Chapter 2: Introduction to Slepian Functions

Sarah Kroeker, Alain Plattner

July 12, 2018

After having worked through Chapter 1: Introduction to Spherical Harmonics, we will now have a look at what spherical Slepian functions are. The goal of this chapter is not to give a solid foundation of the mathematical intricacies. This can be obtained from the many research articles in the literature. Rather, we will look at some basic ideas and plot the first few Slepian functions.

1   Basic ideas

When plotting spherical harmonics in chapter 1, it became clear that all of these spherical harmonics are functions that cover the entire sphere. This is of course an advantage if we try to describe functions (or maps), that cover the entire planet, such as gravity fields, magnetic fields, etc. We saw in chapter 1 that we can generate pretty much any spatial pattern we like by simply summing up different spherical harmonics each multiplied by a different factor called "coefficient".

But what can we do if we only have information within a specific region? Sometimes it's fine to just describe that information as point values. But in some cases, as for example when we measure gravity or magnetic data at satellite altitude, and we want to calculate, what the corresponding magnetic or gravity field looks like on the planet's surface (assuming there are no gravity or magnetic sources between the planet and the satellite, or that they are negligible), we need to describe these fields in spherical harmonics.

This is where the Slepian functions come in. Slepian functions are linear combinations of spherical harmonics which means that they are constructed by multiplying each spherical harmonic function with a factor (called Slepian coefficient) and then adding them all up. The name Slepian function stems from the first author of a research article, that described the original idea for 1-dimensional functions.

There are different types of Slepian functions that are constructed in different ways but the basic idea is always to solve an optimization problem, in which we try to balance the number of spherical-harmonic functions we use (the maximum spherical-harmonic degree) in the construction of the Slepian functions, and how much they are concentrated within the region that we are interested in.

2   Classical Scalar Slepian Functions

Let's first look at the classical scalar Slepian functions, which are spectrally limited and spatially optimized. These functions are suitable to use when you are limited to information within a specific region as mentioned above, but only when that information is constrained to the surface of the sphere as point values; classical vector Slepian functions will be addressed in a later chapter after a discussion of vector spherical harmonics. The Slepian functions are bandlimited by the maximum spherical harmonic degree L for region of concentration R. We will choose our region to be the continent **Africa** for our first example. The following examples will include a region of an axisymmetric polar cap, a polar ring, and both polar cap and ring again when rotated away from the north pole.

2.1   Calculating Coefficients for a Region

In the Slepian folder start MATLAB and run:

```
demos_chapter_two(2.1)
```

You may also find it helpful to open `demos_chapter_two.m` in a text editor and browse the script as you read through this tutorial. The demo file is divded into cases for each subsection of this chapter.

The function `glmalpha` is used to calculate unit-normalized spherical harmonic coefficients given the following required inputs:

  `TH` - a desired region (in this example: `'africa'`)

  `L` - the bandwidth, or maximum spherical harmonic degree.

If you find the program running too slowly for the purpose of quickly going through this tutorial or freezing up in MATLAB you may need to edit the demo file and set `L` in `glmalpha()` to a smaller value, say 5 for example.

Thus the output of `glmalpha` we used for this example is:

  `G` - a unitary matrix of localization coefficients.

There are several other inputs and outputs that may and should be used later; see `help glmalpha` for more details.

We specify one other input to indicate the ordering of the spherical harmonic coefficients. In our codes we use two different methods of ordering: ADDMOUT and ADDMON.

Remember that spherical harmonics have a degree l and an order m. Degrees are all integers greater than zero and orders for each degree vary between $-l$ and l.

We could order these spherical harmonics as

l	0	1	1	1	2	2	2	2	2	etc.
m	0	-1	0	1	-2	-1	0	1	2	etc.

This is called the ADDMOUT ordering and is the ordering in which the columns of the Slepian matrices in `glmalpha`, `gradvecglmalpha`, and all these other similar functions are returned.

The other ordering that we may use is:

l	0	1	1	1	2	2	2	2	2	etc.
m	0	0	-1	1	0	-1	1	-2	2	etc.

This one we call ADDMON and it is the ordering in which the coefficients from functions like `LocalInnerField` (which will be discussed in a later chapter) are returned. Every function that returns coefficients or matrices of coefficients should state in their help menu in which format the coefficients are. Just run `help [function]` in MATLAB.

Returning to our example, if you run `help glmalpha` you will read that it uses the ADDMOUT ordering and thus requires us to use `0` as the fourth input. We will choose a value of `L=20`. Now we can run:

```
[G] = glmalpha('africa',20,[],0)
```

and observe the output `G` which should be a $(L+1)^2 \times (L+1)^2$ orthonormal matrix of coefficients. Each column of `G` contains a set of coefficients for a single slepian function which are in the ADDMOUT ordering:

$$
G = \begin{bmatrix}
g_{00,1} & g_{00,2} & \cdots & g_{00,(L+1)^2} \\
g_{1-1,1} & g_{1-1,2} & \cdots & g_{1-1,(L+1)^2} \\
\vdots & \vdots & & \vdots \\
g_{LL,1} & g_{LL,2} & \cdots & g_{LL,(L+1)2}
\end{bmatrix}
$$

where the subscript of each element contains the lm values and the slepian function number separated by a comma respectively. The first column `G(:,1)` is the coefficients corresponding to the "best" Slepian funtion, or the one that is most spatially concentrated within our chosen region of Africa. `G(:,2)` would be the second best, `G(:,3)` would be the third best, and so on until `G(:,(L+1)^2)`.

Once the coefficients have been calculated we desire a neutral type of sorting in the format called LMCOSI to prepare for plotting.

In the lmcosi format, we explicitly write out the degrees, orders, and coefficients which in the following I will call $c_{l\,m}$

l	m	co	si
0	0	$c_{0\,0}$	0
1	0	$c_{1\,0}$	0
1	1	$c_{1\,-1}$	$c_{1\,1}$
2	0	$c_{2\,0}$	0
2	1	$c_{2\,-1}$	$c_{2\,1}$
2	2	$c_{2\,-2}$	$c_{2\,2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

As you can see this last ordering is an entire matrix instead of just a vector. This is a disadvantage to run calculations but it is less ambiguous than just a vector of coefficients.

Luckily we have programs that take care of these issues. These programs have the creative names `coef2lmcosi` and `lmcosi2coef`. You can probably guess from the name what each does.

For our Africa example we will need to feed our coefficients into `coef2lmcosi` and again tell it whether they are in the ADDMOUT or ADDMON format. This is indicated by using the value 1 for the second input (see `help coef2lmcosi`). The output is saved as *lmcs* but you can name it whatever you would like. Run:

```
lmcs=coef2lmcosi(G(:,1),1)
```

The first column of G is used for now in our example for simplicity's sake.

We now have one more step before we are ready to plot and that is to evaluate the coefficients using the function `plm2xyz`. The required inputs are our *lmcosi* format matrix and a chosen resolution. You can choose *res*=1, or 0.5, or even 0.1 for a very high resolution image, but keep in mind this will also use much calculation time and memory. We will give the output the arbitrary name 'data' and choose a resolution of 0.5. Run:

```
data=plm2xyz(lmcs,0.5)
```

And finally to plot we will use `plotplm`:

```
plotplm(data,[],[],1,0.5)
```

where the fourth input chooses the plotting method. The value '1' corresponds to a Mollweide projection of Earth.

We use `kelicol(1)` for a more clear color scheme where white sections are of value 0. It is recommended to change the maximum and minimum of the color scheme which you can do by running:

```
caxis([-1,1]*max(abs(caxis)));
```

As expected, the fluctuations are concentrated on the center of Africa and negligible elsewhere. The plot produced by `demos_chapter_two(2.1)` should match Figure 1.
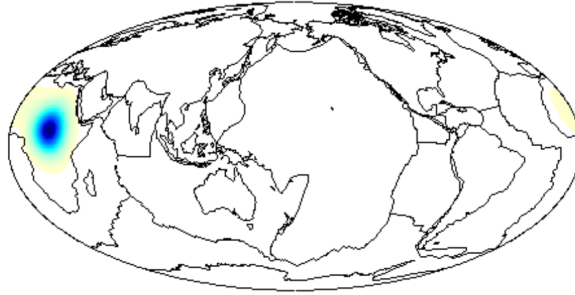

Figure 1: Result of example using glmalpha() with L=20 to calculate coefficients, plm2xyz() to evaluate them, and plotplm() to plot with method 1 - the Mollweide projection. Only the first "best" concentrated Slepian function is plotted. The color scheme is kelicol(1) and max/min values are changed to fit the max/min of data plotted.

Exercise: Pick a different a region, or subtract a region from Africa to use as the domain and plot again. You can edit `demos_chapter_two.m`, write your own `.m` file, or run line by line in matlab.

2.2   Calculating Coefficients for an Axisymmetric Polar Cap

We now look at the case of an axisymmetric polar cap.

You can run:

```
demos_chapter_two(2.2)
```

which will plot both a Mollweide projection, as well as the perspective of looking down at the North Pole.

The functions used for this type of region follow the same format as in section 2.1, however we have changed the inputs of `glmalpha()`. TH is now an opening angle in degrees, which we have chosen to be 40, and a value of 1 is passed as the third input to indicate we are creating coefficients for a polar cap of radius `TH`.

To plot as a sphere, a value of 2 for the plotting method should be given to `plotplm()` and then running `view(2)` will rotate the sphere so that the North Pole is pointing out at the viewer.

The demo should produce the same images as Figure 2 below.
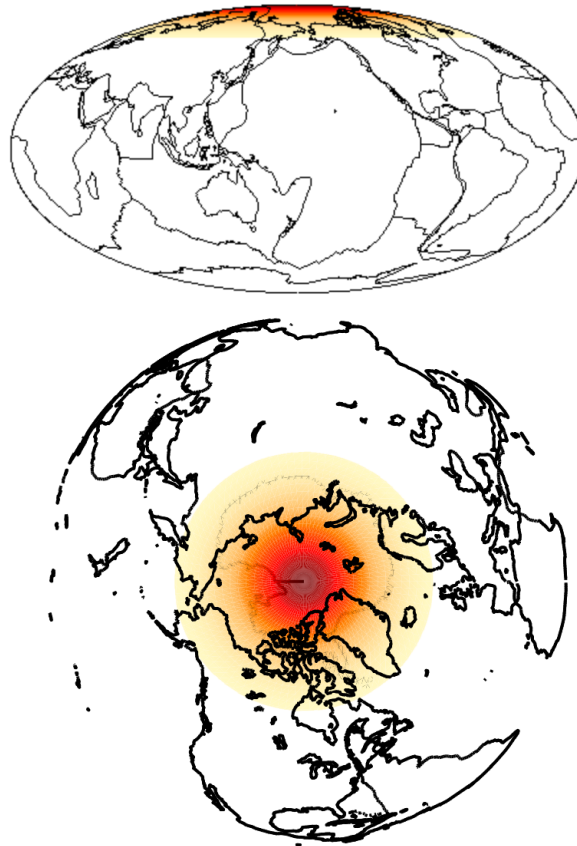

Figure 2:   The first Slepian function of a polar cap of opening angle 40 degrees plotted in both the Mollweide projection (top) and on a sphere with the North pole rotated to the center of the image (bottom).

Exercise: The first Slepian function appears roughly symmetric. Try plotting other Slepian functions and see how the distribution changes when they are less concentrated in the specified polar cap.

2.3   Calculating Coefficients for an Axisymmetric Double Polar Cap

The double polar cap is very similar to our previous example. The only difference is choosing our third input into `glmalpha()` to be the value 2 to indicate a double polar cap. The opening angle is still chosen to be 40.

Run:

```
demos_chapter_two(2.3)
```

We plot with a Mollweide projection and on the sphere with the view of the North Pole and South pole.
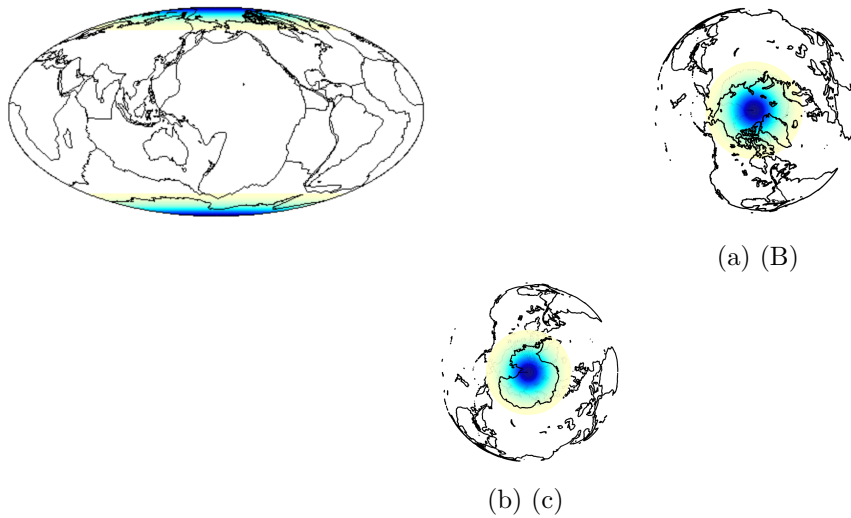
(a) (B)

(b) (c)

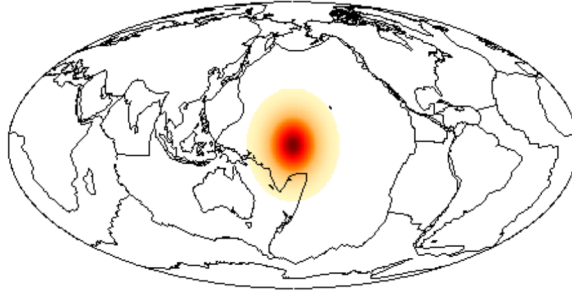Figure 3: The first Slepian function of a double polar cap of opening angle 40 degrees plotted in the Mollweide projection (top), on a sphere with the North pole rotated to the center of the image (bottom left), with the South pole rotated to the center (bottom right).

2.4   A Rotated Polar Cap

If the region of interest is a polar cap or double polar cap that is not axisymmetric and needs to be rotated from the North Pole then a different function is used to produce our matrix of coefficients, `G`.

Run:

```
demos_chapter_two(2.4)
```

Instead of `glmalpha()` we are using `glmalphatoJ()` which requires us to give the following inputs:

`TH` - the opening angle
`L` - maximum spherical harmonic degree
`phi` - the center longitude in degrees of the polar cap of interest
`theta` - the center colatitude in degrees of the polar cap of interest
`omega` - the anticlockwise azimuthal rotation in degrees (rotation of the region itself if necessary)
`J` - the number of Slepian functions to be calculated

This function requires you to know the center colatitude and longitude of your region and to choose the number of Slepian functions to be calculated. For the demo we chose our polar cap to be centered on the antimeridian at the equator, and chose J=50 as we did for earlier examples in which L=20.

7

The first Slepian function was evaluated and plotted, however it should be noted that for `glmalphaptoJ()` this does not necessarily correspond to the "best" concentrated Slepian function overall as in our examples above. Instead, the matrix `G` in this case contains column vectors where the best of every order m was calculated first, then the second best of that order, and so forth, repeating this process for each order. So be careful when using this function.

The plot produced by the demo should match that of figure 4 below:

Figure 4: The first Slepian function of a polar cap of opening angle 40 degrees plotted in the Mollweide projection. This polar cap has been rotated with its center on the antimeridian at the equator using *glmalphaptoJ()*.

2.5   Calculating Coefficients for a Polar Ring

Our next example involves a polar ring, or the difference of two polar caps of different opening angle.

Run:

```
demos_chapter_two(2.5)
```

and observe the output.

For this demo, `glmalpharing()` was used to produce our matrix of coefficients. `TH` was given to be a vector of two values in degrees `[20 40]` that represent the opening angles of two regions. The difference between these regions is our ring. This function can also accept just one value and calculate a normal polar cap.

`glmaalpharing()` only requires three inputs: the opening angle(s) of the polar cap(s), `mL`, and a value of 1 to indicate if the `G` matrix will be sorted in order of its eigenvalues (which is also the default input if nothing is specified).
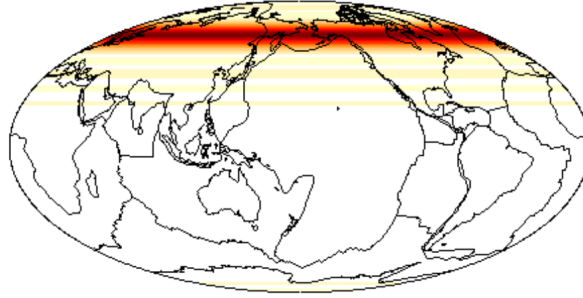
With plotting you should see:

Figure 5: The first Slepian function of a polar ring which is the difference between two polar caps of opening angle 40 degrees and 20 dgrees, plotted in the Mollweide projection. *glmalphaptoJ()*.

3 Linear Combinations of Classical Scalar Slepian Functions

Let's do another quick example where we plot a linear combination of Slepian functions. We can add any combination the columns of `G` together and also multiply by any value or a vector of scalars. We can repeat the steps from earlier and run:

```
lmcs=coef2lmcosi((5*G(:,1))+(3*G(:,30)),1)
data=plm2xyz(lmcs,0.5)
```

And if you plot it you should see:
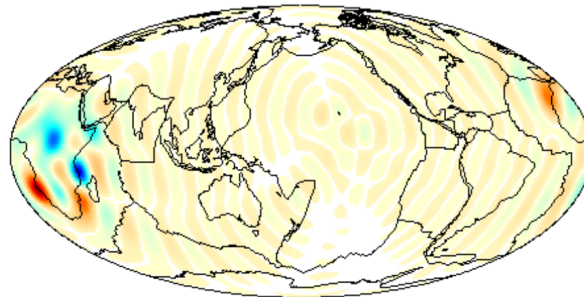
Exercsie: Practice plotting different linear combinations of slepian functions. Try multiplying by a vector of random coefficients.

4   Choosing The Number of Slepian Functions

We will now take a look at our vector of eigenvalues, `V`. You can plot the eigenvalues versus their corresponding slepian function j by setting `J=(L+1)^2` and running the following:

```
plot(1:J,V)
```

Using the `V` from Example One we produce Figure 2 from which we can see that our eigenvalues drop off rapidly in value and become zero near $\jmath = 50$.
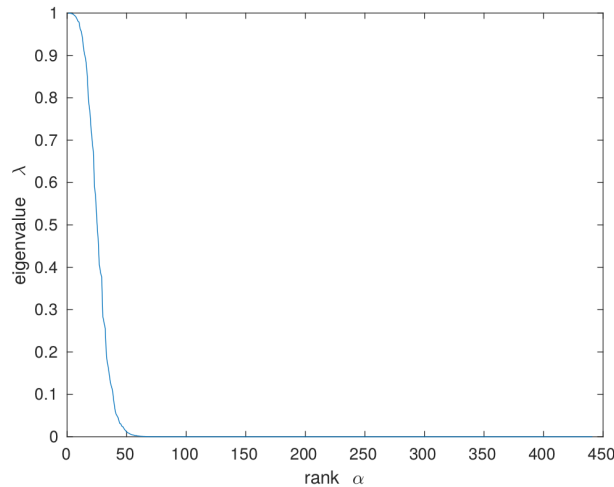

Figure 6:   Eigenvalues plotted on the y-axis versus their corresponding Slepian function number on the x-axis. We can see that most of the energy is occuring in functions corresponding to $j <= 50$.

Plotting this curve may be useful in selecting which Slepian functions will best represent the region of interest, as you will soon see.

Lets first plot a linear combination of all solutions and see what it looks like. To evaluate all coefficients you can run:

```
for j=1:J
    g{j} = plm2xyz(coef2lmcosi(G(:,j),1),.5)
end
```

We can then add the different elements of `g` into one matrix `f`:

```
f = zeros*size(g{1}))
for j=1:J
    f = f + g{j}
```

```
end
```

and then plot with:

```
plotplm(f,[],[],4,.5)
```

Your figure should match that of Figure 3. We can now see that using all of the eigensolutions does not work when we are trying to define a region like Africa.
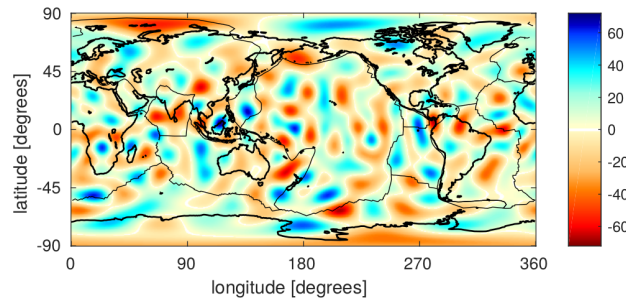

Figure 7: A linear combination of all eigensolutions. L=20.

If we refer back to Figure 2 we can see that our eigenvalues drop off to zero around $j = 50$. Let's now reset `J=50` and see what our plot looks like. You should be able to produce Figure 4, in which we see that our information has been concentrated within our region of choice, Africa.
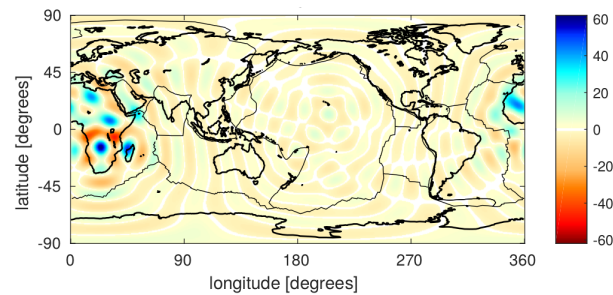

Figure 8: Using a linear combination of the first 50 slepian functions to represent Africa.

Exercise: Create a vector of random numbers the same length as the matrix `f` we plotted in Figure 4 and multiply it with `g{j}` inside the `for` loop. How does this change the plot when using the same eigensolutions? You should produce something similar to Figure 5.
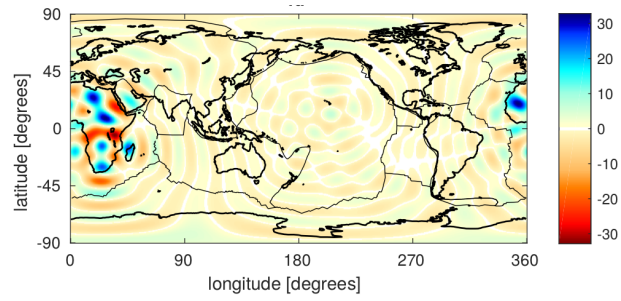
Figure 9: Using a linear combination of the first 50 slepian functions multiplied by a vector of random coefficients.

This tutorial is currently under construction. Please check back later for more by keeping your software updated.