In [1]:

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm

import os.path

try:
    import pyradi.ryplot as ryplot
    pyradiryplotImported = True
except ImportError:
    pyradiryplotImported = False


# get rbf from here:
#   https://github.com/treverhines/RBF.git
#   https://github.com/NelisW/RBF (forked from above)
# rbf has a poly.c file that must be compiled to a poly.pyx file.
# it seem that different versions of Python require recompile of the file.
# to do this cd to the folder that contains setup.py (RBF/) and then
#   python setup.py install
# this will install rbf in the Lib/sitepackages folder, ready for work.


from rbf.interpolate import RBFInterpolant
```

Using radial basis functions for smoothing/interpolation on a sphere

This work was made possible by Trever Hines, the man behind https://github.com/treverhines/RBF.git. Trever Hines kindly reworked and updated his RBF module to perform interpolation on a sphere. The data is converted from azimuth and polar/elevation angles to (x,y,z), which are then interpolated over in the Cartesian domain. His approach obtain the symmetry around the sphere and does not require any special spherical kernels. Trever's example script is shown below.

See here for a general introduction on interpolation by radial basis functions: https://github.com/NelisW/PythonNotesToSelf/blob/master/RBF-Interpolation.ipynb

Scipy has some RBF interpolation functionality but not quite what we need here: http://scipy-cookbook.readthedocs.io/items/RadialBasisFunctions.html

There is a more powerful radial base function package available here:
https://github.com/treverhines/RBF
https://rbf.readthedocs.io/en/latest/
which provides additional capabilities not available in the scipy package.

Note It appears that Hines follow the Scipy convention for $\epsilon = \sigma$ in a Gauss function.

Radial basis functions can be used for smoothing/interpolating scattered/unstructured data in n-dimensions, but should be used with caution for extrapolation outside of the observed data range.

In [2]:

```
# seed for random number generator
rseed = 1
# number of observation points
nobs = 100
# show input data markers in interpolated contour graphs
showMarkers = True
```

General support functions

In [3]:

```
def spherical_to_cartesian(azim,elev):
    """Convert azimuth/elevation angles to (x,y,z) on a sphere, Numpy version.

    Azimuth in the domain $[0..2\pi]$
    Elevation in the domain $[-\pi/2..\pi/2]$

    Returns [x,y,z] values on a sphere
    """
    x = np.sin(elev+np.pi/2)*np.cos(azim)
    y = np.sin(elev+np.pi/2)*np.sin(azim)
    z = np.cos(elev+np.pi/2)
    return np.array([x,y,z]).T
```

In [4]:

```
def spherical_to_cartesianDf(row):
    """Convert azimuth/elevation angles to (x,y,z) on a sphere, Pandas version.

    Azimuth in the domain $[0..2\pi]$
    Elevation in the domain $[-\pi/2..\pi/2]$

    Returns new columns for [x,y,z] values on a sphere
    """
    x = np.sin(row['Elev']+np.pi/2) * np.cos(row['Azim'])
    y = np.sin(row['Elev']+np.pi/2) * np.sin(row['Azim'])
    z = np.cos(row['Elev'])
    return pd.Series({'x':x, 'y':y, 'z':z})
```

In [5]:

```
def makeRegGrid(gint_azim, gint_elev):
    """Make a regular grid of points in mesh form, given azimuth and elevation coord

    Azimuth in the domain $[0..2\pi]$
    Elevation in the domain $[-\pi/2..\pi/2]$

    Inputs:
        gint_azim azimuth grid interval in degrees
        gint_elev elevation grid interval in degrees

    Returns
        msh_azim 2D mesh grid varying along azimuth
        msh_elev 2D mesh grid varying along elevation

    """

    num_azim = int(360 / gint_azim + 1)
    num_elev = int(180 / gint_elev + 1)
    grd_azimV = np.linspace(0.0,2*np.pi,num_azim)
    grd_elevV = np.linspace(-np.pi/2,np.pi/2,num_elev)
    msh_azim,msh_elev = np.meshgrid(grd_azimV,grd_elevV)

    return msh_azim,msh_elev,grd_azimV,grd_elevV
```

In [6]:

```
def true_function(msh_azim,msh_elev,ftype='original'):
    """Create different test targets
    """

    if 'asym-cardoid' in ftype:
        razim = 1 + np.sin(msh_azim)
        relev = np.cos(msh_elev) * np.abs(np.tan(1.3* np.pi/2+0.2*msh_elev))
        out = 0.1 + razim * relev
    elif 'basic' in ftype:
        out = np.cos(msh_azim) * np.cos(msh_elev)
    else:
        # create some arbitary function that we want to recontruct with interpolatio
        cart = spherical_to_cartesian(msh_azim,msh_elev)
        out = (np.cos(cart[:,2] - 1.0) *
             np.cos(cart[:,0] - 1.0) *
             np.cos(cart[:,1] - 1.0))
    return out
```

The requirement near-equal or uniform spacing on the sphere means limits the construction to a geodesic structure with limited choice of the number of points. The grids were originally created as triangulated icosahedra in Meshmixer, but can also be created in Blender as the vertices of an icosphere. Export the object to the wavefront obj format and extract the vertices.

In [7]:

```
def makeTestSet(nobs,ftype,rseed=1,doRandom=True):
    if doRandom:
        np.random.seed(rseed)
        # make the observation points in spherical coordinates
        obs_azim = np.random.uniform(0.0,2*np.pi,nobs)
        obs_elev = np.random.uniform(-np.pi/2,np.pi/2,nobs)
        # get the catesian coordinates for the observation points
        obs_cart = spherical_to_cartesian(obs_azim,obs_elev)
    else:
        # this file is a regular equally-spaced geodesic
#         obs_cart = np.loadtxt('data/vertexsphere_2_162.txt')
        # this file has the same as above but extra vertices around planes and x axi
        obs_cart = np.loadtxt('data/compositesphere.txt')
        # avoid divide by zero
        obs_cart += np.finfo(float).eps
        obs_azim = np.arctan2(obs_cart[:,1],0.001+obs_cart[:,0])
        obs_elev = np.arctan2( obs_cart[:,2], np.sqrt(obs_cart[:,1]**2 + obs_cart[:,
#         obs_elev = np.arctan2(obs_cart[:,1]/np.sin(obs_azim),0.001+obs_cart[:,2])
        obs_azim += np.pi
    # get the latent function at the observation points
    obs_vals = true_function(obs_azim,obs_elev,ftype=ftype)
    return obs_cart, obs_vals, obs_azim, obs_elev

#     x = np.sin(elev+np.pi/2)*np.cos(azim)
#     y = np.sin(elev+np.pi/2)*np.sin(azim)
#     z = np.cos(elev+np.pi/2)
```

In [8]:

```
def writeFunction(filename, nobs,ftype='original',rseed=rseed,
                  rscale=[1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0]):
    """Create different test targets, write to file

    The filename as given is preprended by the data function type.

    Inputs:
        filename: filename to be used when writing
```

```
        nobs: number of data samples in the file, randomly created
              azim in [0..2\pi], elel in [\pi/20..\pi/2]
        ftype: data function type, one of 'asym-cardoid','basic' or  'original'
        rseed: seed to be used in random number generator
        rscale: set of scale values to be applied in subsequent columnns. The
                resulting data vector will be multiplied by this scale factor

    Returns
        Nothing.

    """

    obs_cart, obs_vals, obs_azim, obs_elev = makeTestSet(nobs,ftype,rseed,doRandom=F
    out = true_function(obs_azim,obs_elev,ftype=ftype)
    df = pd.DataFrame({'Alti':np.ones(obs_azim.shape),
                       'Azim':obs_azim*180/np.pi,
                       'Elev':obs_elev*180/np.pi,
                       1:out})
    # build the dummy data frame by just scaling out
    for item in rscale[1:]:
        df[item] = df[1] * item

    df.to_csv(ftype+filename, index=False,header=False,sep=' ')
    print(f'Data written to {ftype+filename}')
    print(df.head())


writeFunction('testdata.env',nobs=nobs,ftype='asym-cardoid')
writeFunction('testdata.env',nobs=nobs,ftype='basic')
writeFunction('testdata.env',nobs=nobs,ftype='original')
```

```
Data written to asym-cardoidtestdata.env
   Alti        Azim          Elev         1       1.1       1.2       1.3  \
0   1.0  269.932645  3.171746e+01  0.100001  0.110001  0.120001  0.130001
1   1.0  269.932645 -3.171746e+01  0.100002  0.110002  0.120002  0.130002
2   1.0   90.067355  3.171746e+01  2.685704  2.954275  3.222845  3.491415
3   1.0   90.067355 -3.171746e+01  4.612720  5.073992  5.535264  5.996536
4   1.0  211.687367  1.272222e-14  1.031683  1.134851  1.238019  1.341187

        1.4       1.5       1.6       1.7       1.8       1.9       2.0
0  0.140001  0.150001  0.160001  0.170002  0.180002  0.190002  0.200002
1  0.140002  0.150002  0.160002  0.170003  0.180003  0.190003  0.200003
2  3.759986  4.028556  4.297127  4.565697  4.834267  5.102838  5.371408
3  6.457808  6.919080  7.380352  7.841624  8.302896  8.764168  9.225440
4  1.444356  1.547524  1.650692  1.753860  1.857029  1.960197  2.063365
Data written to basictestdata.env
   Alti        Azim          Elev         1      1.1       1.2       1.3  \
0   1.0  269.932645  3.171746e+01 -0.001000 -0.00110 -0.001200 -0.001300
1   1.0  269.932645 -3.171746e+01 -0.001000 -0.00110 -0.001200 -0.001300
2   1.0   90.067355  3.171746e+01 -0.001000 -0.00110 -0.001200 -0.001300
3   1.0   90.067355 -3.171746e+01 -0.001000 -0.00110 -0.001200 -0.001300
4   1.0  211.687367  1.272222e-14 -0.850927 -0.93602 -1.021112 -1.106205

        1.4      1.5       1.6       1.7       1.8       1.9       2.0
0 -0.001400 -0.00150 -0.001600 -0.001700 -0.001800 -0.001900 -0.002000
1 -0.001400 -0.00150 -0.001600 -0.001700 -0.001800 -0.001900 -0.002000
2 -0.001400 -0.00150 -0.001600 -0.001700 -0.001800 -0.001900 -0.002000
3 -0.001400 -0.00150 -0.001600 -0.001700 -0.001800 -0.001900 -0.002000
4 -1.191298 -1.27639 -1.361483 -1.446576 -1.531669 -1.616761 -1.701854
Data written to originaltestdata.env
   Alti        Azim          Elev         1       1.1       1.2       1.3  \
0   1.0  269.932645  3.171746e+01 -0.006713 -0.007384 -0.008055 -0.008727
1   1.0  269.932645 -3.171746e+01 -0.132561 -0.145817 -0.159073 -0.172329
2   1.0   90.067355  3.171746e+01  0.024032  0.026435  0.028839  0.031242
3   1.0   90.067355 -3.171746e+01  0.474576  0.522034  0.569491  0.616949
```

```
4   1.0  211.687367  1.272222e-14 -0.006796 -0.007476 -0.008156 -0.008835

        1.4       1.5       1.6       1.7       1.8       1.9       2.0
0 -0.009398 -0.010069 -0.010740 -0.011412 -0.012083 -0.012754 -0.013426
1 -0.185585 -0.198841 -0.212097 -0.225353 -0.238610 -0.251866 -0.265122
2  0.033645  0.036048  0.038452  0.040855  0.043258  0.045661  0.048064
3  0.664407  0.711864  0.759322  0.806779  0.854237  0.901695  0.949152
4 -0.009515 -0.010195 -0.010874 -0.011554 -0.012234 -0.012913 -0.013593
```

Demo example from Hines' code

RBF Interpolation on a unit sphere. This is done by converting theta (azimuthal angle) and phi (polar/elevation angle) into cartesian coordinates and then interpolating over R^3.

Credit: script by Trever Hines (treverhines@gmail.com), using his RBF module.
https://github.com/treverhines/RBF
https://rbf.readthedocs.io/en/latest/

The code shown here was converted from Trever's original for polar angle $[0..\pi]$ to elevation angle $[-\pi/2..\pi/2]$.

In [9]:

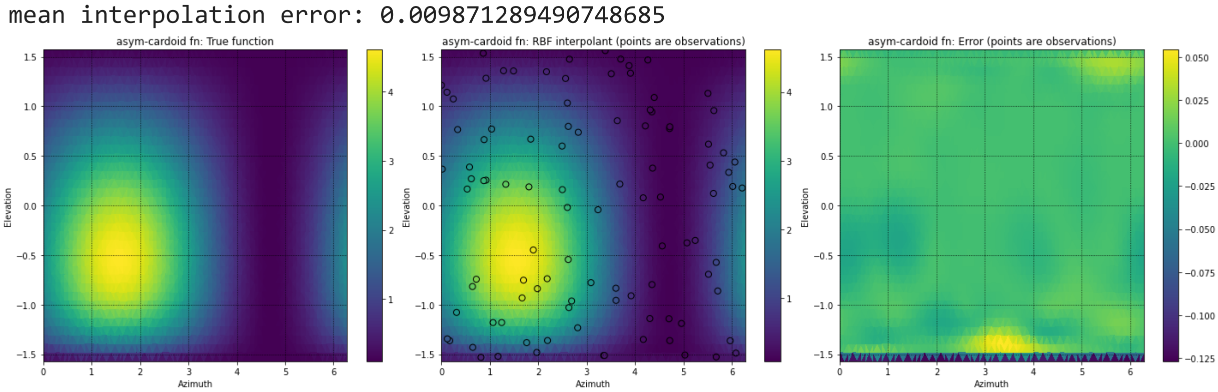
```
def plotResultDemo(msh_azim,msh_elev,val_true,val_itp,obs_azim,obs_elev,obs_vals,ier
    ## PLOTTING
    plt.figure(figsize=(20,6))

    # plot in regular grid
    plt.subplot(1, 3, 1)
    plt.title(f'{ftype} fn: True function')
    p = plt.tripcolor(msh_azim,msh_elev,val_true,cmap='viridis')
    plt.colorbar(p)
    plt.xlabel('Azimuth')
    plt.ylabel('Elevation')
    plt.xlim(0,2*np.pi)
    plt.ylim(-np.pi/2,np.pi/2)
    plt.grid(ls=':',color='k')
    plt.tight_layout()

    # plot the interpolant in spherical coordinates
    plt.subplot(1, 3, 2)
    plt.title(f'{ftype} fn: RBF interpolant (points are observations)')
    # plot the interpolated function
    p = plt.tripcolor(msh_azim,msh_elev,val_itp,cmap='viridis')
    # plot the observations
    plt.scatter(obs_azim,obs_elev,c=obs_vals,
                s=50,edgecolor='k',cmap='viridis',
                vmin=p.get_clim()[0],vmax=p.get_clim()[1])
    plt.colorbar(p)
    plt.xlabel('Azimuth')
    plt.ylabel('Elevation')
    plt.xlim(0,2*np.pi)
    plt.ylim(-np.pi/2,np.pi/2)
    plt.grid()
    plt.grid(ls=':',color='k')
    plt.tight_layout()


    if ierror is not None:
        # plot the interpolant in spherical coordinates
        plt.subplot(1, 3, 3)
        plt.title(f'{ftype} fn: Error (points are observations)')
        # plot the interpolated function
```

```
        p = plt.tripcolor(msh_azim,msh_elev,ierror,cmap='viridis')
        plt.colorbar(p)
        plt.xlabel('Azimuth')
        plt.ylabel('Elevation')
        plt.xlim(0,2*np.pi)
        plt.ylim(-np.pi/2,np.pi/2)
        plt.grid()
        plt.grid(ls=':',color='k')
        plt.tight_layout()
```

In [10]:

```
doExample = True

if doExample:
    np.random.seed(rseed)

    # test function name
    ftype = 'asym-cardoid'
#     ftype = 'original'

    # create the test input set
    obs_cart, obs_vals, obs_azim, obs_elev = makeTestSet(nobs,ftype)

    #create the interpolation set
    msh_azim,msh_elev,_,_ = makeRegGrid(gint_azim=5, gint_elev=5)
    msh_azim = msh_azim.flatten()
    msh_elev = msh_elev.flatten()

    #the true function in spherical coordinates
    val_true = true_function(msh_azim,msh_elev,ftype=ftype)

    # get the Cartesian coordinates for the interpolation points
    cart_itp = spherical_to_cartesian(msh_azim,msh_elev)

    # eps value
    epsvalue = 1
    # basis function is the default phi = phs3, replace if necessary
    # create an RBF interpolant from the cartesian observation points.
    # use the default `RBFInterpolant` parameters here, nothing special.
    I = RBFInterpolant(obs_cart,obs_vals,phi='phs3',eps=epsvalue)
    # evaluate the interpolant on the interpolation points
    val_itp = I(cart_itp)

    # compute and print the mean L2 error
    ierror = val_true - val_itp
    mean_error = np.mean(np.abs(ierror))
    print('mean interpolation error: %s' % mean_error)

    plotResultDemo(msh_azim,msh_elev,val_true,val_itp,obs_azim,obs_elev,obs_vals,ier
```

```
mean interpolation error: 0.009871289490748685
```



Read data from file

This example uses similar code from the above example, but reads the data from a file.

The file contains different sample points, each on a new line. Each line has the following columns:
`'Alti', 'Azim', 'Elev','R100','R90','R80','R70','R60','R50','R40','R30','R20','R10','R00'`
where

1. Alti is the altitude for the data point. The data may contain different spherical sub-datasets, but all such points must be on one altitude.
2. Azim is the azimuth angle for the data point $[0..2\pi]$ in degrees, or alternatively only in a hemisphere $[0..\pi]$ in degrees
3. Elev is the elevation angle for the data point $[-\pi/2..\pi/2]$ in degrees.
4. Then follows eleven columns of values to be interpolated.

The output of the interpolation function below is a separate CSV file for each altitude and lookup column on a regular grid that can be interpolated with most available 2-D interpolation software.

For the purpose of testing one of three files are created by the above function `writeFunction(filename,nobs,ftype,rseed,rscale)`, but in the eventual application the files will be created by some other means.

The data file can contain data for only a hemisphere, in which case the full hemisphere is completed by mirroring.

The different RBF functions available are shown in Trever's docs at https://rbf.readthedocs.io/en/latest/basis.html.

Play around with the `epsvalue` for the RBF, values around unity should work fine.

In [11]:

```
def plotResults2(obs_azim,obs_elev,obs_vals,msh_azim,msh_elev,val_itp):

        plt.figure(figsize=(25,10))

        # plot the input in spherical coordinates
        plt.subplot(1, 2, 1)
        plt.title('Input values')
        # plot the input data
        p = plt.tripcolor(obs_azim,obs_elev,obs_vals,cmap='viridis')
        # plot the observations
        plt.scatter(obs_azim,obs_elev,c=obs_vals,
                    s=50,edgecolor='k',cmap='viridis',
                    vmin=p.get_clim()[0],vmax=p.get_clim()[1])
        plt.colorbar(p)
        plt.xlabel('Azimuth')
        plt.ylabel('Elevation')
        plt.xlim(0,2*np.pi)
        plt.ylim(-np.pi/2,np.pi/2)
        plt.grid()
        plt.grid(ls=':',color='k')
        plt.tight_layout()
```

```
        # plot the interpolant in spherical coordinates
        plt.subplot(1, 2, 2)
        plt.title('RBF interpolant (points are observations)')
        # plot the interpolated function
        p = plt.tripcolor(msh_azim,msh_elev,val_itp,cmap='viridis')
        # plot the observations
        plt.scatter(obs_azim,obs_elev,c=obs_vals,
                    s=50,edgecolor='k',cmap='viridis',
                    vmin=p.get_clim()[0],vmax=p.get_clim()[1])
        plt.colorbar(p)
        plt.xlabel('Azimuth')
        plt.ylabel('Elevation')
        plt.xlim(0,2*np.pi)
        plt.ylim(-np.pi/2,np.pi/2)
        plt.grid()
        plt.grid(ls=':',color='k')
        plt.tight_layout()
```

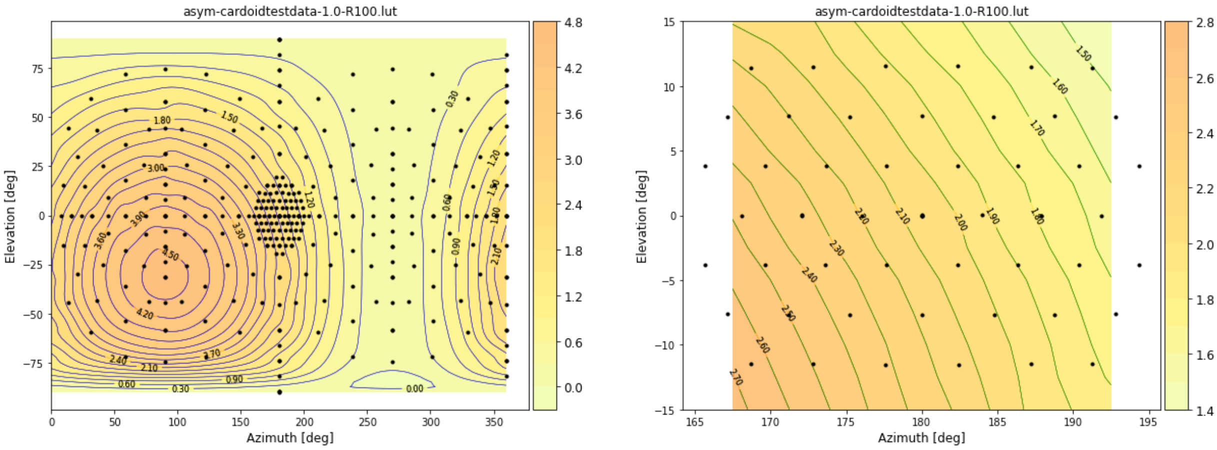
In [12]:

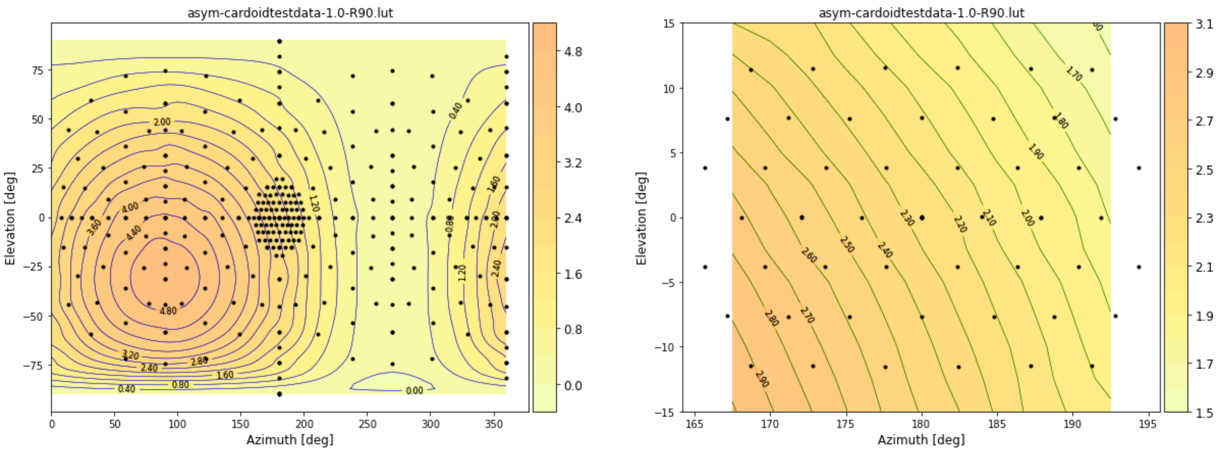
```
# select one of these data files
filename = 'originaltestdata.env'
filename = 'basictestdata.env'
filename = 'asym-cardoidtestdata.env'

#intervals in the output grid in degrees
gint_azim=2.5
gint_elev=2.5
epsvalue = 1
doPlot = False # graphs are written to disk, anyway
randomiseregulargrid = True
usespherical_to_cartesian = True
zoomAzDeg = 15
zoomElDeg = 15

rcolumns = [ 'R100','R90','R80','R70','R60','R50','R40','R30','R20','R10','R00']
columns = ['Alti', 'Azim', 'Elev',] + rcolumns

df = pd.read_csv(filename, names=columns,sep=' ')

# only half hemisphere, fill in the rest
if df[df['Azim']>180].shape[0] == 0:
    dfm = df.copy()
    #remove the two end azim columns, assuming that they already exist
    dfm = dfm[dfm.Azim != 0.]
    dfm = dfm[dfm.Azim != 180.]
    # only positive azim angles
    dfm['Azim'] = 360. - dfm['Azim']
    df = df.append(dfm)
    del dfm

if randomiseregulargrid:
    # add random to azim/elev to break regular structure, to avoid singular matrix e
    randerror = 0.01
    np.random.seed(rseed)
    df['Azim'] += np.random.uniform(-randerror,randerror,df['Azim'].shape[0])
    df['Elev'] += np.random.uniform(-randerror,randerror,df['Elev'].shape[0])

# get unique values
altiUnique = df['Alti'].unique()
azimUnique = df['Azim'].unique()
elevUnique = df['Elev'].unique()

#create the output interpolation support set
msh_azim,msh_elev,grd_azimV,grd_elevV = makeRegGrid(gint_azim=gint_azim, gint_elev=g
```

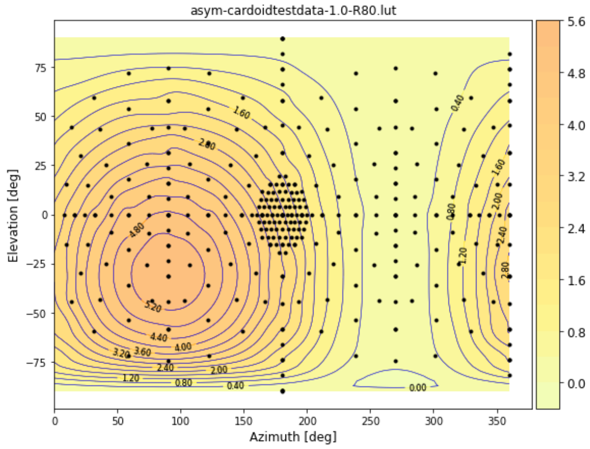
```
# print(msh_azim.shape)
msh_azim = msh_azim.flatten()
msh_elev = msh_elev.flatten()
# get the Cartesian coordinates for the interpolation points
cart_itp = spherical_to_cartesian(msh_azim,msh_elev)

# do for altitudes one at a time
icnt = 0
filenames = []
for alti in altiUnique:
    # and Rx one at a time
#     for rcol in [rcolumns[0]]:
    for rcol in rcolumns:
        icnt += 1
        # extract only the columns needed
        dfs = df[df['Alti']==alti][['Azim', 'Elev',rcol]]
        obs_azim = dfs['Azim'].values * np.pi / 180
        obs_elev = dfs['Elev'].values * np.pi / 180
        obs_vals = dfs[rcol].values

        fbasename = ''.join(os.path.basename(filename).split('.')[:-1])
        ofilename = f'{fbasename}-{alti}-{rcol}.lut'
        print(f'{ofilename}:  processing {rcol}, {dfs.shape} entries')
        filenames.append(ofilename)

        if usespherical_to_cartesian:
            # use data in raw format to calc cartesian values
            obs_cart = spherical_to_cartesian(obs_azim,obs_elev)
        else:
            # use data in df format to calc cartesian values, synced with value
            # this was used experimentally at some point
            dfs[['x', 'y', 'z']] = df.apply(spherical_to_cartesianDf,axis=1)
            obs_cart = dfs[['x', 'y', 'z']].values

        # create an RBF interpolant from the cartesian observation points, default p
        I = RBFInterpolant(obs_cart,obs_vals,phi='phs3',eps=epsvalue)
        # evaluate the interpolant on the interpolation points
        val_itp = I(cart_itp)

        if doPlot and not pyradiryplotImported and icnt == 1:
            plotResults2(obs_azim,obs_elev,obs_vals,msh_azim,msh_elev,val_itp)

        vals = val_itp.reshape(grd_elevV.shape[0],grd_azimV.shape[0])
        np.savetxt(ofilename, vals, delimiter=' ')

        if pyradiryplotImported:
            p = ryplot.Plotter(icnt,1,2,figsize=(20,7),doWarning=False)

            # full set
            p.meshContour(1,grd_azimV*180/np.pi,grd_elevV*180/np.pi,vals,15,
                          ofilename,'Azimuth [deg]','Elevation [deg]',
                          meshCmap=cm.Wistia, cbarshow=True,
                          contLabel=True,contFmt='%.2f')
            if showMarkers:
                markers = ryplot.Markers(markerfacecolor='r', marker='*')
                for az, el in zip(obs_azim,obs_elev):
                    markers.add(az*180/np.pi,el*180/np.pi,markerfacecolor='k', marke
                markers.plot(p.getSubPlot(1))

            # zoomed in subset select by angle around Az=pi, El=0
            selectAz = np.all([ np.abs(grd_azimV-np.pi) <= zoomAzDeg*np.pi/180], axi
            selectEl = np.all([ np.abs(grd_elevV) <= zoomElDeg*np.pi/180], axis=0)
            sAz = grd_azimV[selectAz]
            sEl = grd_elevV[selectEl]
            smsh_azim,smsh_elev = np.meshgrid(selectAz,selectEl)
```

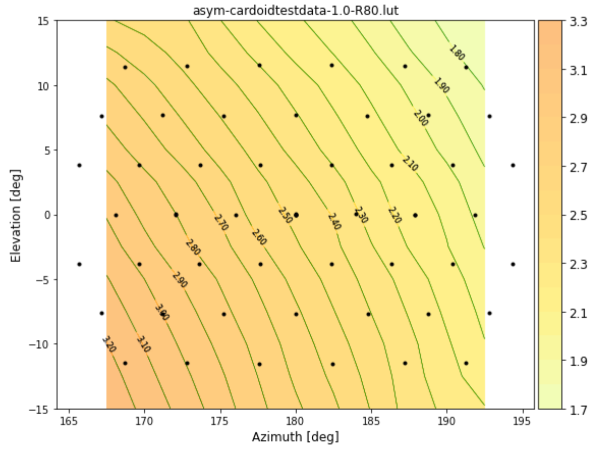
```
            selectVals = smsh_azim * smsh_elev
            sVals = vals[selectVals].reshape(sEl.shape[0],sAz.shape[0])

            p.meshContour(2,sAz*180/np.pi,sEl*180/np.pi,sVals,15,
                          ofilename,'Azimuth [deg]','Elevation [deg]',
                          meshCmap=cm.Wistia, cbarshow=True,
                          contLabel=True,contFmt='%.2f')
            if showMarkers:
                markers = ryplot.Markers(markerfacecolor='r', marker='*')
                for az, el in zip(obs_azim,obs_elev):
                    if  np.abs(az-np.pi) <= zoomAzDeg*np.pi/180 and np.abs(el) <= zo
                        markers.add(az*180/np.pi,el*180/np.pi,markerfacecolor='k', m
                markers.plot(p.getSubPlot(2))

            fbasename = ''.join(os.path.basename(ofilename).split('.')[:-1])
            p.saveFig(fbasename+'.png')
```

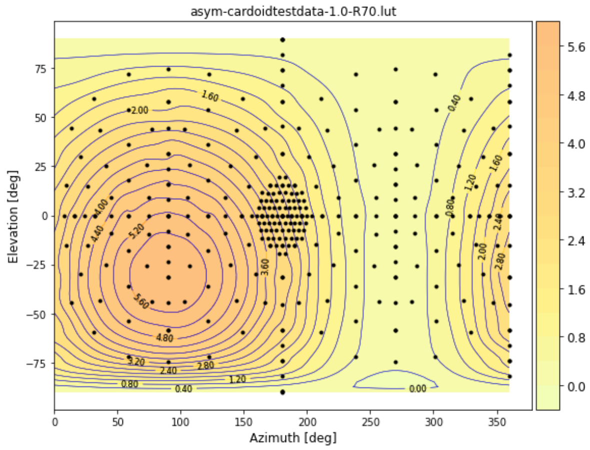
```
asym-cardoidtestdata-1.0-R100.lut:  processing R100, (337, 3) entries
asym-cardoidtestdata-1.0-R90.lut:  processing R90, (337, 3) entries
asym-cardoidtestdata-1.0-R80.lut:  processing R80, (337, 3) entries
asym-cardoidtestdata-1.0-R70.lut:  processing R70, (337, 3) entries
asym-cardoidtestdata-1.0-R60.lut:  processing R60, (337, 3) entries
asym-cardoidtestdata-1.0-R50.lut:  processing R50, (337, 3) entries
asym-cardoidtestdata-1.0-R40.lut:  processing R40, (337, 3) entries
asym-cardoidtestdata-1.0-R30.lut:  processing R30, (337, 3) entries
asym-cardoidtestdata-1.0-R20.lut:  processing R20, (337, 3) entries
asym-cardoidtestdata-1.0-R10.lut:  processing R10, (337, 3) entries
asym-cardoidtestdata-1.0-R00.lut:  processing R00, (337, 3) entries
```

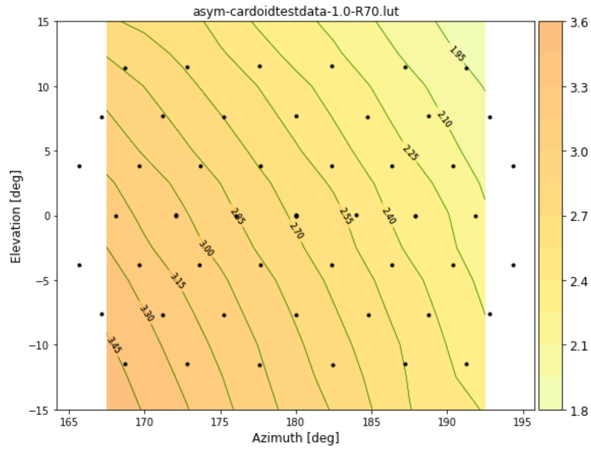


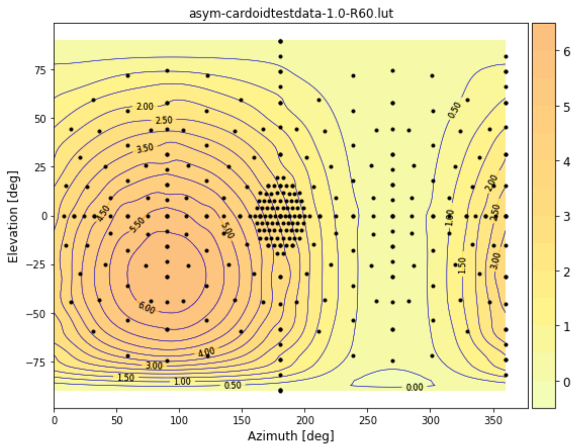


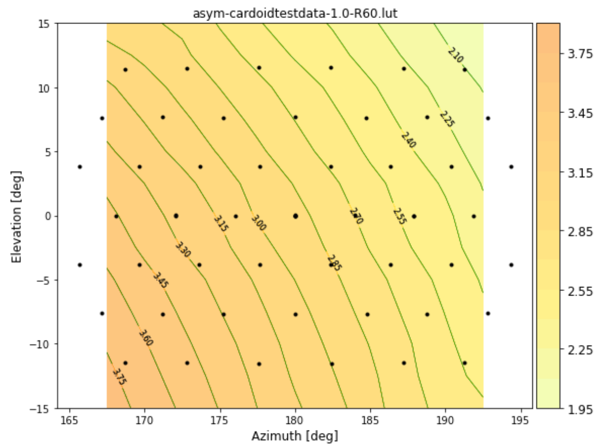

asym-cardoidtestdata-1.0-R80.lut
Elevation [deg]
Azimuth [deg]

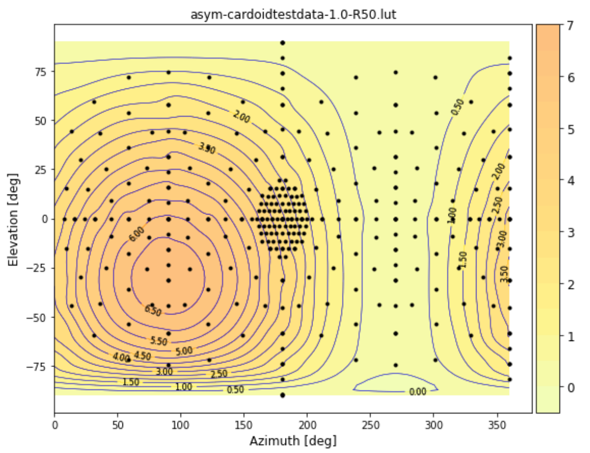

asym-cardoidtestdata-1.0-R80.lut
Elevation [deg]
Azimuth [deg]

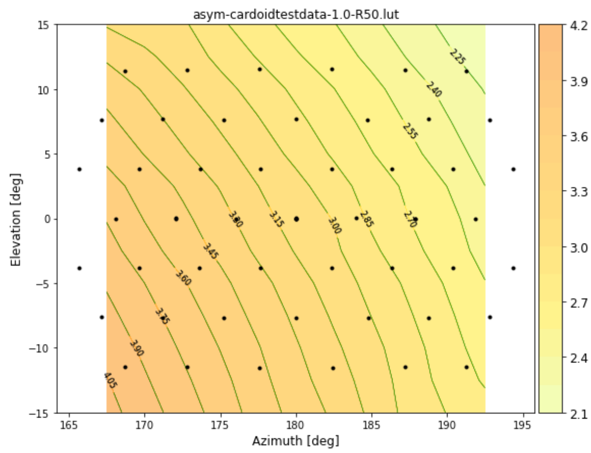

asym-cardoidtestdata-1.0-R70.lut
Elevation [deg]
Azimuth [deg]

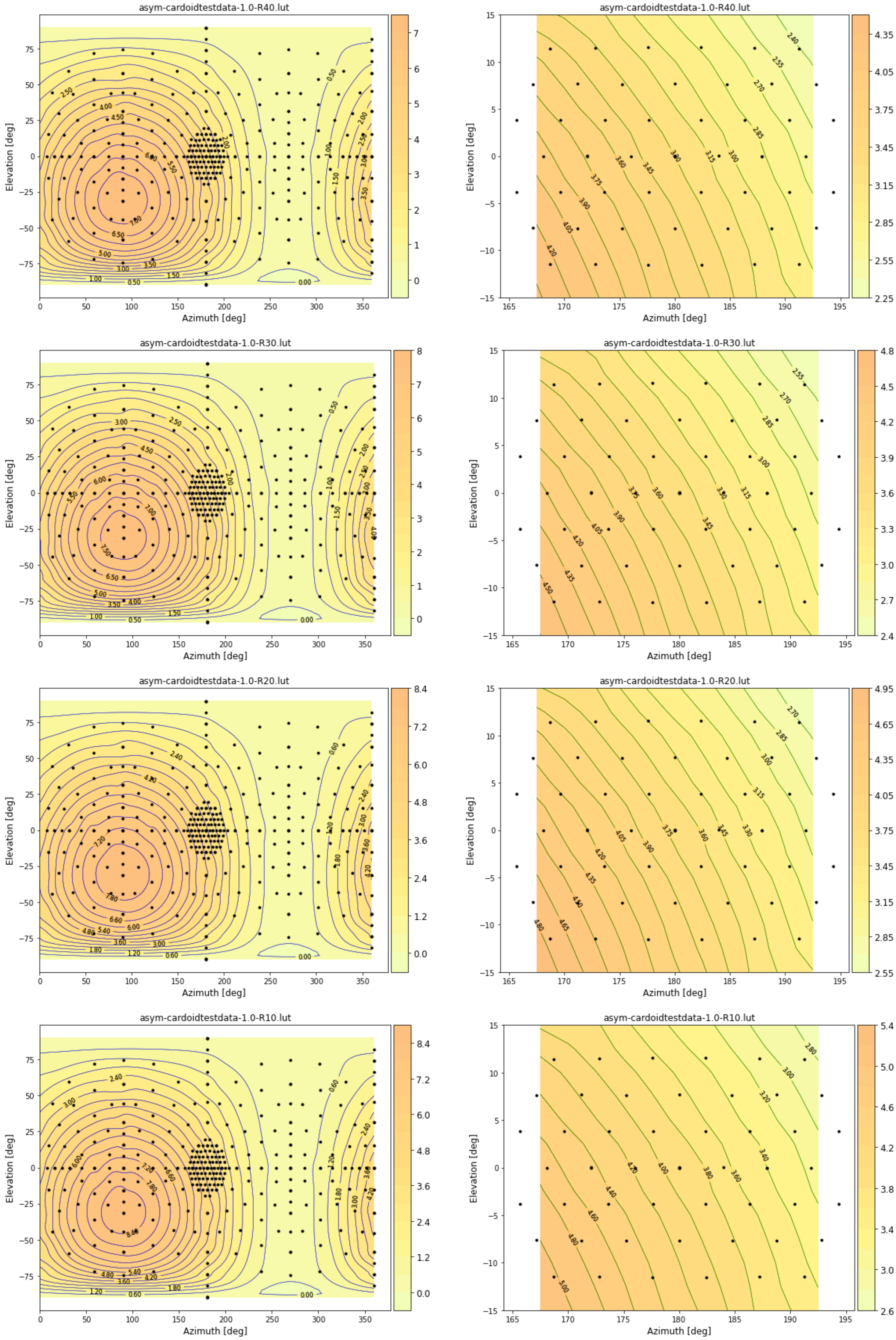

asym-cardoidtestdata-1.0-R70.lut
Elevation [deg]
Azimuth [deg]

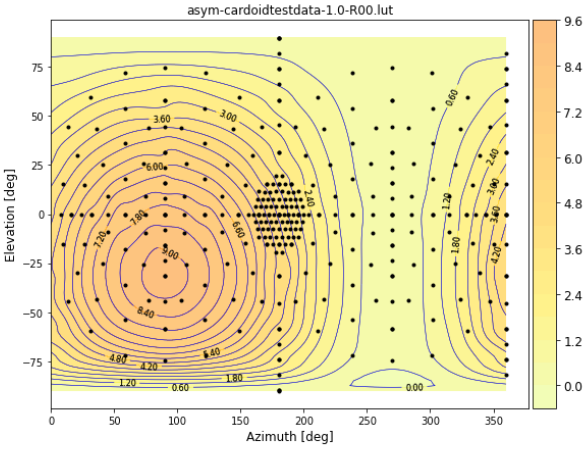

asym-cardoidtestdata-1.0-R60.lut
Elevation [deg]
Azimuth [deg]

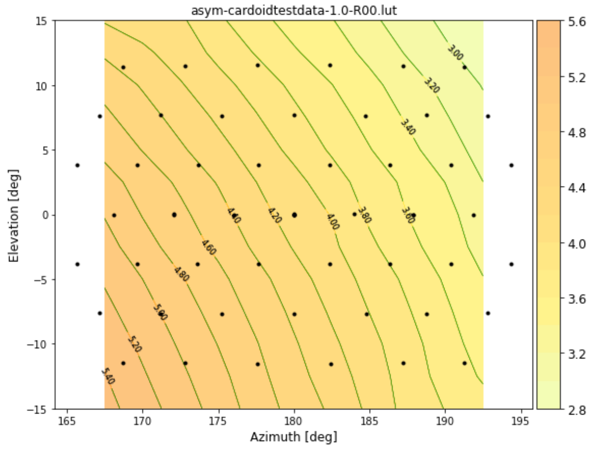

asym-cardoidtestdata-1.0-R60.lut
Elevation [deg]
Azimuth [deg]


asym-cardoidtestdata-1.0-R50.lut
Elevation [deg]
Azimuth [deg]


asym-cardoidtestdata-1.0-R50.lut
Elevation [deg]
Azimuth [deg]


asym-cardoidtestdata-1.0-R40.lut
asym-cardoidtestdata-1.0-R40.lut
asym-cardoidtestdata-1.0-R30.lut
asym-cardoidtestdata-1.0-R30.lut
asym-cardoidtestdata-1.0-R20.lut
asym-cardoidtestdata-1.0-R20.lut
asym-cardoidtestdata-1.0-R10.lut
asym-cardoidtestdata-1.0-R10.lut
Elevation [deg]
Azimuth [deg]





Python and module versions, and dates

In [13]:

```
# to get software versions
# https://github.com/rasbt/watermark
# https://github.com/rasbt/watermark/blob/master/docs/watermark.ipynb
# you only need to do this once
# pip install watermark
# conda install -c conda-forge watermark

%load_ext watermark
%watermark -v -m -p numpy,scipy,rbf,matplotlib -g
```

```
Python implementation: CPython
Python version       : 3.8.3
IPython version      : 7.26.0

numpy     : 1.18.5
scipy     : 1.7.1
rbf       : 2019.1.27+208.gb1ca1fa
matplotlib: 3.4.3

Compiler    : MSC v.1916 64 bit (AMD64)
OS          : Windows
Release     : 10
Machine     : AMD64
Processor   : Intel64 Family 6 Model 165 Stepping 2, GenuineIntel
CPU cores   : 16
Architecture: 64bit

Git hash: fd48ef33030e2832c6811f9c8db998efc502e050
```

In [ ]:

```

```