,clmpsi]
end
```

Out[10]:
```
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 can obtain as done in previous cases a system of two first order equations instead of one second order equation for the wave equation $\partial_t^2\phi = \Delta\phi$:

$$\partial_t\phi = \psi \ , \ \partial_t\psi = \Delta\phi$$

Using the expansion into spherical harmonics along with the knowledge that they constitute a basis we arive at the following equations for the coefficients (unit sphere):

$$\partial_t c^{\phi}_{lm} = c^{\psi}_{lm} \ , \ \partial_t c^{\psi}_{lm} = -l(l+1)c^{\phi}_{lm}$$

We'll make a function that describes this system and takes as input a matrix of the c coefficients U and l_{max}:

In [11]:
```
function wave(U,l_max)
    c_phi=U[1]
    c_psi=U[2]
    cdot_phi = c_psi
    cdot_psi=[0.0+0.0*im]
    for i in 2:(l_max+1)^2
        push!(cdot_psi,0.0+0.0*im)
    end
    i=1
    for l in 0:l_max
        for m in 1:(2*l+1)
            cdot_psi[i] = -l*(l+1)c_phi[i]
            i+=1
        end
    end
    Udot = [cdot_phi,cdot_psi]
    return Udot
end
```

Out[11]:
```
wave (generic function with 1 method)
```

For simplicity let's use an Euler integrator (remember to pass l_max):

In [12]:
```
function Euler(f,y,h,l_max)
    k=f(y,l_max)
    y = y.+h.*k
    return y
end
```

Out[12]:
```
Euler (generic function with 1 method)
```

We can now write a code that integrates the problem for given l_{max} and initial condition U_0. Using a time step of 0.01 we see that to get from $t = 0$ to $t = 10$ we need 10000 steps:

In [13]:
```
function integrator(lmax,U0)
    U=U0
    Uall=[U]
    h=0.01
    for i in 1:1000
        U=Euler(wave,U,h,lmax)
        push!(Uall,U)
    end
    return Uall
end
```

Out[13]:
```
integrator (generic function with 1 method)
```

Then, the task is done just by calling the functions (the code takes a long time to compile, so we'll do $l_{max} = 4$ for the small case, $l_{max} = 6$ as a medium case and $l_{max} = 8$ as a large case):

In [14]:
```
U0_1=init(4);
U0_2=init(6);
U0_3=init(8);
```

In [15]:
```
Uall_1=integrator(4,U0_1);
Uall_2=integrator(6,U0_2);
Uall_3=integrator(8,U0_3);
```

We will see if the evolution is the expected one once we do the animations.

d) Create a series of figures or a movie that shows how the solution ϕ 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}.

We have in $Uall_i$ the evolution from 1 to 10 seconds for an increasing number of maximum spherical harmonics mode. All we have to do is to make an animation to show this. To do that, we can make a surface plot where the radius denotes the value of the field at a given direction. Let's write a function that takes as inputs the evolution of the expansion parameters along with the maximum spherical harmonics mode and returns an animation showing the evolution. First, we need a function that makes a sphere parametric of the radius.

In [37]:
```
function sphere(f,c,lmax)
x = zeros(100, 100)
y = zeros(100, 100)
z = zeros(100, 100)

phi_values = range(-pi, pi; length = 100)
theta_values = range(0, pi; length = 100)

for (i, theta) in enumerate(theta_values)
    for (j, phi) in enumerate(phi_values)
```

```
        x[i, j] = (1 + f(float(theta), float(phi), c, lmax)) * cos(phi) * sin(theta)
        y[i, j] = (1 + f(float(theta), float(phi), c, lmax)) * sin(phi) * sin(theta)
        z[i, j] = (1 + f(float(theta), float(phi), c, lmax)) * cos(theta)
    end
end
    return x,y,z
end
```

Out[37]:
```
sphere (generic function with 1 method)
```

We will then need a function that takes as inputs the angles and returns the value of our field given the spherical harmonic coefficients and the maximum l.

In [40]:
```
function reconstruct(theta, phi, c, lmax)
    varphi=0
    Y=sYlm_values(theta,phi,lmax,0)
    for i in 1:(lmax+1)^2
        varphi+=Y[i]*c[i]
    end
    return real(varphi)
end
```

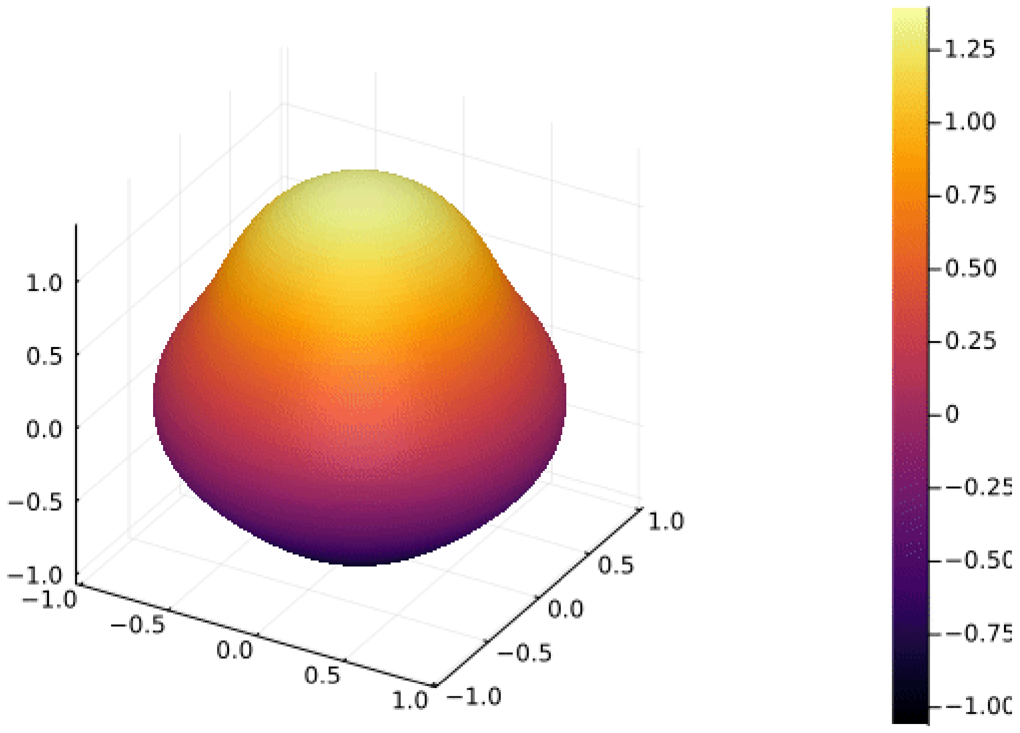
Out[40]:
```
reconstruct (generic function with 1 method)
```

Putting everything together

In [66]:
```
anim = @animate for i in 1:1000
    surface(sphere(reconstruct,Uall_1[i][1],4))
end
gif(anim, "Spherical_Harmonics_4.gif", fps = 15)
```

```
[ Info: Saved animation to C:\Users\thoma\Spherical_Harmonics_4.gif
```

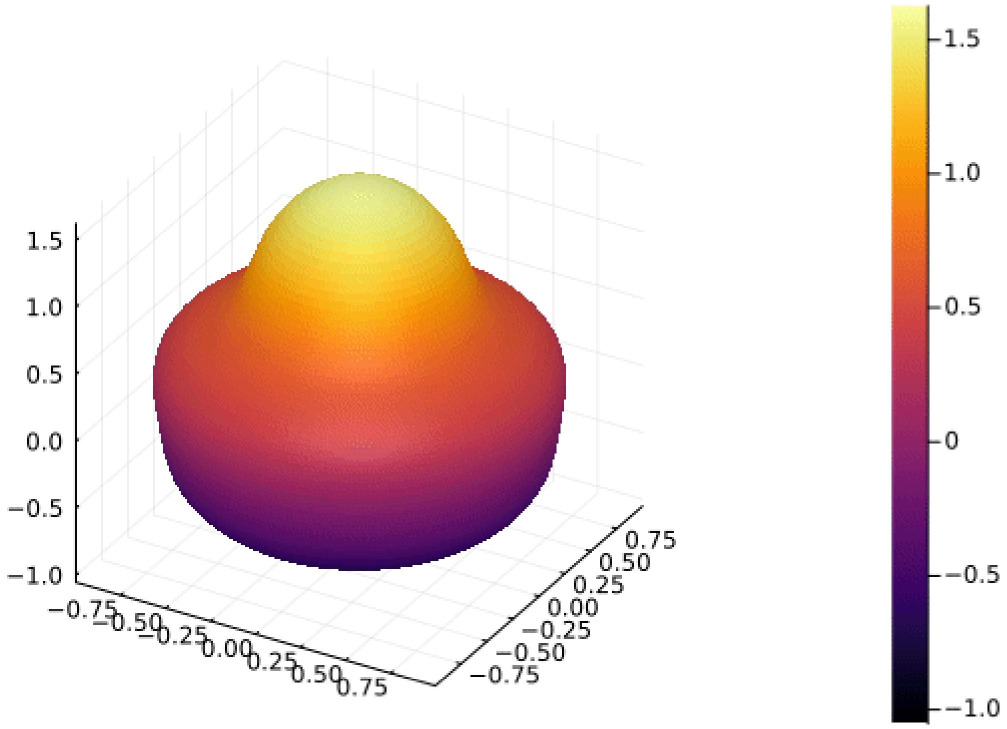
Out[66]:



In [67]:
```
anim = @animate for i in 1:1000
    surface(sphere(reconstruct,Uall_2[i][1],6))
end
gif(anim, "Spherical_Harmonics_6.gif", fps = 15)
```

```
[ Info: Saved animation to C:\Users\thoma\Spherical_Harmonics_6.gif
```

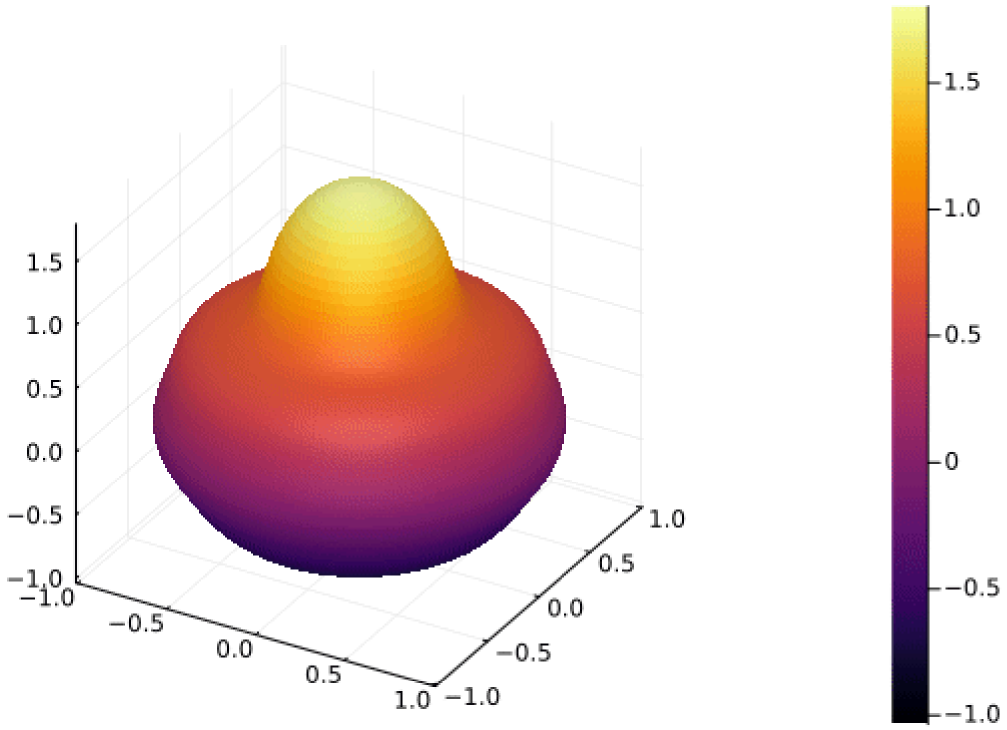
Out[67]:



In [68]:
```
anim = @animate for i in 1:1000
    surface(sphere(reconstruct,Uall_3[i][1],8))
end
gif(anim, "Spherical_Harmonics_8.gif", fps = 15)
```

```
[ Info: Saved animation to C:\Users\thoma\Spherical_Harmonics_8.gif
```

Out[68]:



Acknowledgement: We had many useful discussions with my fellow PSIon Marko Brnovic about how to handle different parts of the Homework. My fellow PSIon PA Graham, gave me very useful tips regarding how to do an animation of a surface plote. Finally, ChatGPT helped me present nicely the example of the spherical harmonics (2 significant digits).

```
In [ ]:
```