

Chapter 2: Introduction to Slepian Functions

Alain Plattner, Kylee Ford, Sarah Kroecker

July 10, 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. The functions will be bandlimited by the maximum spherical harmonic degree L for region of concentration R . We will choose the continent **Africa** as R for the following example in which we will calculate and plot spherical harmonic coefficients.

2.1 Example One - Using `glmalpha()` and plotting evaluated Slepian function coefficients

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

TH - a desired region, in our case: `'africa'`. This may also be any other spherical cap or ordered list defining a closed curve

L - the bandwidth, or maximum spherical harmonic degree

Thus the outputs of `glmalpha` we will need for this example are a unitary matrix of localization coefficients `G` and the eigenvalues `V`. There are several other inputs and outputs that may and should be used; see `help glmalpha` for more details.

We will 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 are using 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.

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

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

and observe the output `G` which should be a $(L+1)^2 \times (L+1)^2$ orthonormal matrix of coefficients and `V` a single vector of eigenvalues. 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 function, 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_{lm}

l	m	co	si
0	0	c_{00}	0
1	0	c_{10}	0
1	1	c_{1-1}	c_{11}
2	0	c_{20}	0
2	1	c_{2-1}	c_{21}
2	2	c_{2-2}	c_{22}
\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 because these are the coefficients corresponding to the “best” Slepian function as discussed above.

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,[],[],4,0.5)
```

for which our input choices follow the second input scheme of `help plotplm`.

Run `kelicol(1)` for a more clear color scheme where white sections are of value 0, and you should see a map of Earth projected into a 2D plane with latitude as the y-axis and longitude on the x-axis. 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. Your plot should match that of Figure 1.

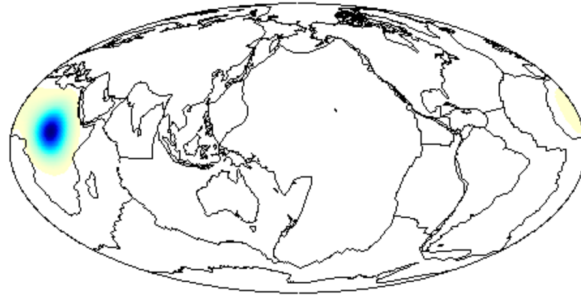
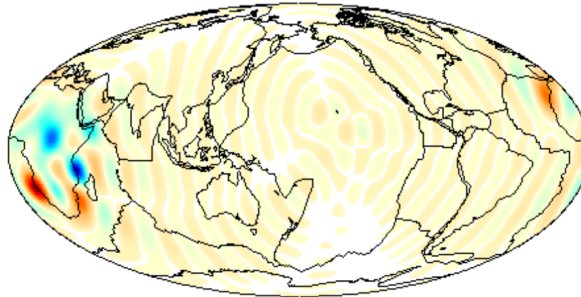


Figure 1: Result of example using `glmalpha()` with $L=20$ to calculate coefficients, `plm2xyz()` to evaluate them, and `plotplm()` to plot with method 1 - the Mollweide projection. Only the first “best” concentrated eigensolutions are plotted. The color scheme is `kelicol(1)` and max/min values are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

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))+(10*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.

Exercise: Pick a different a region, or subtract a region from Africa to use as the domain and plot again.

2.2 Example Two - Looking at Eigenvalues

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 $j = 50$.

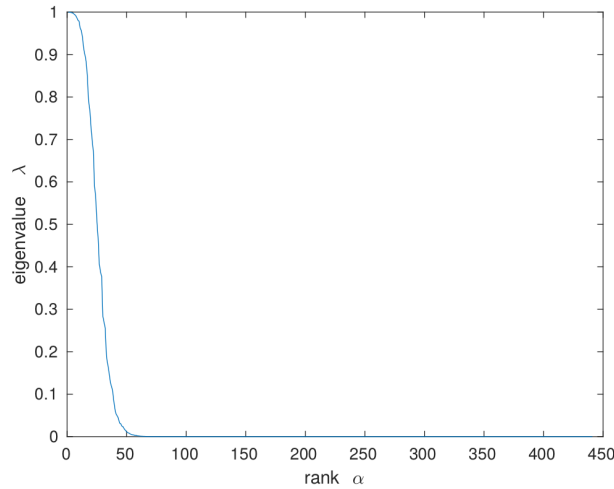


Figure 2: Eigenvalues plotted on the y-axis versus their corresponding Slepian function number on the x-axis. We can see that most of the energy is occurring in functions corresponding to $j \leq 50$.

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.

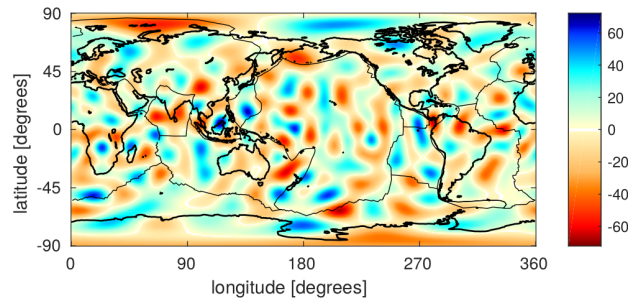


Figure 3: A linear combination of all eigensolutions. $L=20$.

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.

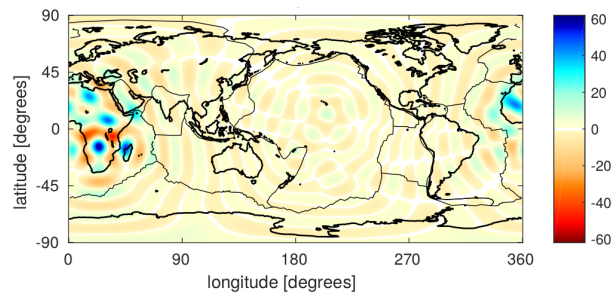


Figure 4: Using a linear combination of the first 50 slepian functions to represent Africa.

Exercise: Create a vector of random numbers the same length as the matrix \mathbf{f} we plotted in Figure 4 and multiply it with $\mathbf{g}\{j\}$ inside the **for** loop. How does this change the plot when using the same eigensolutions? You should produce something similar to Figure 5.

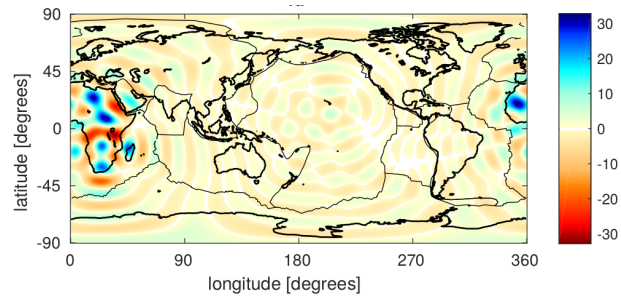


Figure 5: Using a linear combination of the first 50 slepian functions multiplied by a vector of random coefficients.

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