process-sam-gravity-data

March 23, 2016

1 Explore and process the South America gravity data

This notebook loads and processes the raw gravity data in ../data/goco05s-sam-s0.1deg-h50km.gdf. The data are gravity values in mGal. This notebook calculates the gravity disturbance, performs topographic correction, calculates and removes the effect of sediments (taken from the CRUST1.0 model). The results are saved to text files in the data folder for use in inversion and making figures.

1.1 Package imports

```
In [1]: %matplotlib inline
```

Load the standard scientific Python stack to numerical analysis and plotting.

```
In [2]: from __future__ import division
        import datetime
        import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits.basemap import Basemap
        import multiprocessing
        import seaborn  # Makes the default style of the plots nicer
```

The computations generate a lot of run-time warnings. They aren't anything to be concerned about so disable them to avoid clutter.

```
In [3]: import warnings
        warnings.simplefilter('ignore')
```

Load the required modules from Fatiando a Terra.

```
In [4]: from fatiando.gravmag import tesseroid, normal_gravity
        from fatiando import gridder, utils
        import fatiando
```

```
In [5]: print("Version of Fatiando a Terra used: {}".format(fatiando.__version__))
```

```
Version of Fatiando a Terra used: 237ba1dd35d47ea0a5e286781faddfe60ab4d12d
```

Load our custom classes and functions.

```
In [6]: from mohoinv import TesseroidRelief, make_mesh
        from datasets import fetch_crust1, load_icgem_gdf, down_sample
```

Get the number of cores in this computer to run the some things in parallel.

```
In [7]: ncpu = multiprocessing.cpu_count()
        print("Number of cores: {}".format(ncpu))
```

```
Number of cores: 8
```

1.2 Load and plot the raw data

```
In [8]: data = load_icgem_gdf('../data/goco05s-sam-s0.1deg-h50km.gdf')
```

The data is return in a Python dictionary. The following fields are read from the file:

```
In [9]: print(data.keys())
```

```
['area', 'longitude', 'height', 'shape', 'latitude', 'gravity_ell', 'metadata']
```

`metadata` is the file header.

```
In [10]: print(data['metadata'])
```

```
generating_institute     gfz-potsdam
     generating_date     2015/08/13
        product_type     gravity_field
                body     earth
           modelname     goco05s
     max_used_degree          280
         tide_system     zero_tide
          functional     gravity_ell  (centrifugal term included)
                unit     mgal
          refsysname     WGS84
            gmrefpot      3.98600441800E+14 m**3/s**2
        radiusrefpot     6378137.000 m
          flatrefpot      3.352810664747480E-03   (1/298.25722356300)
         omegarefpot      7.29211500000E-05 1/s
       long_lat_unit     degree
      latlimit_north       20.000000000000
      latlimit_south      -60.000000000000
      longlimit_west       270.00000000000
      longlimit_east       330.00000000000
            gridstep      0.10000000000000
     height_over_ell      50000.0000 m
  latitude_parallels          801
 longitude_parallels          601
number_of_gridpoints       481401
            gapvalue        9999999.0000
       weighted_mean      9.6376213E+05 mgal
            maxvalue      9.6671268E+05 mgal
            minvalue      9.6273315E+05 mgal
         signal_wrms      1.1097072E+03 mgal
         grid_format     long_lat_value

          longitude    latitude    gravity_ell
            [deg.]      [deg.]          [mgal]
```

We'll need to down sample this data set to a larger grid spacing because it's too large for modeling.

```
In [11]: arrays = data['latitude'], data['longitude'], data['height'], data['gravity_ell']
         downsample_every = 2
         lat, lon, height, grav, shape = down_sample(arrays, data['shape'],
                                                     every=downsample_every)
         area = (lat.min(), lat.max(), lon.min(), lon.max())
         print("Data area (S, N, W, E): {}".format(area))
         print("Number of points in latitude and longitude: {}".format(shape))
         print("Original dataset size: {}".format(data['shape']))
```

```
Data area (S, N, W, E): (-60.0, 20.0, 270.0, 330.0)
Number of points in latitude and longitude: (401, 301)
Original dataset size: (801, 601)
```

Setup a basemap to plot the data with an appropriate projection.

```
In [12]: bm = Basemap(projection='cyl',
                      llcrnrlon=area[2], urcrnrlon=area[3],
                      llcrnrlat=area[0], urcrnrlat=area[1],
                      lon_0=0.5*(area[2] + area[3]), lat_0=0.5*(area[1] + area[0]),
                      resolution='l')
```

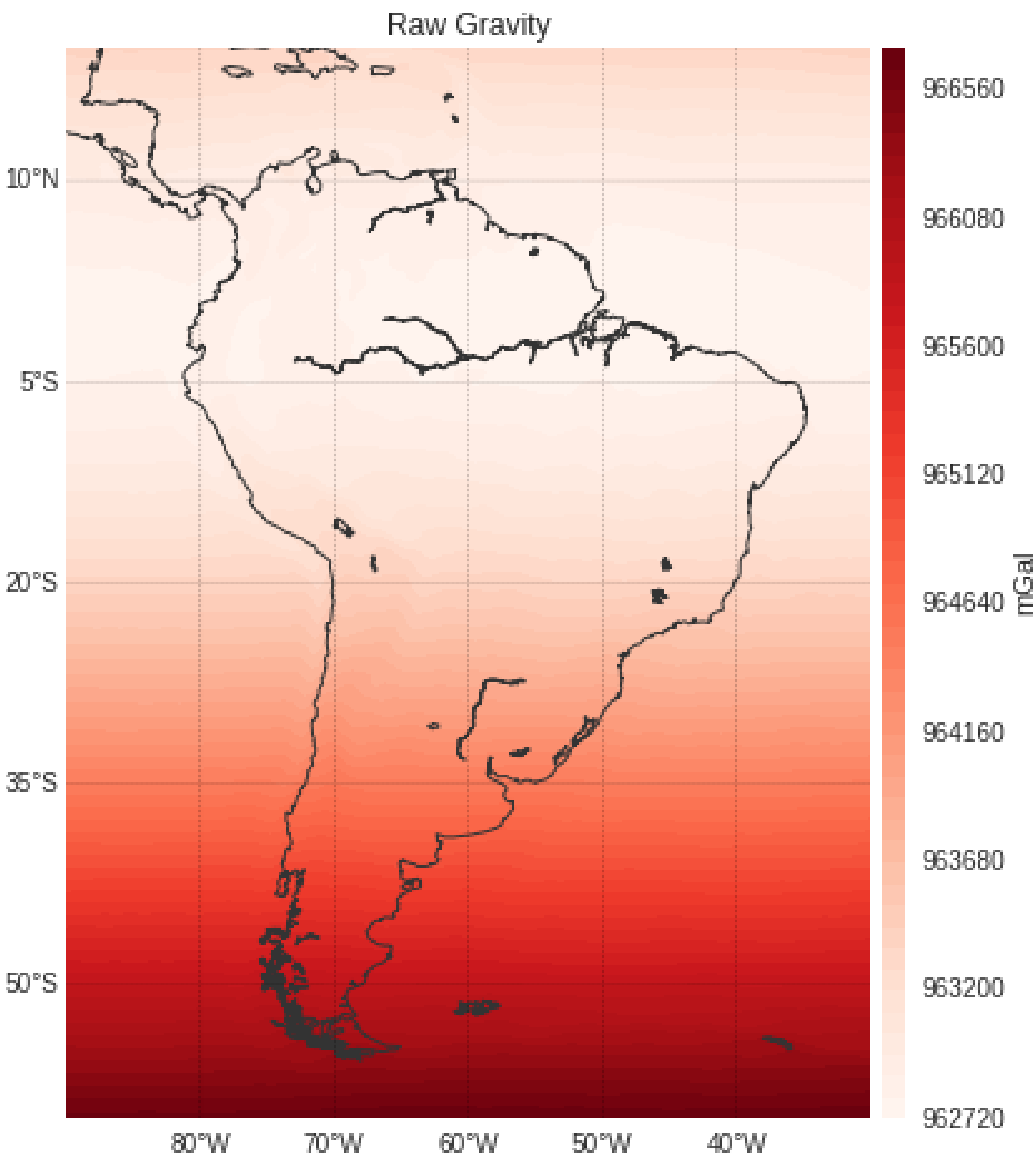
Make a plotting function to avoid repeating this code.

```
In [13]: def plot_data(lat, lon, data, shape, cmap, cblabel='mGal', levels=60, ranges=True):
             x, y = bm(lon, lat) # Transform lat and lon into plot coordinates
             kwargs = dict(cmap=cmap)
             if ranges:
                 ranges = np.abs([data.min(), data.max()]).max()
                 kwargs['vmin'] = -ranges
                 kwargs['vmax'] = ranges
             fig = plt.figure(figsize=(7, 6))
             bm.contourf(x.reshape(shape), y.reshape(shape), data.reshape(shape), levels,
                         **kwargs)
             plt.colorbar(pad=0.01, aspect=50).set_label(cblabel)
             bm.drawmeridians(np.arange(-80, -30, 10), labels=[0, 0, 0, 1], linewidth=0.2)
             bm.drawparallels(np.arange(-50, 30, 15), labels=[1, 0, 0, 0], linewidth=0.2)
             bm.drawcoastlines(color="#333333")
             plt.tight_layout(pad=0)
             return fig
```

Plot the raw gravity data.

```
In [14]: plot_data(lat, lon, grav, shape, 'Reds', ranges=False)
         plt.title('Raw Gravity')

Out[14]: <matplotlib.text.Text at 0x7f0cc85d4910>
```



1.3 Calculate gravity disturbance

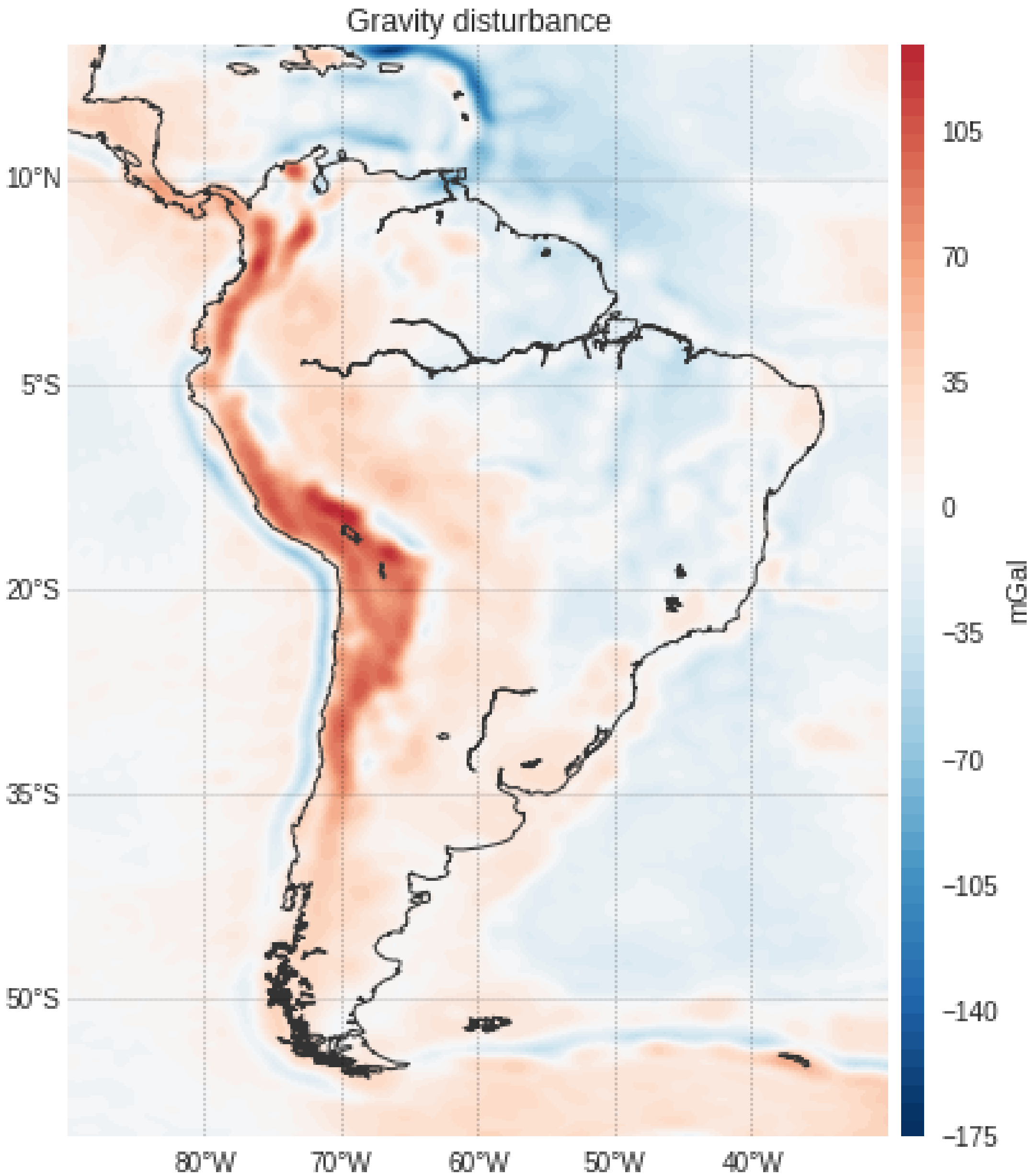
The gravity distance is the raw gravity data minus the effect of the Normal Earth (γ) calculated at the observation point.

Fatiando a Terra offers the `gamma_closed_form` function that calculates γ at any latitude and height using the closed form formula in Li and Gotze (2001).

```
In [15]: disturbance = grav - normal_gravity.gamma_closed_form(lat, height)
```

```
In [16]: plot_data(lat, lon, disturbance, shape, 'RdBu_r')
         plt.title('Gravity disturbance')
```

```
Out[16]: <matplotlib.text.Text at 0x7f0cc83b4150>
```



1.4 Terrain correction

The next step is to remove the effect of the topography and water layer. Let's load the topography data downloaded from ICGEM. This data is ETOPO1 calculated by interpolation on the specified grid points of our data.

```
In [17]: topo_data = load_icgem_gdf('../data/topography-sam-s0.1deg.gdf', usecols=[-1])
```

```
In [18]: print(topo_data['metadata'])
```

```
generating_institute     gfz-potsdam
     generating_date     2015/08/14
```

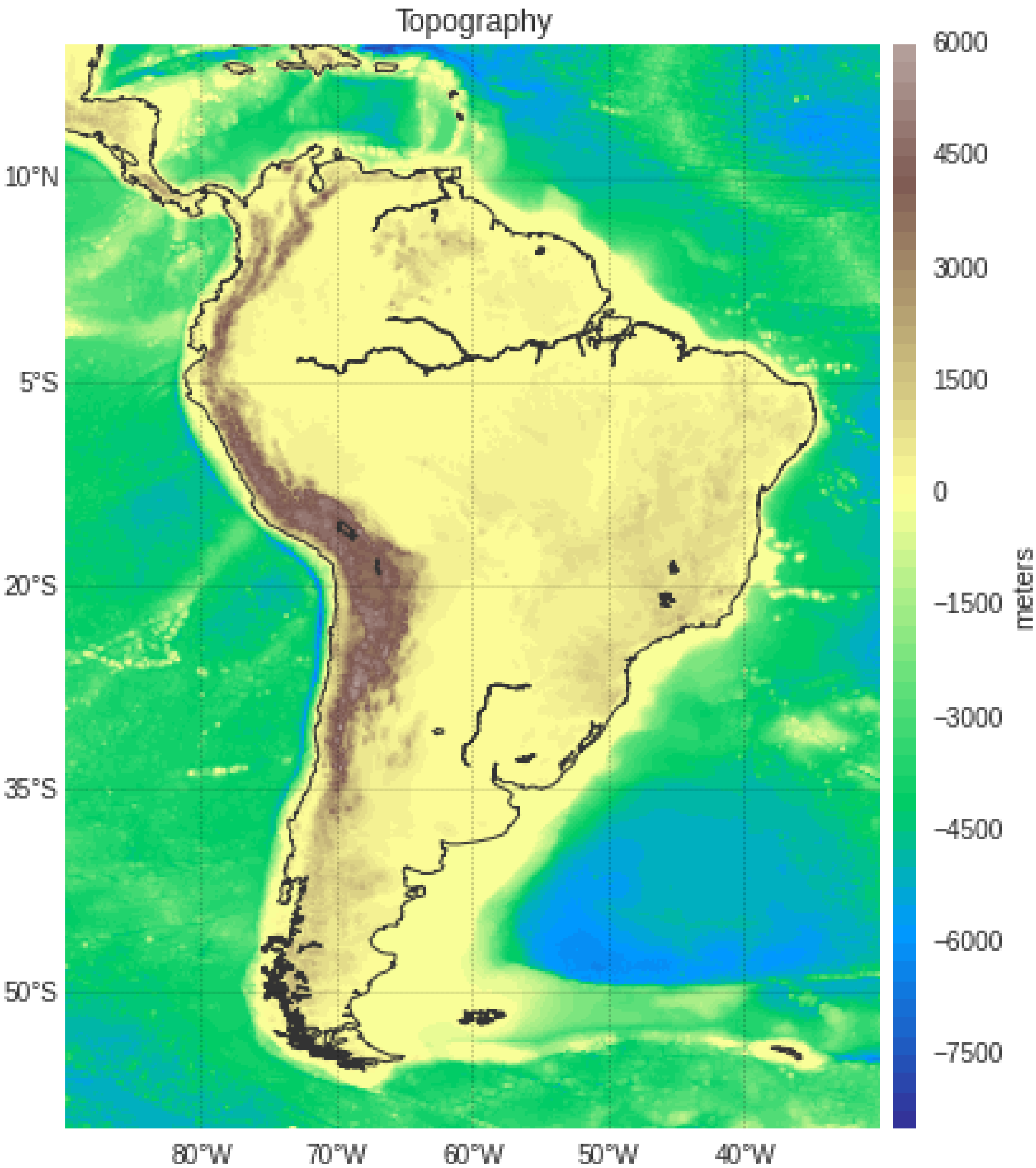
```
        product_type     topography
                body     earth
           modelname     etopo1_bin_int
          functional     topography_grd (grid)=>bi-linear interpolation
                unit     meter
          refsysname     WGS84
        radiusrefsys     6378137.000 m
          flatrefsys      3.352810664747480E-03   (1/298.25722356300)
       long_lat_unit     degree
      latlimit_north       20.000000000000
      latlimit_south      -60.000000000000
      longlimit_west       270.00000000000
      longlimit_east       330.00000000000
            gridstep      0.10000000000000
  latitude_parallels           801
 longitude_parallels           601
number_of_gridpoints        481401
            gapvalue          99999.0000
       weighted_mean     -2.1030878E+03 meter
            maxvalue      6.0260000E+03 meter
            minvalue     -8.3820000E+03 meter
         signal_wrms      2.4270796E+03 meter
         grid_format     long_lat_value

          longitude    latitude    topography_grd
            [deg.]      [deg.]         [meter]
```

In [19]:
```
topo, _ = down_sample([topo_data['topography_grd']], topo_data['shape'],
                      every=downsample_every)
```

In [20]:
```
plot_data(lat, lon, topo, shape, cmap='terrain', cblabel='meters')
plt.title('Topography')
```

Out[20]: <matplotlib.text.Text at 0x7f0cc685ab10>



We'll make a tesseroid model of the topography so that we can calculate it's gravitational effect in spherical coordinates. The `make_mesh` function of `mohoinv.py` automates this process for us. Each point in the topography grid is at the center of the top face of a tesseroid.

```
In [21]: topo_model = make_mesh(area, shape, topo, reference=0)
```

Now we need to set a density value for the topography. We'll use the standard 2670 kg/m^3 for the rocks (topography > 0) and -1630 = 1040 - 2670 kg/m^3 for water (topography < 0).

```
In [22]: topo_density = 2670*np.ones(topo_model.size)
         # Density in the oceans is rho_water
         topo_density[topo_model.relief < topo_model.reference] = -1630
         topo_model.addprop('density', topo_density)
```

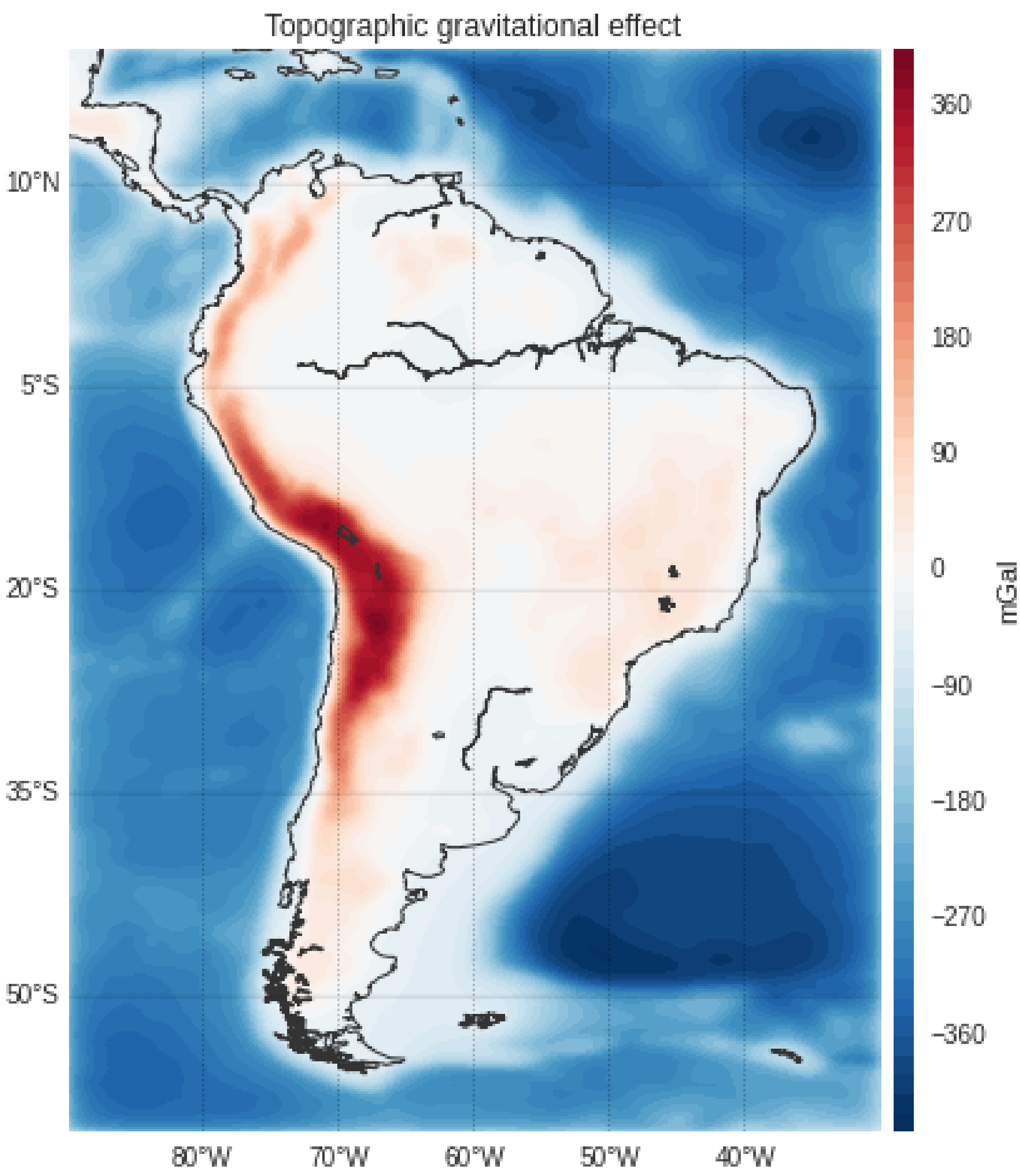
Forward model the effect of the topography in spherical coordinates using tesseroids. Use all available cores for this calculation. It will take a while.

```
In [23]: %time topo_effect = tesseroid.gz(lon, lat, height, topo_model, njobs=ncpu)
```

```
CPU times: user 2.94 s, sys: 888 ms, total: 3.83 s
Wall time: 1h 6s
```

```
In [24]: plot_data(lat, lon, topo_effect, shape, cmap='RdBu_r')
         plt.title('Topographic gravitational effect')
```

```
Out[24]: <matplotlib.text.Text at 0x7f0cc5e26450>
```

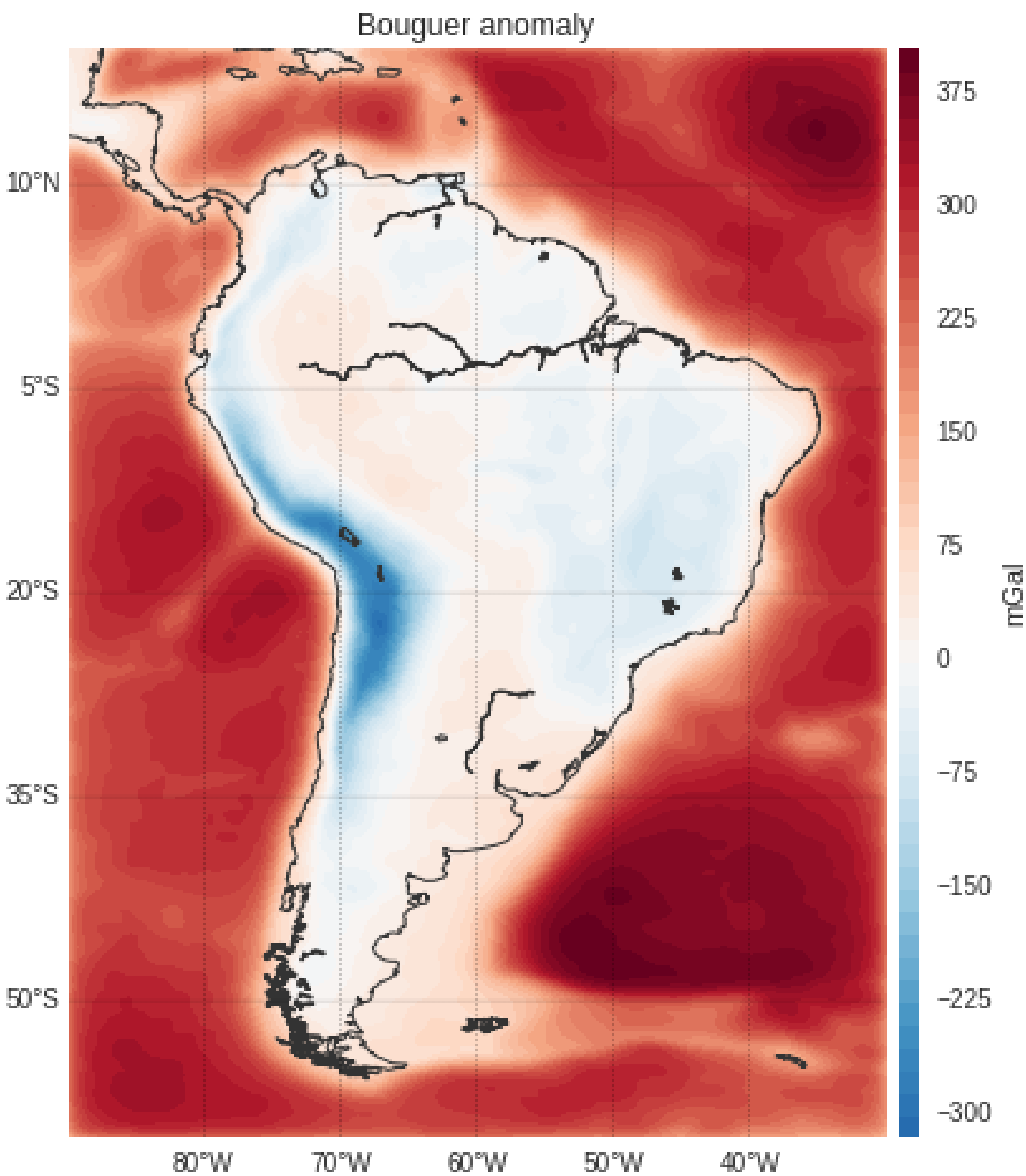


The Bouguer anomaly will be the disturbance minus the effect of the topography.

```
In [25]: bouguer = disturbance - topo_effect
```

```
In [26]: plot_data(lat, lon, bouguer, shape, cmap='RdBu_r')
         plt.title('Bouguer anomaly')
```

```
Out[26]: <matplotlib.text.Text at 0x7f0cfcaf0a90>
```



1.5 Remove the effect of sediments

We need to remove the gravitational effect of the sedimentary layers from our data to isolate the Moho effect. We'll not consider any other crutal density anomalies because South America is not well represented in the

CRUST1.0 model. Instead of assuming a most likely wrong crustal density, we choose to err on the side of simplicity and assume no crustal density anomalies.

Load the CRUST1.0 model for South America.

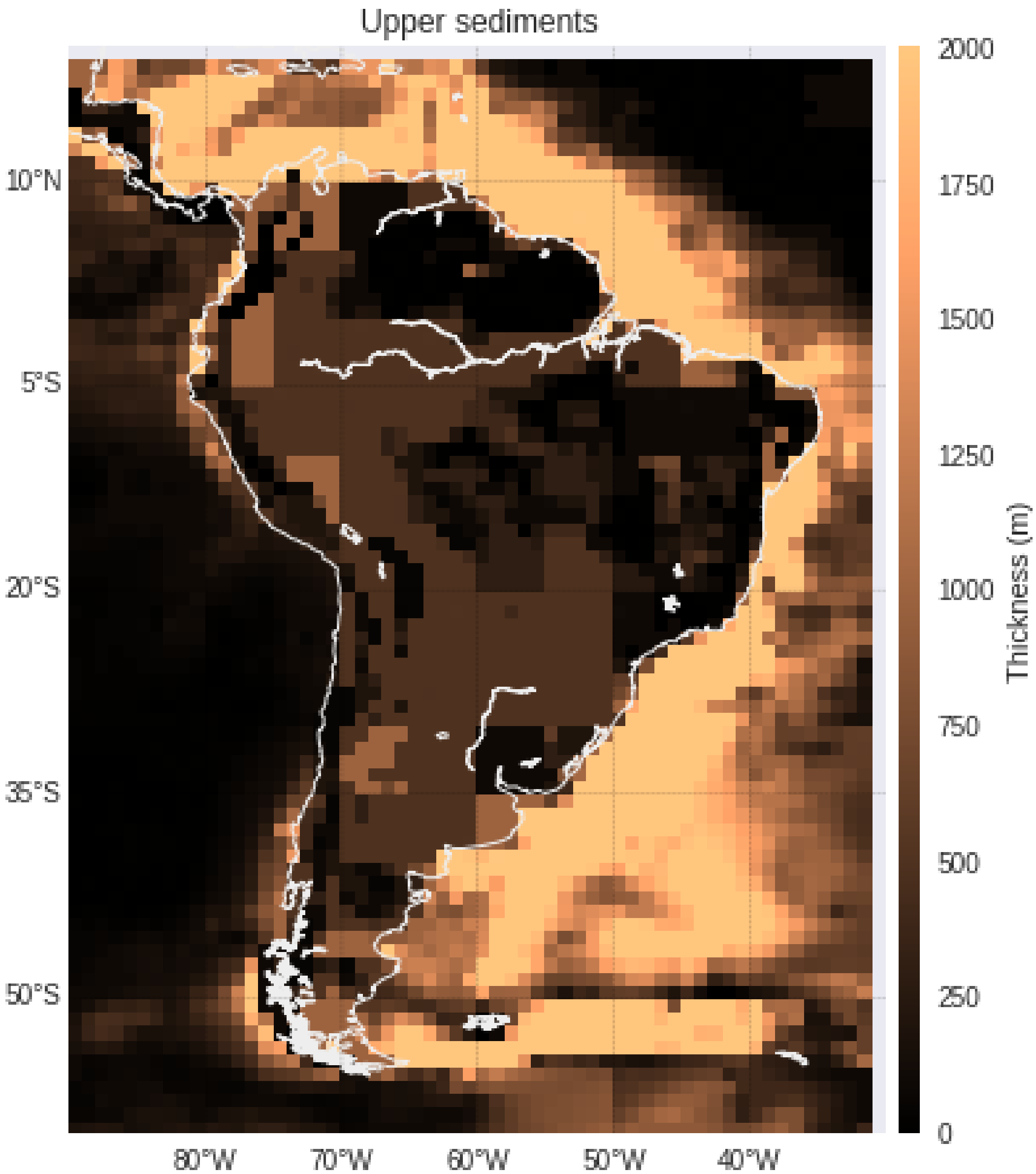
```
In [27]: crust1 = fetch_crust1('../data/crust1.0.tar.gz').cut(area)
```

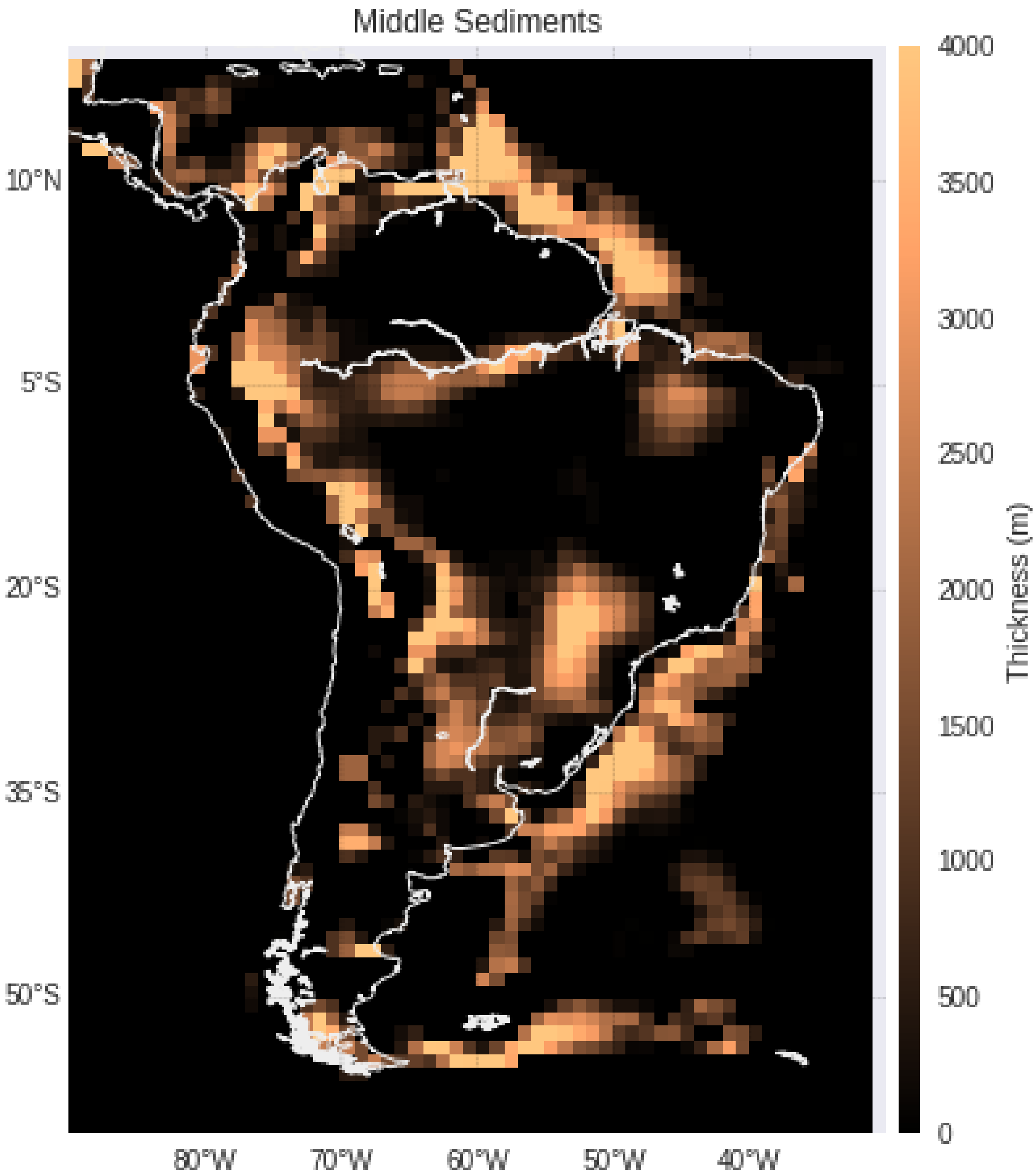
Get the three sedimentary layers from the model. We need to transform their density values into density contrasts with respect to the standard crustal density od 2670 kg/m^3.

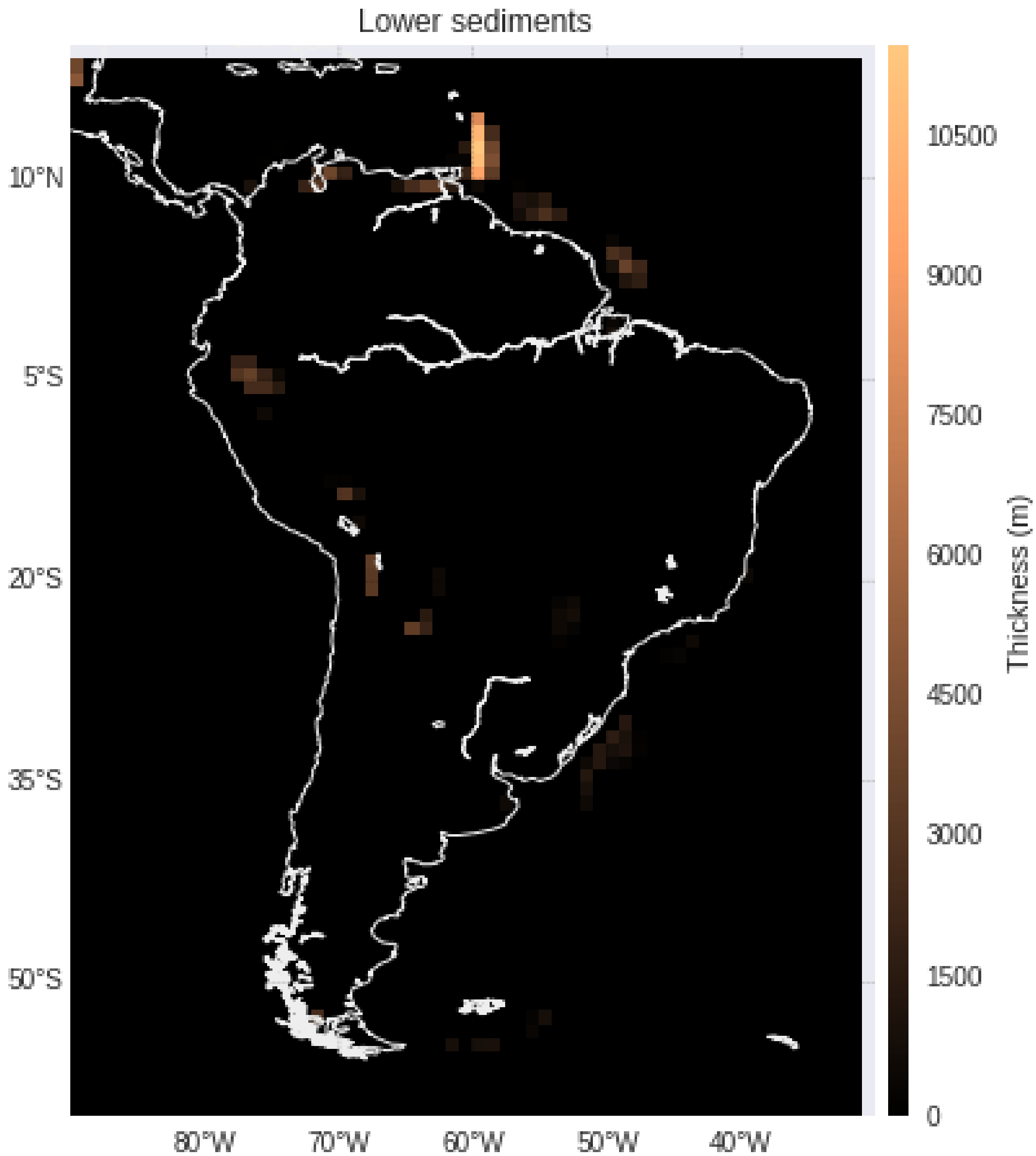
```
In [28]: layers = [l.contrast('density', 2670)
                   for l in [crust1.upper_sediments, crust1.middle_sediments,
                             crust1.lower_sediments]]
         layer_names = ['Upper sediments', 'Middle Sediments', 'Lower sediments']
```

Plot the sediment thickness.

```
In [29]: x, y = bm(crust1.lon + 360, crust1.lat) # Transform lat and lon into plot coordinates
         for layer_name, layer in zip(layer_names, layers):
             fig = plt.figure(figsize=(7, 6))
             bm.pcolormesh(x, y, layer.thickness, cmap='copper')
             plt.colorbar(pad=0.01, aspect=50).set_label('Thickness (m)')
             bm.drawmeridians(np.arange(-80, -30, 10), labels=[0, 0, 0, 1], linewidth=0.2)
             bm.drawparallels(np.arange(-50, 30, 15), labels=[1, 0, 0, 0], linewidth=0.2)
             bm.drawcoastlines(color="#eeeeee")
             plt.title(layer_name)
             plt.tight_layout(pad=0)
```


Upper sediments
10°N
5°S
20°S
35°S
50°S
80°W
70°W
60°W
50°W
40°W
2000
1750
1500
1250
1000
750
500
250
0
Thickness (m)


Middle Sediments
10°N
5°S
20°S
35°S
50°S
80°W
70°W
60°W
50°W
40°W
4000
3500
3000
2500
2000
1500
1000
500
0
Thickness (m)



Calculate the gravitational effect of each layer and the total effect of all layers.

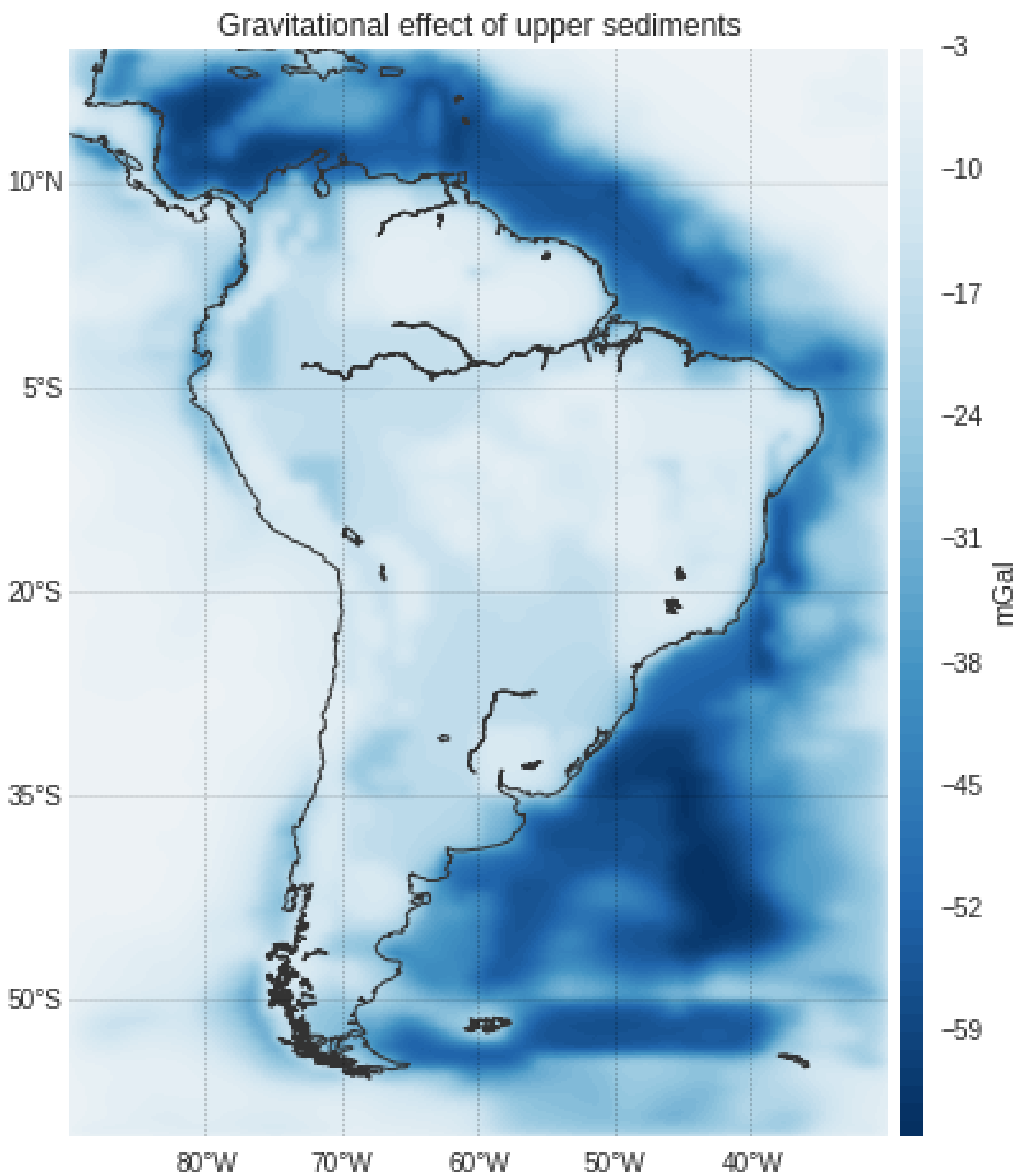
```
In [30]: %%time
         sediment_effects = [tesseroid.gz(lon, lat, height, list(layer.tesseroids), njobs=ncpu)
                             for layer in layers]
```

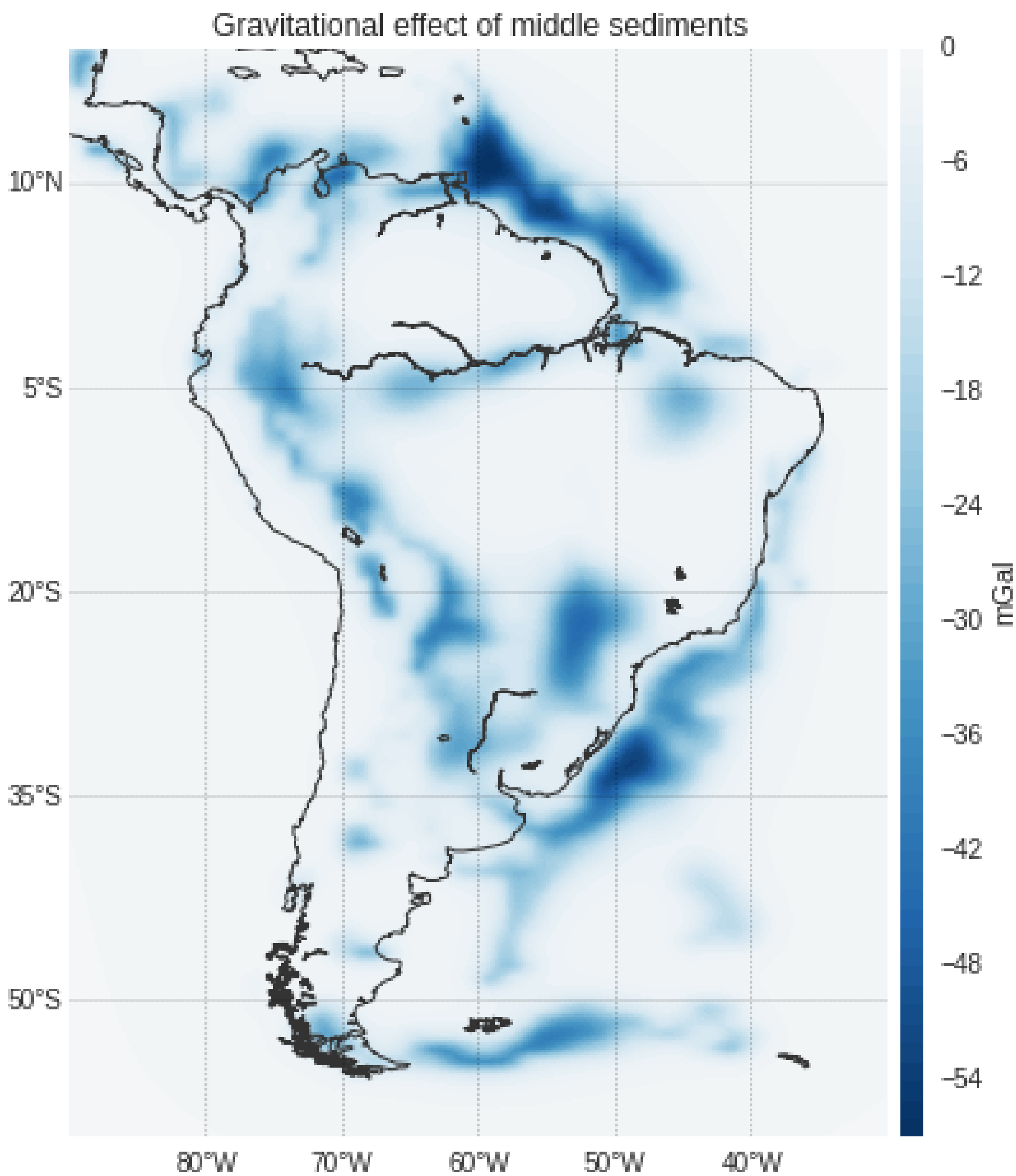
```
CPU times: user 2.02 s, sys: 296 ms, total: 2.32 s
Wall time: 3min 9s
```

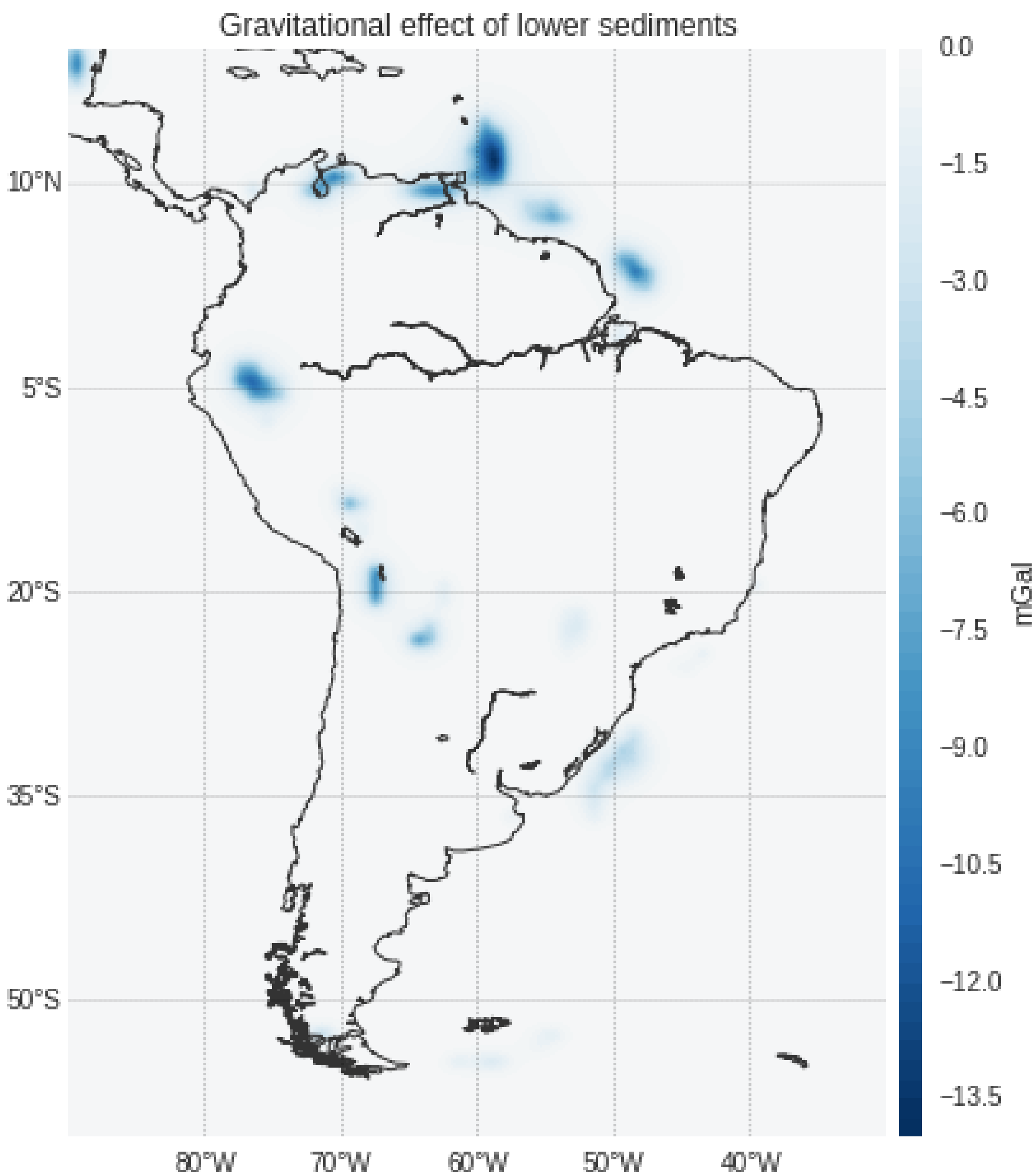
```
In [31]: total_sediment_effect = np.sum(sediment_effects, axis=0)
```

Plot the effect of each layer and the total sediment effect.

```
In [32]: for layer_name, effect in zip(layer_names, sediment_effects):
             plot_data(lat, lon, effect, shape, 'RdBu_r')
             plt.title("Gravitational effect of " + layer_name.lower())
```

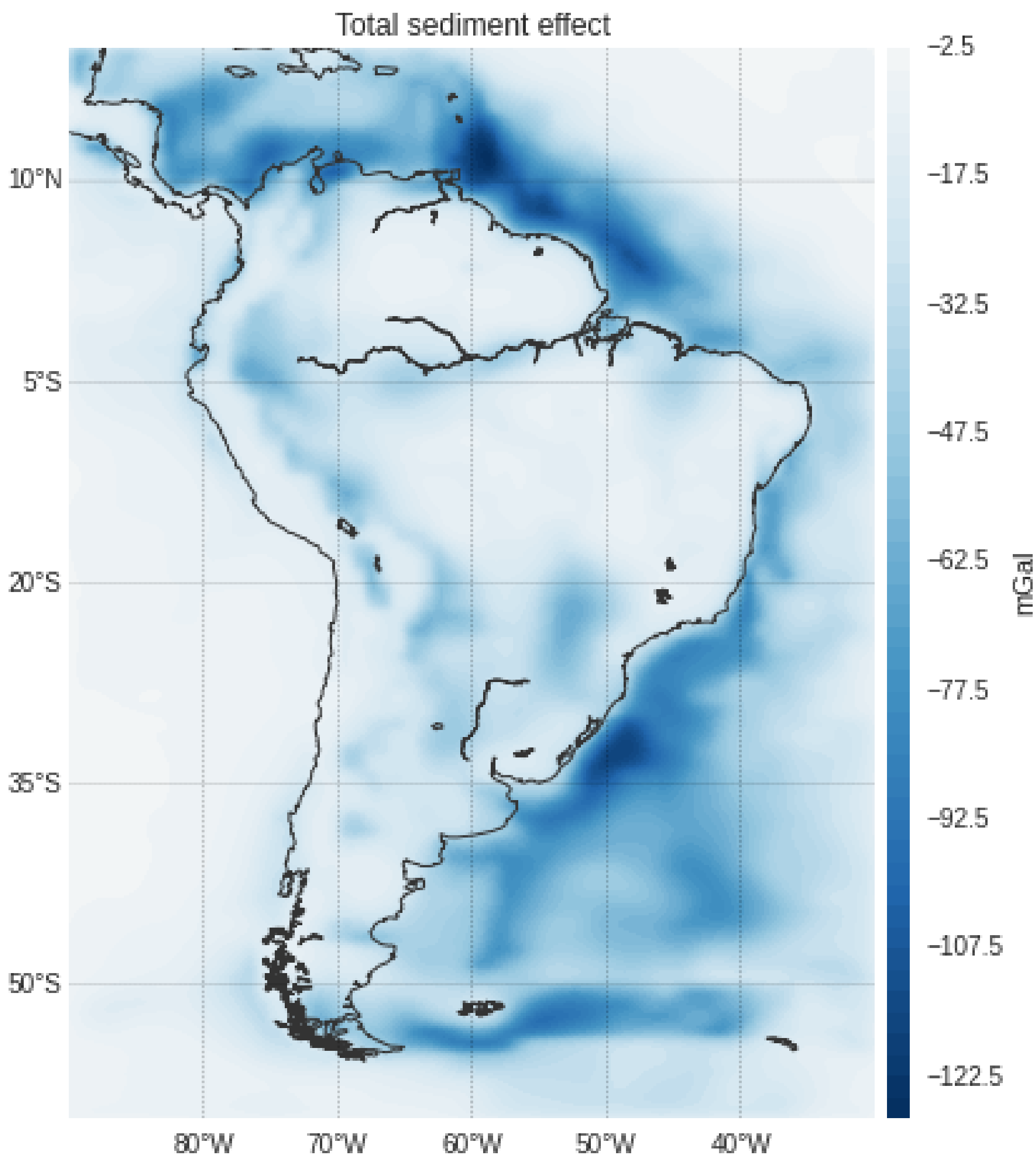

Gravitational effect of upper sediments
10°N
5°S
20°S
35°S
50°S
80°W
70°W
60°W
50°W
40°W
−3
−10
−17
−24
−31
−38
−45
−52
−59
mGal


Gravitational effect of middle sediments
10°N
5°S
20°S
35°S
50°S
80°W
70°W
60°W
50°W
40°W
0
−6
−12
−18
−24
−30
−36
−42
−48
−54
mGal




```
In [33]: plot_data(lat, lon, total_sediment_effect, shape, cmap='RdBu_r')
         plt.title('Total sediment effect')

Out[33]: <matplotlib.text.Text at 0x7f0ce348e5d0>
```

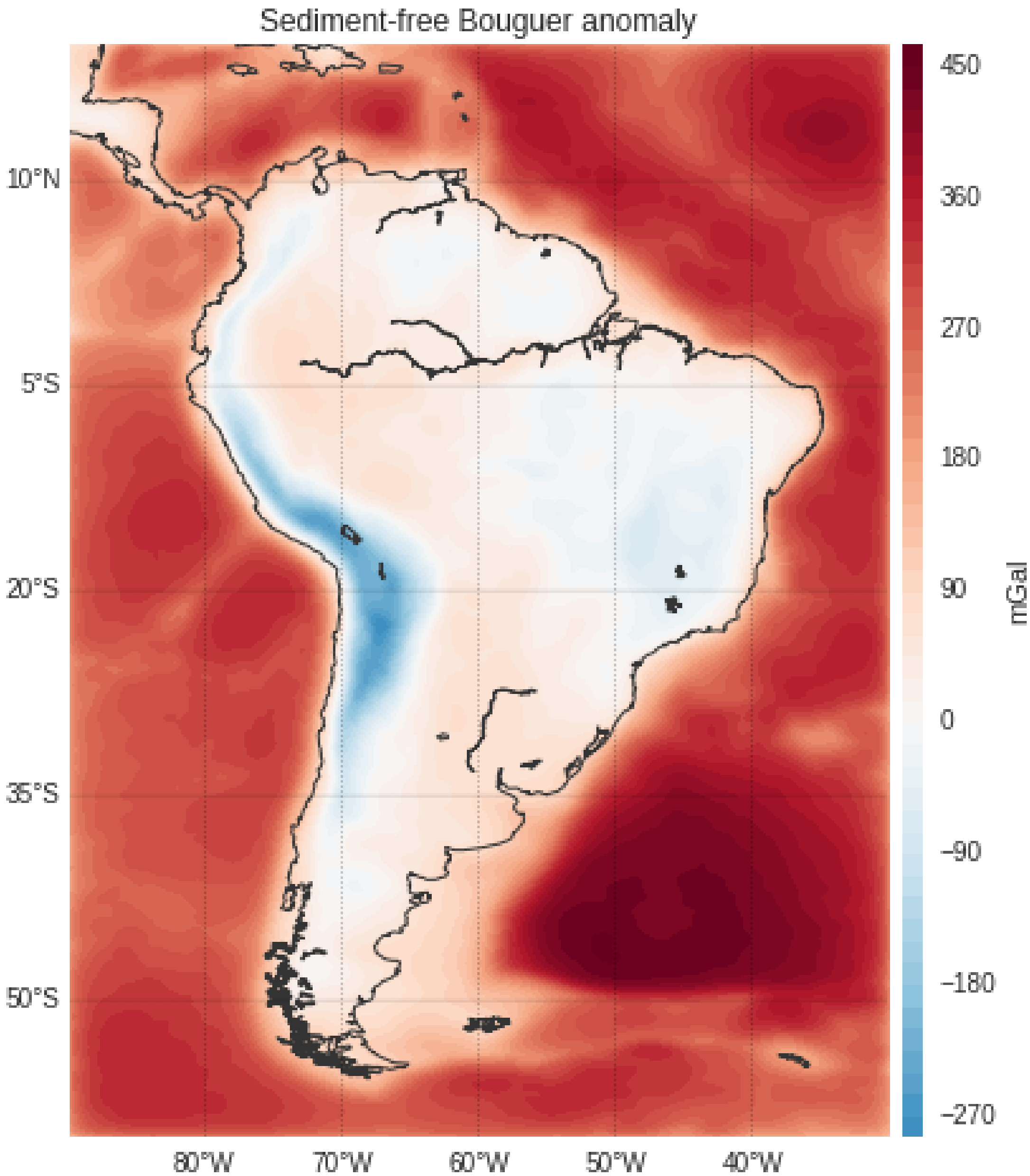


Calculate the sediment-free Bouguer anomaly.

```
In [34]: sedfree_bouguer = bouguer - total_sediment_effect
```

```
In [35]: plot_data(lat, lon, sedfree_bouguer, shape, cmap='RdBu_r')
         plt.title('Sediment-free Bouguer anomaly')
```

```
Out[35]: <matplotlib.text.Text at 0x7f0ce321e890>
```



1.6 Save the data

Save the processed data and all intermediate values to the space-delimited text file ../data/processed-goco5s-data-sam-h50km.txt.

In [36]:
```
now = datetime.datetime.utcnow().strftime('%d %B %Y %H:%M:%S UTC')
header = '\n'.join([
    '# Generated by process-sam-gravity-data.ipynb on {date}'.format(date=now),
    '# shape (nlat, nlon):',
    '# {nlat} {nlon}'.format(nlat=shape[0], nlon=shape[1]),
    '# lat lon height topo gravity disturbance topo_effect bouguer upper_sediment ' + \
```

```
        'middle_sediment lower_sediment total_sediment sedfree_bouguer'
    ])
with open('../data/processed-goco5s-data-sam-h50km.txt', 'w') as f:
    f.write(header)
    np.savetxt(f, np.c_[lat, lon, height, topo, grav, disturbance, topo_effect,
                        bouguer,sediment_effects[0], sediment_effects[1],
                        sediment_effects[2], total_sediment_effect,
                        sedfree_bouguer],
               fmt='%.5f')
```