Chapter 3: Introduction to altitude-cognizant scalar Slepian functions

Kylee Jo Ford, Sarah Kroeker, Alain Plattner

July 12, 2018

1 Introduction

Now that we understand the classical scalar Slepian functions, we can consider the altitude of a satellite from which the data was taken. Atlitude-cognizant scalar Slepian functions can optimize the linear combination of spherical harmonics by either a downward continuation from the satellite radius to the planet's surface or a upward continuation from the planet's surface to the satellite radius. The altitude-cognizant scalar Slepian functions (AC-SSF) are constructed through the maximum spherical harmonic degree L, region of concentration, planet radius, and the average satellite radius. Since there is 'negligible' potential field in the area between the planet's surface and the satellite altitude, we optimize the potential from the planet to create a model. The AC-SSF optimizes the linear combination of scalar spherical harmonics to incorporate the satellite altitude at which the data was taken. Basically, we are able to take the field data collected by a satellite at some altitude and, through an optimal linear combination, we are able to plot the field on the planet's surface.

*See Plattner and Simons (2017) for an in depth explanation.

2 Application

The following will include the methods for which we put Slepian functions to use.

2.1 Using glmalphapotup.m

To acquire the Slepian functions, given a data set, we may use the function `glmalphapotup`. This function requires the following inputs: TH (or dom for domain), L,

rnew – average satellite altitude

rold – planet radius

The outputs of this function are G, the matrix of spherical harmonic coefficients on the planet's surface, and V, the eigenvalues or the suitability factors. For all options for this function, see `help glmalphapotup`.

In our example, we will use a familiar domain, Africa. Set the paramaters:

```
dom = 'africa';
L = 20;
rnew = 6371+400;
rold = 6371;
```

As mentioned in the previous chapter, we must know whether our spherical harmonic coefficients are in ADDMOUT or ADDMON ordering. By using the `help` function, it is clear that the output is in ADDMOUT ordering; this will be useful information in a moment. So, we can run:

```
[G,V] = glmalphapotup(dom,L,rnew,rold);
```

2.2 Plotting

However, to plot the Slepian function at satellite altitude, we must upward continue the functions by using the function `potup`. This requires the inputs: G, radius to the satellite, planet radius, and L.

2.2.1 Example One:

For this example, we will use the spherical harmonic coefficient for the optimal Slepian function, which would be the first column of G. Let's first plot the Slepian functions on the surface of the planet. We will first need to convert our G into lmcosi, since G is in ADDMOUT, so run:

```
lmcosi = coef2lmcosi(G(:,1),1);
```

To plot the Slepian functions on the surface of the planet, we may use the same functions as in the previous chapter, for the classical scalar Slepian functions: `plm2xyz` and `plotplm`. Now set the resolution:

```
res = 1;
```

and convert the coefficients to xyz coordinates by running:

```
data = plm2xyz(lmcosi,res);
```

Finally, to plot the figure on a flat surface, run:

```
plotplm(data,[],[],4,res)
```

Let's add a colorbar and make the color scale nicer by running:

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

To plot the figure on a 3-D sphere, run:

```
plotplm(lmcosi,[],[],2,res)
```

For the results, see Figure 1.

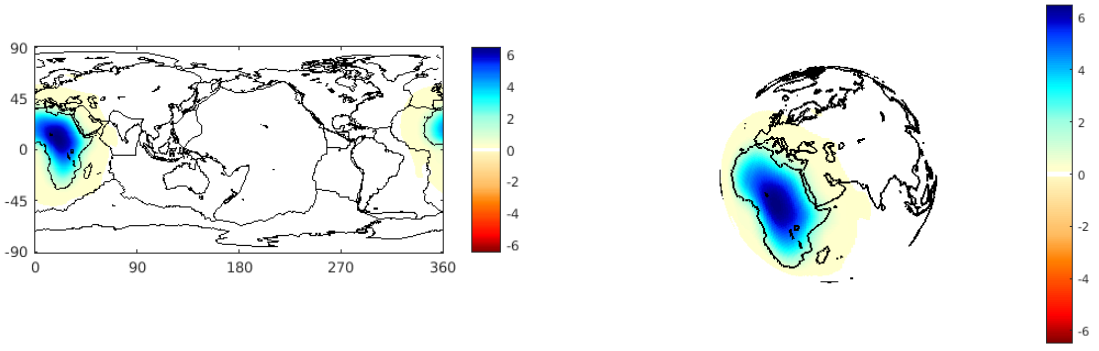

Figure 1: Plot of G(:,1).

Let's now use the function `potup` to upward continue the Slepian functions to satellite altitude. We can now run:

```
coef = potup(G(:,1),rnew,rold,L,0);
```

providing the output of the spherical harmonic coefficient vector for radial field at the new altitude. We need to convert this to lmcosi format:

```
lmcosi = coef2lmcosi(coef,1);
```

We can plot the functions at the satellite altitude. To plot on a flat surface, run:

```
data = plm2xyz(lmcosi,res);
plotplm(data,[],[],4,res)
caxis([-1,1]*max(abs(caxis)))
kelicol(1)
```

To plot on a sphere, run:

```
plotplm(lmcosi,[],[],2,res)
```

For the results, see figure 2.

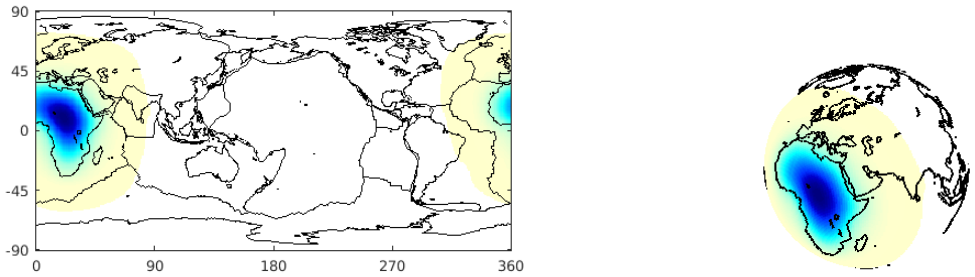

Figure 2: Upward continued G(:,1).

Exercise: Let's try another region, say a spherical cap:

```
dom = [16]
```

Run every step above again and plot on a sphere. What is the result? Compare your result to figure 3.

Exercise: We can try another region, say a spherical ring:

```
dom = [100,25]
```

What is the result? Compare to figure 3.

Figure 3: Result with dom = 16 and dom = [100,25]

See the `help` function to see what other regions we can choose for `glmalphapotup`.

Exercise: Let's try to rotate a region. We will walk through this together. In order to rotate our region, we must use the function, `glmalphapotuptoJp`. This function requires an angular extent of a spherical cap or angles between two spherical caps, where we want to look at the ring between them. See `help glmalphapotuptoJp` for more information on the required parameters. Let's set our `TH` and other paramaters as follows:

```
TH = 16;
phi = 90;
theta = 20;
omega = 25;
J = 50;
```

Now let's get our spherical harmonic coefficients:

```
[Grot,V] = glmalphapotuptoJp(TH,L,rnew,rold,phi,theta,omega,J);
```

We can now upward continue these and plot them:

```
coef = potup(Grot(:,1),rnew,rold,L,0);
lmcosi = coef2lmcosi(coef,1);
data = plm2xyz(lmcosi,res);
plotplm(data,[],[],4,res)
```

Plotting on the 3D sphere:

```
plotplm(lmcosi,[],[],2,res)
```

The rotated spherical cap is shown in figure 4.

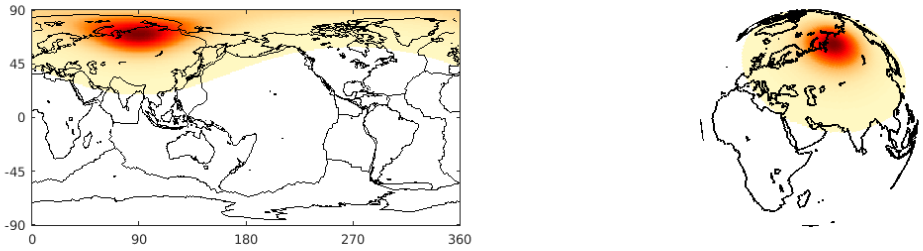

Figure 4: Rotated spherical cap of TH = 16.

Exercise: Let's try a rotated ring. We will need to change only the `TH` so we have a ring. So set:

```
TH = [100,25]
```

Now we can obtain the spherical harmonic coefficients:

```
[Grot,V] = glmalphapotuptoJp(TH,L,rnew,rold,phi,theta,omega,J);
```

Upward continue and plot:

```
coef = potup(Grot(:,1),rnew,rold,L,0);
lmcosi = coef2lmcosi(coef,1);
data = plm2xyz(lmcosi,res);
plotplm(data,[],[],4,res)
```

This rotated spherical ring is shown in figure ...

Exercise: Try the second best Slepian function, now in Africa:

```
dom = 'africa';
[G,V] = glmalphapotup(dom,L,rnew,rold);
coef = potup(G(:,2),rnew,rold,L,1);
lmcosi = coef2lmcosi(coef,1);
```

```
plotplm(lmcosi,[],[],2,res)
kelicol(1)
```

This results in figure 5. Compare with figure 2. What do you notice?

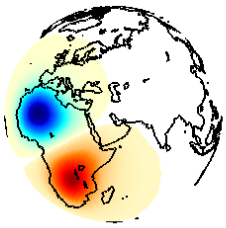

Figure 5: Plot with the second best Slepian function, G(:,2).

2.3 Linear combinations of Slepian functions

Now that we know how to create and plot the Slepain functions at satellite altitude with different regions, we can create a linear combination of the altitude-cognizant Slepian functions. What does the linear combination

```
h = G(:,1)+G(:,2);
```

look like? You will need to input this linear combination into your command line and run the same process, with `h` instead of `G(:,1)` in the functions:

```
coef = potup(h,rnew,rold,L,1);
lmcosi = coef2lmcosi(coef,1);
plotplm(lmcosi,[],[],2,res)
caxis([-1,1]*max(abs(caxis)))
kelicol(1)
```

This should result in figure 6. Compare this figure with figures 2 and 4. What do you notice?

Exercise: Try some of your own linear combinations and examine the plots.

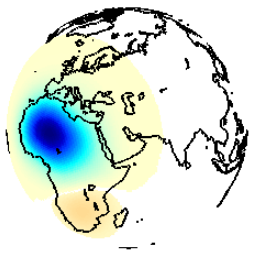

Figure 6: Plot with h = G(:,1)+G(:,2).

2.3.1 Example Two:

Now, let's try a set of linear combinations including all of the spherical harmonic coefficients in our case. The length of our matrix, G, is $(L+1)^2$. Let:

```
K = length(G)
```

We need to determine a proper value for `J` to plot the optimal linear combination of Slepian functions. We will first need to plot our index `j` versus `V`:

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

Now we can analyze the plot to determine which value of J such that most of the eigenvalues are concentrated within our region. Our plot will look like figure 7, as an exponential decay curve.

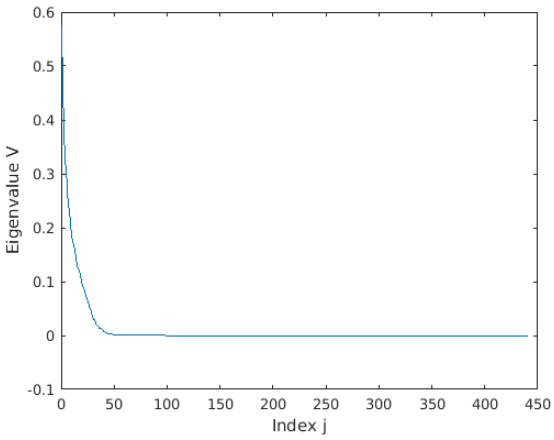

Figure 7: Eigenvalue V vs index j.

By analyzing the graph, it looks like the eigenvalues near zero at about $J = 50$. You will have to manually choose your J value, so run:

```
J = 50;
```

We want to choose some random values to create our linear combination:

```
u = randn(K,1);
```

and we will need to create a a matrix, `f`, which is the same size as our coefficients. Run:

```
f = zeros(size(G(:,1)));
```

Now we can make a `for` loop to create a linear combination of the coefficients by running:

```
for j=1:J
 f = f + u(j)*G(:,j);
end
```

This will create a matrix, `f`, containing the linear combinations of spherical harmonics. Now we can sort them into the correct format and plot:

```
[F,lon,lat] = plm2xyz(coef2lmcosi(f,1),1);
plotplm(F,lon,lat,4,res)
```

The plot of F is shown in figure 8. We can alternatively directly plot this by:

```
plotplm(coef2lmcosi(f,1),[],[],4,res)
```

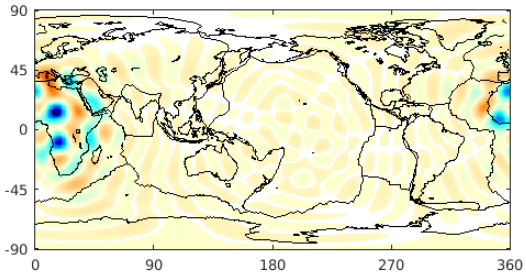

Figure 8: Random linear combination for eignenvalues up to J = 50.

Now, let's use all of the Slepian functions, rather than just up to our value `J`. To do this, let's run a similar code with eigenvalues `1:K`.

```
f2 = zeros(size(G(:,1)));
for j=1:K
 f2 = f2 + u(j)*G(:,j)
end
```

Plot:

```
plotplm(coef2lmcosi(f2,1),[],[],4,res)
```

Figure 9 shows the plot of all of the eigenvalues.

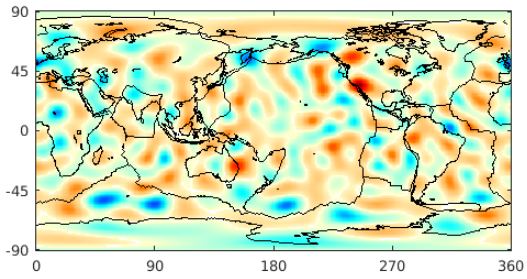

Figure 9: Random linear combination for all eignenvalues.

Now that we understand the building of the linear combinations, we can use `potup` to upward continue the coefficients. We want to use our original `f`, since it would be the best linear combination of Slepian functions. We can do this in one step:

```
fup = potup(f,rnew,rold,L,1);
```

Now we can plot the upward continued coefficients:

```
plotplm(coef2lmcosi(fup,1),[],[],4,res)
```

The best fit Slepian functions, upward continued, are shown in figure 10.

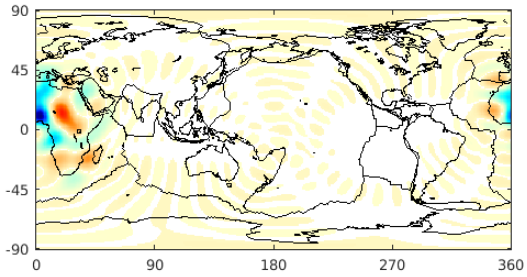

Figure 10: Best linear combination of Slepian functions up to J = 50, upward continued.

Question: Choose a lower J value and compare the plots. What do you notice?

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